CONTENTS

Units	Page No.
1. Internet	1–22
2. Core Java	23–145
3. Java Swings	146–168
4. Java Beans and RMI	169–186
5. Java Servlets	187–192

SYLLABUS

C-137

Internet & Java Programming

Unit – I

Internet: Internet, Connecting to Internet: Telephone, Cable, Satellite connection, Choosing an ISP, Introduction to Internet services, E-Mail concepts, Sending and Receiving secure E-Mail, Voice and Video Conferencing.

Unit – II

Core Java: Introduction, Operator, Data type, Variable, Arrays, Control Statements, Methods & Classes, Inheritance, Package and Interface, Exception Handling, Multithread programming, I/O, Java Applet, String handling, Networking, Event handling, Introduction to AWT, AWT controls, Layout managers, Menus, Images, Graphics.

Unit – III

Java Swing: Creating a Swing Applet and Application, Programming using Panes, Pluggable Look and feel, Labels, Text fields, Buttons, Toggle buttons, Checkboxes, Radio Buttons, View ports, Scroll Panes, Scroll Bars, Lists, Combo box, Progress, Menu and Toolbars, Layered Panes, Tabbed, Tabbed Panes, Split Panes, Layouts, Windows, Dialog Boxes, Inner frame.

JDBC: The connectivity Model, JDBC/ODBC Bridge, java.sql package, connectivity to remote database, navigating through multiple rows retrieved from a database.

Unit – IV

Java Beans: Application Builder tools, the bean developer kit(BDK), JAR files, Introduction, Developing a simple bean, using Bound properties, The Java Beans API, Session Beans, Entity Beans, Introduction to Enterprise Java beans(EJB)

Unit – V

Java Servlets: Servlet basics, Servlet API basic, Life cycle of a Servlet, Running Servlet, Debugging Servlets, Thread-safe Servlets, HTTP Redirects, Cookies, Introduction to Java Server Pages(JSP).

UNIT 1 INTERNET

NOTES

★ STRUCTURE ★

1.0 Learning Objectives
1.1 Introduction
1.2 World Wide Web (WWW)
1.3 Choosing an ISP
1.4 Internet Services
1.5 How does One Send an E-mail?
1.6 ETP Connections
1.7 E-mail Concepts
1.8 Sending and Receiving Secure E-mail
1.9 E-mail:Common Problems and Solutions
1.10 Voice and Video Conferencing.
- *Summary*
- *Review Questions*
- *Further Readings*

1.0 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- discuss history, components and organization of Internet.
- illustrate methods of connecting to Internet.
- define World Wide Web (WWW) and E-mail.
- describe process of sending and receiving secure E-mail.
- state about voice and video conferencing.

1.1 INTRODUCTION

History of Internet

The Internet as you watching today, its origin is the US Defense Department project in 1969. The subject of the project was wartime digital communications. At that time the telephone system was the only

communications system in use. A major problem had been identified in its design like its dependence on switching stations that could be targeted during an attack. Only voice can be transmitted through, there was no way to send text data electronically. The Defense Advanced Research Projects Agency (DARPA) launched the DARPA Internet Program to design a computer based digital communication system . In 1975, DARPA declared the project as a success and handed its management over to the Defense Communications Agency. Several of today's internet protocols like TCP or IP were stable by 1980, and adopted throughout ARPANET by 1983. In August 1983, there were 562 registered ARPANET hosts.

Driven largely by the development of the PC and LAN technology, subnetting was standardized in 1985 when RFC 950 was released. LAN technology made the idea of a "catenet" feasible, an internet work of networks. Subnetting opened the possibilities of interconnecting LANs with WANs. Domain naming was stable by 1987 when RFC 1034 was released. Until then, hostnames were mapped to IP address using static tables, but the Internet's exponential growth had made this practice infeasible.

In the early 90s the World Wide Web (WWW) has been one of Internet's most exciting recent developments. The idea of hypertext has been around for more than a decade, but in 1989 a team at the European Center for Particle Research (CERN) in Switzerland developed a set of protocols for transferring hypertext via the Internet. In the early 1990s it was enhanced by a team at the National Center for Supercomputing Applications (NCSA) at the University of Illinois, one of NSF's supercomputer centers. The result was NCSA Mosaic, a graphical, point-and-click hypertext browser that made Internet easy. The resulting explosion in "Web sites" drove the Internet into the public eye.

The real progress of Internet started in the 1990s and today we all are familiar with the strength of Internet as commercial, social and political tool. It has already become part of the international vocabulary, and seems headed for even greater prominence. It has been accepted by the business community like banks, with a resulting explosion of service providers, consultants, books, and media coverage. For the next few years, the Internet will almost certainly be content-driven. Although new protocols like WAP by which you can access Internet on your mobile phone are always under development, we have barely begun to explore the potential of Internet in form of E-mail, E-commerce, E-governance etc. The most popular form of Internet is the World Wide Web (WWW), with its potential for simple on-line access to almost any information you need at your home. Broad band, Mobile Internet, IPTV are new buzzword in market for fast and easy access for Internet.

Organization of Internet

The Internet is an organized international collaboration of autonomous, interconnected networks, supports host-to-host communication through protocols and procedures defined by Internet Standards. Nobody really owns or controls the Internet. Rather, participation in the Internet derives from

NOTES

voluntary participation in Internet Standards. Many Internet providers not only adhere to these standards, but make access to their networks available to the public. It is the voluntary interconnection and cooperation of these providers that forms the global Internet. In 1996, approximately 300 service providers are interconnected to form the Internet.

The Internet Society (ISOC) is a professional society that is concerned with the growth and evolution of the worldwide Internet, with the way in which the Internet is and can be used, and with the social, political, and technical issues which arise as a result. The Internet Architecture Board (IAB) is a technical advisory group of the ISOC. It is chartered to provide oversight of the architecture of the Internet and its protocols, and to serve, in the context of the Internet standards. The Internet Engineering Steering Group (IESG) is responsible for technical management of IETF activities and the Internet standards process. As part of the ISOC, it administers the process according to the rules and procedures which have been ratified by the ISOC Trustees. The IESG is directly responsible for the actions associated with entry into and movement along the Internet "standards track," including final approval of specifications as Internet Standards.

Components of Internet

A browser (short for web browser) is a computer program (software) that accesses web pages and displays them on your computer screen. The browser is a client program that is reading files from a web server. The browser sends a request to the server for a specific web page—receives the information in small pieces of information—interprets special commands that format the information and displays the web page. These commands are HTML (Hyper Text Markup Language).

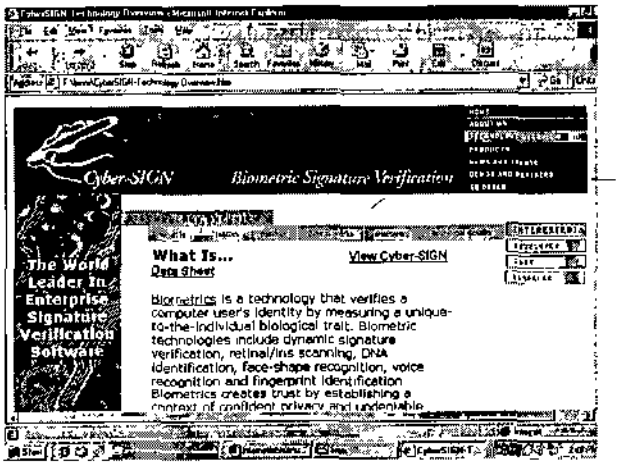

Fig. 1.1. Internet browser screen with an open site

It is the browsers job to interpret the HTML, find all the pictures (or other types of media), take all of these pieces and display the information. Browsers are capable of accepting text, pictures, sound, and movies. Browsers use "helper applications" to generate some of these special effects. Quicktime is an example of this: Quicktime is a program and

a plug-in that allows playing movies through your browser. The two most popular n browsers available are Internet Explorer from Microsoft and Netscape navigator from Sun.

NOTES

1.2 WORLD WIDE WEB (WWW)

The World Wide Web or WWW or W3 refers to the massive collection of multimedia information available over the internet. Although the World Wide Web is often referred to as the internet they are actually two different things. The **internet** is the global collection of computers that transfer information and the wiring that make all of this possible. The **World Wide Web** is a smaller part of the internet. The World Wide Web refers to the documents and related multimedia rich information that uses a specific Internet Protocol called **HTTP**. Since the WWW is a subset of the Internet it stands to reason that the Web could not exist without the internet however the Internet would still be the Internet without the Web (although not as nice). Mosaic was the first client used to access the Web. Mosaic was created by the National Center for Supercomputing Applications (NCSA) and became available to the Internet community in the first half of 1993.

The web talks to computers through the HTTP protocol. More specifically these pages are being displayed through HTML, or the Hyper Text Markup Language, which tells the Web browser how to display the information with it's pictures and text. One of the defining features of the Web is its ability to connect pages to one another with hyperlinks. That's why you can click your way around on the internet. (Or should I say web there?)

Other applications prior to the Mosaic type of browsing relied on a persons knowledge of Internet addressing, hierarchical directory structures, and the application's own set of commands. The browsers that we are familiar with today have simplified all this so we can just click on words —or pictures—to get where we want to go.

This simplistic approach to navigating around the web has been a big factor in the success of the internet. The web with all of its multimedia aspects has become a worldwide phenomena. It doesn't take long to show someone how to click on a link and they are off exploring interests of their choice.

HTML has been developed and expanded from its original set of commands. There are many new exciting technologies like java, DHTML, XML, ASP, JSP, CGI that are changing the web in dynamic tool.

Connecting to Internet

By connecting local, regional, national and international networks, the Internet forms the world's largest network of networks. Computers connected to the Internet work together to transfer information around the world

using servers and clients. A server is a computer that manages the resources on a network and provides a centralized storage for programs and data. The server, also called a host, "serves" files and services out to clients, computers that access the contents of the server. The speed at which data can be transferred between the sender and receiver in a network is called the data transfer rate. Transfer rates are expressed in bits per second (bps). A bit, short for binary digit, is the smallest unit of computerized data. A bit has a value of either 1 or 0 that the computer interprets as "on" or "off" respectively. When calculating Internet-access speeds, it is important to recognize the difference between bits and bytes. There are eight bits in a byte. The small "b" stands for bits, and the big "B" stands for bytes. Transfer speeds are often shown in kilobytes per second (KB/s), and connect speeds are usually quoted in kilobits per second (kbps). For example, if the Web browser is downloading a file at 100 KB/s over a cable modem connection, that is equal to a speed of 800 kbps. Bandwidth is the width of the communication channel, the total volume of traffic that can be transferred across a given transmission line in a given period of time, usually measured in bits per second.

There are four ways of connecting a client computer to the Internet: a dial-up connection using a telephone line or an Integrated Services Digital Network (ISDN), a Digital Subscriber Line (DSL), a cable TV connection and a satellite connection.

Dial-up Connection

A dial-up connection uses the analog telephone line for establishing a temporary communication. Computer's digital signals must be converted to analog signals before they are transmitted over standard telephone lines. This conversion is performed by a modem, a device that modulates (changes into an analog signal) and demodulates (converts an analog signal into a digital signal). Both the sending and receiving ends of a communication channel must have a modem for data transmission to occur. Using a dial-up line to transmit data is similar to using the telephone to make a call. The client computer modem dials the preprogrammed phone number for a user's Internet Service Provider (ISP) and connects to one of the ISP's modems. After the ISP has verified the user's account, a connection is established and data can be transmitted. When either modem hangs up, the communication ends. The advantage of a dial-up line is that it costs no more than a local telephone call. Computers at any two locations can establish a connection using modems and a telephone network, to include wireless modems and cellular telephone connections. The limitation of a connection using the ordinary telephone line is a low speed, 28 kbps. There are dedicated telephone lines that can transmit data at 56 kbps. Most 56 kbps modems connect at a speed less than 46 kbps because of the limitations of analog phone lines and telephone company switches.

NOTES

ISDN

ISDNs are special digital telephone lines that can be used to dial into the Internet at speeds ranging from 64 to 128 kbps. These types of connections are not available everywhere telephone companies have to install special ISDN digital switching equipment. ISDNs require use of a special "digital modem" that sends and receives digital signals over ISDN lines. With an ISDN, the telephone line is divided into three channels (BRI — Basic Rate Interface), two-64 kbps B (bearer) channels that send data and one 16 kbps D (data) channel that sends routing information. This type of access is commonly referred to as 2B+D. To use the ISDN access to the Internet, an ISP has to offer the ISDN access. ISDN lines cost more than normal phone lines, so the telephone rates are usually higher.

Cable TV Connection

Currently most households with cable TV have the option for cable modem Internet access. The cable modem offers a high-speed link at low cost for unlimited, "always connected" access. The connection speeds range from 128 kbps up to 10 mbps (megabits per second). A cable modem is a device that connects to the existing TV cable feed and to an Ethernet network card in the user's PC (also called an NIC—Network Interface Card). The cable network is designed to support the highest speeds in the "downstream" direction, which is from the Internet to the client computer. This downstream speed affects the performance of downloading Web pages and software. The "upstream" bandwidth for data sent from a user's computer to the Internet is typically less, in the range of 200 kbps to 2 mbps. The benefit of the cable modem for Internet access is that, unlike DSL, its performance doesn't depend on distance from the central cable office. However, with the cable TV network, the computer is put on a Local Area Network (LAN) with other users in the neighbourhood and like with any LAN, the performance degrades as usage increases. A more disturbing issue is that of network security. One of the main purposes of a LAN is to allow file sharing among the computers on the LAN. This LAN feature doesn't work well with cable Internet access, as most users do not want neighbours accessing their files. Turning the sharing option off can prevent file sharing. Also, installing the firewall hardware or software may protect from hackers.

DSL (Digital Subscriber Line)

DSL service is a high-speed data service that works over POTS (Plain Old Telephone Service) copper telephone lines and is typically offered by telephone companies without costly installation of a higher-grade cable. DSL uses a different part of the frequency spectrum than analog voice signals, so it can work in conjunction with a standard analog telephone service, providing separate voice and data "channels" on the same line. ADSL (Asymmetric DSL) is the type of DSL that provides different bandwidths

NOTES

in the upstream and downstream directions, giving the user a much bigger "pipe" in the downstream direction. ADSL can support downstream bandwidths of up to 8 mbps and upstream bandwidths of 1.5 mbps. For comparison, a T-1 connection also provides 1.5 mbps. This scheme works well for the typical Internet user; upstream communication is usually small (link requests) compared to downstream communication (Web pages with graphics).

SDSL (Symmetric DSL) offers the same bandwidth capability in both directions. Besides higher bandwidth, some of the advantages of ADSL access from telephone companies are that there are no per-minute charges and the user gets an "always-on" connection for a monthly fee. Most modern computers can be easily equipped to connect to a DSL service. This is accomplished by connecting an ADSL modem to an Ethernet network card in the PC. The downside of DSL includes strict distance limitation that DSL circuits can operate within. As the connection's length increases, the signal quality decreases and the connection speed goes down. DSL services that provide greater that 1.5 mbps require shorter distances to the central office compared to a cable modem that can be located far away from the service provider.

The limit for ADSL service is 18,000 feet (5,460 meters), though for speed and quality of service reasons many ADSL providers place a lower limit on the distances for the service. At the extremes of the distance limits, ADSL customers may see speeds far below the promised maximums, while customers near the central office have the potential for very high speeds. Unlike cable modem technology, DSL provides a point-to-point connection to ISP. DSL proponents claim this technology is both more secure and less prone to local traffic fluctuations than its cable rival. By not sharing a LAN segment with other users, the systems are not as open to intrusion or susceptible to performance degradations related to local traffic.

Satellite Connection

Getting the Internet feed from a satellite is really not all that different from getting TV signals from one. In both cases data is being sent from the satellite to a user's equipment and then translated and decoded. One major limitation of satellite technology is that it can only send data from the satellite to a user's receiver—not the other way. To get around this problem, a separate ISP connection is needed to send data to the Internet, typically over an analog modem. This connection works in conjunction with the satellite feed. As information is requested via the modem line, data are sent back via the satellite. Since most Internet users need high bandwidth from the Web, downstream (typically Web pages and file downloads), and less bandwidth going to the Web, upstream (typically link requests), this scenario of sending upstream data over a standard modem line and downstream data over the high-bandwidth

satellite feed has been effective. The newest satellite technology allows for two-way communications and higher upstream bandwidths. A satellite return channel can be added for traffic bound for the Internet. The upload speeds through this satellite return channel may peak at 128 kbps. Download speeds with this system are up to 400 kbps. Satellite technology has one strong advantage over cable modems and DSL: accessibility. For many it is today's only high-speed option. It can reach areas that are otherwise difficult to establish contact with. The infrastructure exists to provide 400 kbps downstream bandwidth to almost anyone with a 21" satellite dish. It is eight times faster than fastest analog telephone modems and three times faster than ISDN. However, it is not as fast as cable modems or DSL services, which both can provide more than megabits of bandwidth. Also, cable and DSL access methods are cheaper. Equipment required for satellite connection includes installation of a mini-dish satellite receiver and a satellite modem. Like cable modem systems, satellite provides a "shared bandwidth" pipe. This means that download performance may vary depending upon other users of the satellite transponder. Another potential problem can be associated with severe weather. In severe snowstorms and heavy rain, users may experience signal fade.

The general rule about the Internet connection is the faster, the better. The bandwidth and transfer rate determine how quickly pictures, sounds, animation and video clips will be downloaded. Since multimedia and interactivity make the Internet such an exciting tool for information sharing, the speed is the key. Dial-up access provides an easy and inexpensive way for users to connect to the Internet, however, it is a slow-speed technology and most users are no longer satisfied with dial-up or ISDN connections. Fortunately, the broadband access, we once dreamed of, is now possible with TV cable, DSL and satellite links.

1.3 CHOOSING AN ISP

These days, a computer without Internet access is like a car with flat tires—it works, but it won't get you very far. To get online, you need an Internet Service Provider (ISP). You have two general choices: dial up service, which is cheap but slow; and broadband service, which is more expensive and much faster.

Step 1

Determine your needs. Do you want to send and receive E-mail and occasionally surf the Web? An inexpensive dial-up account, which uses regular telephone lines, is probably enough. If you want to connect to your office network, play online games, or download and exchange music and video files, you'll want a speedy broadband connection, such as DSL or cable.

NOTES

Step 2

Find out what hardware is needed. For a dial-up account, a modem is required. Broadband service uses a network interface card (NIC), sometimes called an Ethernet connection. Both are standard equipment on newer computers.

Step 3

Ask your friends and neighbours what Internet service they use and whether they're satisfied with it. Customer service varies from region to region, especially with broadband providers, so seek out a local recommendation.

Step 4

Dial-up users should make sure an ISP has local access numbers (telephone numbers) in your area to avoid long-distance charges.

Step 5

Check for DSL and cable broadband providers in your area. (DSL ISPs use telephone wiring, while cable ISPs use cable TV wiring.) For technical reasons, DSL is sometimes not available in rural or suburban areas.

Step 6

Ask potential broadband providers about package deals. Cable companies may discount Internet access if you buy cable TV service, and phone companies sometimes offer DSL-telephone packages.

Step 7

Consider other broadband options. Try satellite broadband if you live in a rural area. People in some urban areas can get broadband access from a fixed wireless ISP, which use flat, square antennae mounted on your roof to send and receive Internet traffic via radio waves. Some brand-new housing developments have built-in fiber-optic Internet access.

Step 8

Get a T-1 line only if you have heavy use and serious business needs, as this service can cost $500 or more per month.

Step 9

Choose 3G based broadband connection if you want to take advantage of some of its special features, such as content from Time Inc. magazines like Sports Illustrated or People. If you have a broadband connection, you can still use AOL with a "bring your own access" account that costs a little more than half of the usual $24-per-month AOL dial-up price. AOL is useful if you travel a lot with a laptop because you can use its toll-free number almost anywhere.

Tips

- A broadband connection can be shared by several computers. See how to network your computers?
- Some experts recommend choosing an ISP that does not require you to install its own specialized software on your computer.
- With a broadband connection, your computer is always connected to the Internet. Get firewall software to protect against potential snoopers. See how to buy computer software?
- In the fast and furious internet world, even big name ISPs go out of business. This means you'll have to find a new service, and possibly a new E-mail.

Internet Service Providers (ISPs) in India

There are in all 183 operating Internet Service Providers in India. Of them, 41 ISPs (listed below) have all-status. The remaining are particular state-pacific.

Table 1.1

BSNL	CMC	RPG Infotech	Essel Shyam Communications
Sify	Siti Cable Network	Gateway Systems (India)	World Phone Internet Services
VSNL	Guj Info Petro	Hughes Escorts Communications	Astro India Networks
Reliance	Primus Telecommunications India	ERNET India	RailTel Corporation
Data Infosys	GTL	Jumpp India	L&T Finance
HCL Infinet	Primenet Global	Tata Internet Services	Tata Power Broadband
Bhartim Infotel	Pacific Internet India	In2Cable (India)	Reliance Engineering Associates
BG Broad India	Swiftmail Communications	Estel Communication	Bharti Aquanet
Trak Online Net India	Spectra Net	Reach Network India	i2i Enterprise
Tata Teleservices (Maharashtra)	Comsat Max	Gujarat Narmada Valley Fertilizers Corporation	HCL Comnet Systems and Services
Harthway Cable			

NOTES

1.4 INTERNET SERVICES

E-mail

Electronic Mail is a method of sending or receiving a message from a user at a computer to a recipient on another computer.

An E-mail message consists of a **header** and the **body of the message.** The first part, the header, contains the information about where the message has to be sent, and the path that has followed to reach its destination, as well as other information like date, return path, etc. The body is the actual message that is being sent.

An e-mail address has the form **user@subdomain.subdomain.domain**, e.g.sharmamkhld@gmail.com

1.5 HOW DOES ONE SEND AN E-MAIL?

Connect up to a service provider, by dial up, leased line, or having access to some network. **Full network connectivity is NOT needed**

Many different mail programs are available, from the most basic one in UNIX, called "mail" up to some sophisticated ones like "elm", "pine".

These same programs allow you to read your mail, and reply by simply pressing a key. You can also have some aliases, so you do not need to remember the exact address of people to whom you mail quite often

E-mail also provides services to people without full Internet connectivity. For example, many FTP sites (see below) have an electronic address to which you can send a message, asking for certain file, and that file will be mailed to your. Other services that can be done via e-mail are archie lookup, IP address resolver, WHOIS service.

Search Engines

A search engine is a tool that searches web pages, indexes them, and identifies web pages that are related to certain keywords and topics that you ask for. You essentially go to the site that offers a search—type in some key words—get related web addresses to view. There are several different search engines available on the internet. No search engine has a complete directory of every web site and search engines have different algorithms and techniques that they use—so it might pay off to use a combination of search engines available.

There are lots of free search engines. AltaVista is a great search engine that allows you to enter key words. AskJeeves is more of an "ask a question—and I'll find you an answer" type of a search engine. Google is more or less a huge list of bookmarks that allow you to walk through categories until you get to what you are looking for. They all have their place on the internet. They all work a little different—possibly

with different results - so you need to explore a little and find out which method of searching you like best. Searching the internet can be frustrating. However, most users don't take the time to read the instructions. Students don't think they have time to do read a couple of pages that do not directly pertain to accomplishing the impending deadline. Actually, if you know how to use a search engine it can greatly reduce the time and frustration in obtaining that key piece of needed information. So make some time and learn how to use the search engine first.

Google is (in my opinion) one of the best search engines and it's free (as are most search engines). As with any tool, if you learn how to utilize the different features like image search, group search etc., it can become a better tool for you to search on internet.

Chat Groups

Chatting is similar to a telephone conversation except it's text, your picture using Web camera over the internet. When you chat with someone over a network you use the computer, internet, modem, Web camera. Microphone instead of speaking into the telephone chatting is a method of talking to someone over a network in real time. Chatting is different than e-mail in that the messages are synchronous or being sent at the same time. Remember e-mail is sent and stored on a mail server to be read by the recipient at a later time which could be a minute, hour, day or even a week or more.

Chatting software like Yahoo messenger, Rediffbol etc., usually splits your screen into two (one for you and the other for whoever you are chatting with). Lets say that I want some information urgent and I know that my friend is online on his computer. I can invite him for online chat with me to get help. Remember if you ask someone a question in e-mail you may not get your answer in time. At least in a chat session you know there is someone at the other end and their response will be immediate. You can get your free chatid from Gmail, yahoo, rediff or India times website, if you have E-mail account.

FTP and TELNET

Sometimes you want to send large files or electronic data to other computers on network. ftp is a tool by which you can upload and download files via computer networks. You should connect with a ftp server using a user name and password. After login you can upload or download files using simple ftp commands. You can use Graphical ftp software like cuteftp also that are easy to use.

Telnet

The Telnet (telecommunications network) program is intended to provide a remote login or virtual terminal capability across a network. In other

NOTES

words, a user on machine one should be able to log into machine two anywhere on the network, and as far as the user is concerned, it appears that the user is seated in front of machine two. The Telnet service is provided through TCP's port number 23. In your campus if you have your Windows terminal in one lab an your Unix or Linux server at another location using telnet you can connect with that server form your client machine.

Telnet was developed because at one time the only method of enabling one machine to access another machine's resources like hard drives and programs stored there was to establish a link using communications devices such as modems or networks into dedicated serial ports or network adapters.

Usually, Telnet involves a process on the server that accepts incoming requests for a Telnet session. On UNIX systems, this process is called telnetd. On Windows NT and other PC-based operating systems, a Telnet Server program is usually involved. The client (the end doing the calling) runs a program, usually called telnet, that attempts the connection to the server.

The Telnet protocol uses the concept of a network virtual terminal, or NVT, to define both ends of a Telnet connection. Each end of the connection (each NVT) has a logical keyboard and printer. The logical printer can display characters, and the logical keyboard can generate characters. The logical printer is usually a terminal screen, whereas the logical keyboard is usually the user's keyboard, although it could be a file or other input stream.

How to Start Telnet?

To start Telnet, you must provide either the name or the IP address of the machine to be connected with. The name can be used only if the system has a means of resolving the name into its IP address, such as with the Domain Name System. A port name can usually be used to connect to a specific service, but this is used infrequently. For example, to connect to a machine with the IP address 192.168.0.40 , you would enter this command:

Telnet 192.168.0.40

If the server had the name aimcaserver, which was resolvable into its IP address, you could issue this command:

Telnet Aimcaserver

When the connection is established, a user ID and password are requested. You can login with given user ID that is valid on the remote system. A typical connection to a UNIX server looks like this:

NOTES

telnet 192.168.0.40

Trying...

Connected to aimcaserver

Escape character is '^]'.

login: user1

password: xxxxxx

$

As you can see in the preceding code, Telnet tried to connect to the remote system, told you it was connected, then set up the communications parameters between the two systems. When that was done, the login prompt was displayed (as on any UNIX terminal), followed by a password request. If the login and password are enabled, the UNIX shell prompt (a dollar sign) is shown to indicate that the remote machine is now active.

Once the connection is successfully established, your session behaves as though you were on the remote machine, with all valid commands of that operating system. All instructions are relative to the server, so a directory command shows the current directory on the server, not the client. To see the client's directory, you would have to enter command mode. After learning general Unix commands like ls, pwd, cal you can try in your lab.

FTP

File Transfer Protocol, usually called FTP, is a utility for managing files across machines without having to establish a remote session with Telnet. FTP enables you to transfer files back and forth, manage directories, and access electronic mail. The term uploading and downloading are associated with FTP.

FTP uses two TCP channels. TCP port 20 is the data channel, and port 21 is the command channel. FTP is different from most other TCP/IP application programs in that it does use two channels, enabling simultaneous transfer of FTP commands and data. In FTP parlance, the two channels that exist between the two machines are called the protocol interpreter, or PI, and the data transfer process, or DTP. The PI transfers instructions between the two implementations using TCP command channel 21, and the DTP transfers data on TCP data channel 20.

FTP is similar to Telnet in that it uses a server program that runs continuously and a separate program that is executed on the client. On UNIX systems, these programs are named ftpd and ftp, respectively (similar to telnetd and telnet).

FTP Commands

FTP internal protocol commands are four-character ASCII sequences terminated by a newline character. Some of the codes require parameters

after them. One primary advantage to using ASCII characters for commands is that a user can observe the command flow and understand it easily.

1.6 FTP CONNECTIONS

NOTES

FTP is usually started with the name or address of the target machine. As with Telnet, the name must be resolvable into an IP address for the command to succeed. The target machine can also be specified from the FTP command line. For example, to connect to the IP address 192.165.145.40, you would issue this command:

ftp 192.165.145.40

When FTP connects to the destination, you must be able to log into the system as a valid user, using valid user name and password. The following session will show use of ftp on server using a login and password for the remote machine:

C:\> ftp 192.165.145.40
Login : user1
Password : ******
ftp> ls

FTP user commands

FTP Command	Description
ascii	Switch to ASCII transfer mode
binary	Switch to binary transfer mode
cd	Change directory on the server
close	Terminate the connection
del	Delete a file on the server
dir	Display the server directory
get	Fetch a file from the server
help	Display help
lcd	Change directory on the client
mget	Fetch several files from the server
mput	Send several files to the server
open	Connect to a server
put	Send a file to the server
pwd	Display the current server directory
quote	Supply an FTP command directly
quit	Terminate the FTP session

Newsgroups

Newsgroups are a place where people can share information about a certain common topic like sports, movies or IT even, that interests

them. Usenet or Newsgroups are basically a very large number of gathering places for people to discuss topics that they have in common. There are approx 1 over 50,000 newsgroups that range from great topics with great discussion to stuff that shouldn't be on the internet at all. It can be a good way to find out what other people are doing or thinking about a certain topic. You need to search for a topic that interests you and then evaluate the group to verify that the content is worth reading. Like we have group of Amrapali Institute and all the members of this group can share their views any time with all other members.

Internet Blogs

A blog is a frequently updated online personal journal or diary. It is a place to express yourself to the world. A place to share your thoughts and your passions. Really, it's anything you want it to be. For our purposes we'll say that a blog is your own Website that you are going to update on an ongoing basis. Blog is a short form for the word weblog and the two words are used interchangeably.

Here are some more definitions:

"...the first journalistic model that actually harnesses rather than merely exploits the true democratic nature of the web. It's a new medium finally finding a unique voice." — Andrew Sullivan

"[a] collection of posts...short, informal, sometimes controversial, and sometimes deeply personal...with the freshest information at the top." —Meg Hourihan

So you may be think why anyone would want to have their own blog. We believe the answer lies in the fact that every human has a voice and wishes their voice to be heard. The Internet is a medium that is unparalleled in its reach. Never before have average people like you or me been able to reach a global audience with so little trouble. Bloggers have the opportunity of reaching hundreds or even thousands of people each and everyday.

Social Networking

Social Networking is a new way of communicate on internet and 21st century communicates by social networking today. Social networking is the grouping of individuals into specific groups, like small rural communities or a neighbourhood subdivision, if you will. Although social networking is possible in person, especially in the workplace, universities, and high schools, it is most popular online.

When it comes to online social networking, websites are commonly used. These websites are known as social sites like Orkut. Social networking websites function like an online community of internet users. Depending on the website in question, many of these online community members share common interests in hobbies, religion, or politics. Once you are granted access to a social networking website you can begin to socialize.

NOTES

This socialization may include reading the profile pages of other members and possibly even contacting them.

The friends that you can make are just one of the many benefits to social networking online. Another one of those benefits includes diversity because the internet gives individuals from all around the world access to social networking sites. This means that although you are in the United States, you could develop an online friendship with someone in Denmark or India. Not only will you make new friends, but you just might learn a thing or two about new cultures or new languages and learning is always a good thing.

As mentioned, social networking often involves grouping specific individuals or organizations together. While there are a number of social networking websites that focus on particular interests, there are others that do not. The websites without a main focus are often referred to as "traditional" social networking websites and usually have open memberships. This means that anyone can become a member, no matter what their hobbies, beliefs, or views are. However, once you are inside this online community, you can begin to create your own network of friends and eliminate members that do not share common interests or goals.

Once you are well informed and comfortable with your findings, you can begin your search from hundreds of networking communities to join. This can easily be done by performing a standard internet search. Your search will likely return a number of results, including MySpace, Facebook, IBibo, Twitter, Orkut, Yahoo! 360, and Classmates and many more .

1.7 E-MAIL CONCEPTS

Electronic mail (E-mail) is the most popular tool on internet that is used to send or receive electronic message, documents, attachments over the internet. Using E-mail any document can be sent on its destination in a matter of seconds. There are lot of free E-mail service providers on internet like Hotmail, Gmail, Yahoo, Rediffmail etc.,m from where you can get your free Email account with your E-mail-id and password. Everybody that uses the internet has a unique E-mail address. E-mail addresses always have an at sign (@) in them like sharmamkhld@gmail.com. Microsoft also offers a free e-mail program Microsoft Outlook Express which is very good. These require a SMTP (simple mail transfer protocol) server so they will not work with ore many other free web based mail services. Just like the postman must be able to read the address to hand deliver your regular mail to other person, the E-mail address must be exact in order for the computers to deliver it to any one electronic mailbox, that means you must have correct E-mail-id of a person before sending your E-mail. Some popular free E-mail provider sites are:

(a) www.hotmail.com

(b) www.yahoo.co.in

(c) www.rediffmail.com

(d) www.webdunia.com

(e) www.indiatimes.com

(f) www.gmail.com

Here are a few E-mail terms that you must be familiar with to start using E-mail.

Address Book	List of personal contact with their e-mail addresses
Forward	Sends the original message on to someone else
Reply	Sends a e-mail to the sender of the message you are currently reading (sort of like someone sending you a self addressed stamped envelope) this copies the senders e-mail address as well as the text from the message, you type your additional comments if necessary and send the message.
Attachments	Sends actual documents created with other programs along with (or attached to) your E-mail
CC Carbon Copy	Sends the same message to several people (separate their addresses with commas)
BC Blind Carbon Copy	Send Message To Several Addresses Without showing everyone all of the addresses (this is a good way to avoid those annoying messages that are 2 lines but 4 pages of addresses you really could care less about)
Bounced	If a message has a invalid E-mail address it will be marked undeliverable and will bounce back to your E-mail address (the ender)

1.8 SENDING AND RECEIVING SECURE E-MAIL

E-mail lets you send messages anywhere in the world over a computer network. To send or receive email you need an Internet connection and email software (also known as an email client) or a Web browser. Because it's so convenient and easy to use, email has rapidly become part of everyday life; but this widespread use also makes it a primary target for attackers.

NOTES

E-mail: Security Threats and Prevention Tips

- **Bluesnarfing:** Stealing information from mobile devices using a wireless connection
- **Data Theft:** The unauthorized taking or interception of computer-based information
- **Interception:** Receiving E-mail that is directed to another person
- **Malware:** Programs that are designed to harm your computer
- **Newsletters:** Some electronic newsletters are informative, while others are a form of spam
- **Phishing:** Phishing attacks trick you into giving away confidential personal information.
- **Spoofing:** Type of attack where the source of an E-mail is faked
- **Spyware:** Software that sends information from your computer to a third party without your consent
- **Unsolicited Mail:** E-mail that you didn't ask for and typically don't want

1.9 E-MAIL: COMMON PROBLEMS AND SOLUTIONS

Your inbox is bombarded with junk mail

If you get email from an unknown address, this is a sign of spam. You might have made your email address publicly available somewhere. You may want to increase the level of spam filtering in your E-mail settings. For more solutions, visit our page on spam.

All your email is bouncing back

You might have spyware on your machine that is causing all this suspicious activity. Sometimes spyware can block all the E-mails going out of your mailbox and cause them to be returned. To learn more about how to prevent and get rid of it, visit our page on spyware.

E-mail is being sent from your mailbox without your knowledge

It is possible that your email account has been hacked, in which case you might want to report this incident to your E-mail service provider and make a stronger password next time. See our page on strong passwords.

More probably, you might have spyware on your machine that is causing all this suspicious activity. Sometimes spyware sends out automatic emails to people who are in your address book so that they get spyware on their machines, too. To learn more about how to prevent and get rid of it, visit our page on spyware.

You constantly receive promotions from Websites

Sometimes when you visit wellknown Websites or purchase items online, you have to specify your email address in which case they automatically subscribe you on their promotional mailing list. (Look for small print specifying that they will subscribe you.) In this case, you may just want to E-mail them and unsubscribe from such a mailing list. However, you must make sure that they obtained your E-mail address only with your permission. Else, it is spam and you may end up with more spam by contacting them.

MS Outlook strangely re-formats your forwarded e-mails

Outlook reformats your forwarded email because Wordmail is active. When Wordmail is active, Outlook uses MS Word to format your outgoing mail. Word formats the text with more characters per line than other text-based email platforms. Thus, it might appear differently on another platform. To enable/disable Wordmail in Outlook, go to Tools > Options > Mail Format. There, you can toggle the checkbox "Use Microsoft Word to edit mail"

You get E-mails with blocked attachments

Certain file extensions like .exe, .zip, .jar, and .csh are blocked by the email service provider's mail filter. This is done because viruses are commonly passed through files with these extensions. The actual blocking behaviour varies depending on different email providers. If you indeed have to send some attachment with those extensions, you can rename the file to have no extension or some other extension, just for the purpose of sending the attachment, and then make a note to the recipient to rename the file with the proper extension before opening the file.

1.10 VOICE AND VIDEO CONFERENCING

Voice Conferencing or Voice Chat

Voice Conferencing or **Voice Chat** is a modern form of communication used on the Internet. The means of communicating with voice chat is through any of the messengers, mainly Skype, Yahoo! Messenger, AOL Instant Messenger, inSpeak Communicator or Windows Live Messenger. Voice chat has led to a significant increase in distant communications where two or more people from opposite ends of the world can talk almost free of cost.

Voice over IP (VoIP) is a general term for a family of transmission technologies for delivery of voice communications over IP networks such as the Internet or other packet-switched networks. Other terms frequently encountered and synonymous with VoIP are *IP telephony*, *Internet telephony*, *voice over broadband* (VoBB), *broadband telephony*, and *broadband phone*.

Internet telephony refers to communications services—voice, facsimile, and/or voice-messaging applications—that are transported via the Internet, rather than the Public Switched Telephone Network (PSTN). The basic steps involved in originating an Internet telephone call are conversion of the analog voice signal to digital format and compression/translation of the signal into Internet Protocol (IP) packets for transmission over the Internet; the process is reversed at the receiving end.

VoIP systems employ session control protocols to control the set-up and tear-down of calls as well as audio codecs which encode speech allowing transmission over an IP network as digital audio via an audio stream. Codec use is varied between different implementations of VoIP (and often a range of codecs are used); some implementations rely on narrowband and compressed speech, while others support high fidelity stereo codecs.

Video Conferencing

Conducting a conference between two or more participants at different sites by using computer networks to transmit audio and video data. For example, a *point-to-point* (two-person) video conferencing system works much like a video telephone. Each participant has a video camera, microphone, and speakers mounted on his or her computer. As the two participants speak to one another, their voices are carried over the network and delivered to the other's speakers, and whatever images appear in front of the video camera appear in a window on the other participant's monitor.

Multipoint videoconferencing allows three or more participants to sit in a virtual conference room and communicate as if they were sitting right next to each other. Until the mid 90s, the hardware costs made videoconferencing prohibitively expensive for most organizations, but that situation is changing rapidly. Many analysts believe that videoconferencing will be one of the fastest-growing segments of the computer industry in the latter half of the decade.

SUMMARY

- The Internet is an organized international collaboration of autonomous, interconnected networks, supports host-to-host communication through protocols and procedures defined by Internet Standards.
- The World Wide Web or WWW or W3 refers to the massive collection of multimedia information available over the internet.
- It refers to the documents and related multimedia rich information that uses a specific Internet Protocol called **HTTP**.
- Computers connected to the Internet work together to transfer information around the world using servers and clients.

NOTES

- There are four ways of connecting a client computer to the Internet: a dial-up connection using a telephone line or an Integrated Services Digital Network (ISDN), a Digital Subscriber Line (DSL), a cable TV connection and a satellite connection.
- Electronic Mail is a method of sending or receiving a message from a user at a computer to a recipient an another computer.
- FTP is usually started with the name or address of the target machine. As with Telnet, the name must be resolvable into an IP address for the command to succeed. The target machine can also be specified from the FTP command line.
- Voice Conferencing or **Voice Chat** is a modern form of communication used on the Internet. The means of communicating with voice chat is through any of the messengers, mainly Skype, Yahoo! Messenger, AOL Instant Messenger, inSpeak Communicator or Windows Live Messenger.

REVIEW QUESTIONS

1. What is Internet? Describe some applications of Internet.
2. Explain how E-mail are important? Write down name of some popular E-mail service providers.
3. WWW is a collection of trillions of web pages, how you can search some document, image or audio on WWW?
4. Write about ftp and telnet services of Internet.
5. What is World Wide Web (WWW)?
6. Describe different techniques to generate Dynamic HTML pages. How do you make an image clickable in HTML? Give an example.
7. What is net chat?
8. What are news groups and internet blogs?
9. How you can setup an environment for voice chat or voice conferencing?
10. What is VoIP? Explain with example.

FURTHER READINGS

- *Internet and Web Design:* by Ramesh Bangia, Firewall Media.
- *Mastering Java Programes:* by J.B Dixit, Firewall Media.
- *Internet and Java Programming:* by Harish Kumar Taluja. Firewall Media.

UNIT 2 CORE JAVA

NOTES

★ STRUCTURE ★

2.0 Learning Objectives
2.1 Introduction
2.2 Java Features
2.3 Java Statement and Expression
2.4 Java Variables
2.5 Java Operators
2.6 Programming Controls
2.7 Arrays in Java
2.8 Class
2.9 *Object*
1.10 Java Class
1.11 Java Methods
2.12 *Inheritance*
2.13 Java Packages
2.14 Interfaces
2.15 Exception
2.16 Streams
2.17 Java IO
2.18 Input Stream and Reader Classes
2.19 Thread
2.20 Thread States
2.21 Applets vs Applications
2.22 Applet Life cycle
2.23 Parameters in Applet
2.24 Colors, Fonts, Shapes in Applets
2.25 Events in Applet
2.26 Java AWT
2.27 Event Handling
2.28 AWT Events

- *Suummary*
- *Review Questions*
- *Further Readings*

NOTES

2.0 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- discuss history, components and organization of Internet.
- illustrate methods of connecting to Internet.
- define World Wide Web (WWW) and E-mail.
- describe process of sending and receiving secure E-mail.

2.1 INTRODUCTION

In 1990, James Gosling a team member of Sun Microsystems research project was given the task of creating programs to control consumer electronics devices like washing machines, Microwave ovens, Toasters, set top boxes and mobile phones. Gosling and his team of people at Sun Microsystems started this using C++, the language that most programmers were using on that time because of its object-oriented nature. Gosling, however, quickly found that C++ was not suitable for the projects he and his team had change their mind. They ran into trouble many times with complicated nature of C++ such as multiple inheritance of classes and program bugs such as memory leaks , security threats because of pointers, platform dependency etc. Gosling soon decided that he was going design his own, simplified computer language that would avoid all the problems he had with C++. When Gosling completed his language-design project, he had a new programming language that he named **Oak**. (The story goes that the name Oak came to Gosling as he gazed out his office window at an oak tree). The next step for Oak was the Video-On-Demand (VOD) project, in which the language was used as the basis for software that controlled an interactive television system. By the time Sun discovered that the name "Oak" was already claimed and they changed the name to Java, they had a powerful, yet simple, language on their hands.

2.2 JAVA FEATURES

Sun Microsystems defined, Java as "simple, object-oriented, statically typed, compiled, architecture neutral, multi-threaded, garbage collected, robust, secure, and extensible". Sun introduced Java in late 1995, company cofounder **Bill Joy** described the language as follows:

Java is just a small, simple, safe, object-oriented, interpreted or dynamically optimized, byte-coded, architecture-neutral, garbage-collected, multithreaded programming language with a strongly typed exception-handling mechanism for writing distributed, dynamically extensible programs.

Followings are the basic features of Java :

Simple

Java is simple because you will not get many of unnecessary or complex features of other high-level programming languages like C or C++. For example, Java does not support pointers, implicit type casting, structures or unions, operator overloading, templates, header files, or multiple inheritance.

Object-oriented

Just like C++, Java uses classes to organize code into logical modules. At runtime, a program creates objects from the classes. Java classes can inherit from other classes, but multiple inheritance, wherein a class inherits methods and fields from more than one class, is not allowed.

Statically typed

All variables, constants or objects used in a program must be declared before they are used. This enables the Java compiler to locate and report about errors to the programmer.

Compiled and interpreted language

Before you can run a program written in the Java language, the program must be compiled by the Java compiler known as javac. The compilation results in a "byte-code" file that, while similar to a machine-code file, can be executed under any operating system that has a Java interpreter. This interpreter reads in the byte-code file and translates the byte-code commands into machine-language commands that can be directly executed by the machine that's running the Java program. You could say, then, that Java is both a compiled and interpreted language.

Multi-threaded

Java programs can contain multiple threads of execution, which enables programs to handle several tasks concurrently. For example, a multi-threaded program can be in your mobile one thread will run your talk and another will check your talk time or in your internet browser you can open many sites at once in multiple threads. All applications of today and future will need multithread programming.

Garbage collected

Java programs do their own garbage collection, which means that programs are not required to delete objects that they allocate in memory like in C++ using destructors. This reduce many memory-management problems.

Robust

Because the Java interpreter checks all system access performed within a program, Java programs cannot crash the system. Instead, when a

NOTES

serious error is discovered, Java programs create an exception. This exception can be captured and managed by the program without any risk of bringing down the system.

Secure

The Java system not only verifies all memory access but also ensures that no viruses are hitching a ride with a running applet. Because pointers are not supported by the Java language, programs cannot gain access to areas of the system for which they have no authorization.

Platform independent

Platform independence is the capability of the same program to work on different operating systems; Java is completely platform independent. Java's variable types have the same size across all Java development platforms—so an integer is always the same size, no matter which system a Java program was written and compiled on. Also, as shown by the use of applets on the Web, a Java .class file of bytecode instructions can execute on any platform without alteration.

Extensible

Java programs support native methods, which are functions written in another language, usually C++. Support for native methods enables programmers to write functions that may execute faster than the equivalent functions written in Java. Native methods are dynamically linked to the Java program; that is, they are associated with the program at runtime. As the Java language is further refined for speed, native methods will probably be unnecessary.

Well-understood

The Java language is based upon technology that's been developed over many years. For this reason, Java can be quickly and easily understood by anyone with experience with modern programming languages such as C++.

Java Development Kit (JDK)

The Java Development Kit (JDK) is a set of command-line tools that can be used to create Java programs. JDK supports the following operating systems platforms: Microsoft Windows, Sun Solaris Apple MacOS, Unix and Linux. The JDK includes the following tools:

A compiler (javac)

Your source code compiled by javac and a class file in form of byte code generated for you.

An interpreter (java)

To run compiled Java standalone applications.

An applet viewer

To run Java applets.

An archiver

To create compressed archives, and other utilities.
JDK is freely available, you can download it from Sun website.

2.3 JAVA STATEMENT AND EXPRESSION

A statement in Java indicates the simplest tasks you can do in Java, a *statement forms a single Java operation.* All the following are simple Java statements:

```
int poo= 1;
import java.io.*;
System.out.println("This  is a "  + 'name  " " + name1);
Train.engine = true;
```

A **statement** can be a single line or multiple lines, and the Java compiler will be able to read it. The most important thing to remember about Java statements is that each one ends with a semicolon (;). Forget the semicolon, and your Java program will not compile and you will get errors in your program. Java also has compound statements, or blocks, which can be placed wherever you need in a program. Block statements are enclosed by braces either {} or ( ).

An **expression** can be thought of as a programmatic equation. More formally, an expression is a sequence of one or more operands and zero or more operators that produce a result. An example of an expression follows:

marks = tmarks + 13;

In this expression, marks and tmarks are variables, 13 is a literal, and = and + are operators. This expression states that the y variable is divided by 3 using the division operator (/), and the result is stored in x using the assignment operator (=).

2.4 JAVA VARIABLES

Variables are locations in primary memory (RAM) in which your program values can be stored. Each one has a name, a type, and a value. Before you can use a variable, you have to declare it. After it is declared, you can then assign values to it. You can also declare and assign a value to a variable at the same time in Java.

Java actually has three kinds of variables:

1. Instance variables
2. Class variables
3. Local variables.

Instance variables are used to define the attributes of a particular class object. Class variables are similar to instance variables, except their values apply to all that class's instances and to the class itself rather than having different values for each object.

Local variables are declared and used inside method definitions, for example, for index counters in loops, as temporary variables, or to hold values that you need only inside the method definition itself. They can also be used inside blocks. Once the method or block finishes executing, the variable definition and its value destroy. Use local variables to store information needed by a single method and instance variables to store information needed by multiple methods in the object.

Although all three kinds of variables are declared in much the same ways, class and instance variables are accessed and assigned in slightly different ways from local variables.

Variable Declarations

To use any variable in a Java program, you must first declare it. Variable declarations consist of a type and a variable name:

```
int age;
String ename;
```

Variable definitions can go anywhere in a method definition (that is, anywhere a regular Java statement can go), although they are most commonly declared at the beginning of the definition before they are used like:

```
public static void main (String args[])
{
    int count;
    String title;
    boolean draw;
...
}
```

You can declare many variables of the same type in one line: -

```
int x, y, z;
String firstName, LastName;
```

You can also assign each variable an initial value when you declare it:

```
int myAge, mynum = 33;
String myName = "mksharma";
```

```
boolean ismarried = true;
int a = 40, b = 55, c = 66;
```

Local variables must be given some values before they can be used in your Java program otherwise it will not compile if you try to use an unassigned local variable. For this a good idea always to give local variables an initial values like int age=23 or int i=0. Instance and class variable definitions do not have this restriction.

Rules for Variable Naming

Variable names in Java can start with a letter, an underscore (_), or a dollar sign ($). They cannot start with a number. After the first character, your variable names can include any letter or number. Symbols, such as %, *, @, and so on, are often reserved for operators in Java, so be careful when using symbols in variable names.

In addition, as Java is a International language so the Java language uses the Unicode character set. Unicode is a character set definition that not only offers characters in the standard ASCII character set, but also includes several thousand other characters for representing most international alphabets of China or Japan or France. This means that you can use accented characters and other symbols of any international language used on Internet as legal characters in variable names, as long as they have a Unicode character number.

Finally, note that the Java language is case sensitive, which means that uppercase letters are different from lowercase letters. This means that the variable X is different from the variable x, and a sharma is not a Sharma is not a SHARMA. Keep this in mind as you write your own Java programs and when you read Java code other programmer have written.

By convention, Java variables have meaningful names, often made up of several words combined. The first word is lowercase, but all following words have an initial uppercase letter:

```
Button myButton;
long averyBigNumber;
boolean currentWeatherStateOfIndia.
```

Java Data Types

If you want to use a variable in your program, each variable declaration must have a proper data type, which defines what values that variable will hold in future.

The variable data type can be:

1. Any Java primitive data types
2. The name of a class or interface
3. An array.

Primitive Types

Java has eight primitive data types to handle common types for integers, floating-point numbers, characters, and boolean values (true or false). They're called primitive because they're built into the system and are not actual objects, which makes them more efficient to use. Note that these data types are machine-independent, which means that you can rely on their sizes and characteristics to be consistent across your Java programs.

There are four Java integer types, each with a different range of values given in table. All are signed, which means they can hold either positive or negative numbers. Which type you choose for your variables depends on the range of values you expect that variable to hold; if a value becomes too big for the variable type, it is silently trunicated by Java compiler.

Type	*Size*	*Range*
Byte	8 bits	–128 to 127
Short	16 bits	–32,768 to 32,767
Int	32 bits	–2,147,483,648 to 2,147,483,647
Long	64 bits	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Program Sum of two integer values

```
class intExp
{
  public static void main ( String args[] )
{
    int x1 = 118, y1 = 13 , z=0;
    System.out.println("x = " + x1);
    System.out.println("y = " + y1);
    z=x1+y1 ;
    System.out.println("z = " + z);
    }
}
```

Program More operations on two integer values

```
class intArithmetic
{
public static void main (String args[ ])
{
  int x = 17, y = 75;
  System.out.println("x + y = " + (x + y));
```

```
        System.out.println("x – y = " + (x – y));
        System.out.println("x * y = " + (x * y));
        System.out.println("x / y = " + (x / y));
        System.out.println("x % y = " + (x % y));
    }
}
```

Floating Type

Floating-point numbers are used for numbers with has decimal part. Java floating-point numbers follows IEEE 754 (an international standard for defining floating-point numbers and arithmetic). There are two floating-point types: float (32 bits, single precision) and double (64 bits, double precision).

Program **Use of floating data types**

```
class  lessonFloat
 {
 public static void main(String[] s)
  {
  float salary = 9768.60f;
  salary = salary+ 110.45;
  salary = salary – 20.45;
  salary = salary* 3.6;
   System.out.println("Mysalary =" +salary);
  }
}
```

Other Data Types

The char type is used for individual characters. Because Java uses the Unicode character set, the char type has 16 bits of precision, unsigned. Finally, the boolean type can have one of two values, true or false. Note that unlike in other C-like languages, boolean is not a number, nor can it be treated as one. All tests of boolean variables should test for true or false. Note that all the primitive types are written in lowercase.

Example

```
class otherTypes
 {
 public static void main (String[ ] s)
  {
  // More Variables to declare
```

```
    int marks=56;
    float pi=3.14F;
    char initial = 'M';
    String surname = "Sharma";
    }
  }
```

Program

```
class Multipletypes
{
 public static void main (String args[] s)
   {
    short x = 6;
    int y = 4;
    float a = 12.5f;
    float b = 7f;
    System.out.println("x is " + x + ", y is " + y);
    System.out.println("x + y = " + (x + y));
    System.out.println("x - y = " + (x - y));
    System.out.println("x / y = " + (x / y));
    System.out.println("x % y = " + (x % y));
    System.out.println("a is " + a + ", b is " + b);
    System.out.println("a / b = " + (a / b));
 }
 }
```

Output :

```
x is 6, y is 4
x + y = 10
x - y = 2
x / y = 1
x % y = 2
a is 12.5, b is 7
a / b = 1.78571
```

String Data Type

A combination of characters is a string. Strings in Java are instances or objects of the class String. Strings are not simply arrays of characters as they are in C or C++, although they do have many array like characteristics for example, you can test their length, and access and change individual

characters. Because string objects are real objects in Java, they have methods that enable you to combine, test, and modify strings very easily.

When using char type variables you must use single quotes and when using Strings you must use double quotes.

NOTES

Program Use of string data type

```
class stringType
{
public static void main(String[] s)
 {
  String firstName = "Mahesh";
  String lastName = "Sharma";
  System.out.println("My Firstname is " + firstName);
  System.out.println("My Lastname is " + lastName);
 }
}
```

When a string is created or displayed it is enclosed in double quotes, what do we do if we want to display the quotes, then we need to protect them with a\character. Like:

```
class chap2
 {
 public static void main(String[] arguments)
  {
  System.out.println("Amrapali Institute is " + "situated at  " +
                     "\"HALDWANI\"");
  }
 }
```

Other special characters

\' Single quotation mark.

\" Double quotation mark.

\\ Backslash.

\t Tab.

\b Backspace.

\r Carriage Return.

\f Form feed.

\n Newline.

NOTES

To determine the length of a string in Java we use the length() method for the string. Write the given program and test.

```
class stringlength
{
 public static void main(String[] s)
  {
  String firstName = "Vandana";
 System.out.println("The length of " + " firstname is " +  firstName.length());
  }
}
```

To compare strings in Java you use the equals() method. Now type the next program and check the output.

```
class stringcompare
 {
 public static void main(String[] s)
  {
  String firstName = "Vama";
  String firstGuess = "Suhani";
  String secondGuess = "Vama";
  System.out.println("My first daughter name is " +firstGuess + "?");
  System.out.println("Answer: " +  firstName.equals(firstGuess));
   System.out.println("My first daughter name is  "+ secondGuess +
"?");
  System.out.println("Answer: " +  firstName.equals(secondGuess));
  }
 }
```

To convert a string to upper or lower case you can use to Lower Case() or toUpperCase() methods of String class in your program Like:

```
 class convertstring
 {
 public static void main(String[] s)
  {
  String firstName = "Amrapali";
  System.out.println("Lets convert " + firstName  + " to lower case "
                    + firstName.toLowerCase());
  System.out.println("Lets convert " + firstName   + " to upper case"
                    +  firstName.toUpperCase());
  }
 }
```

2.5 JAVA OPERATORS

NOTES

Most of the expressions in Java use operators. Operators are special symbols used to perform operations like arithmetic, assignment, increment and decrement, logical operations etc.

Classification	Operators
Arithmetic	+ – * / %
Relational Operators	< > >= <= == != && || !
Bitwise Operators	& | ^ << >> >>> ~ &= |= ^=
Assignments	= += – = /= %=
Bitwise Assignments	&= |= <<= >>= >>>= ^=
Ternary Operator (if...else shorthand)	?:
Increment	++
Decrement	– –

Arithmetic Operators

Java has five operators for basic arithmetic as given in above table. Each operator takes two operands, one on either side of the operator. The subtraction operator (–) can also be used to negate a single operand. Integer division results in an integer. Because integers don't have decimal fractions, any remainder is ignored. The expression 21/6, for example, results in 3. Modulus (%) gives the remainder once the operands have been evenly divided. For example, 31 % 9 results in 4 because 9 goes into 31 three times, with 4 left over. Note that the result type of most arithmetic operations involving integers is an int regardless of the original type of the operands (shorts and bytes are both automatically converted to int). If either or both operands is of type long, the result is of type long. If one operand is an integer and another is a floating-point number, the result is a floating point.

Program Use of simple arithmetic in Java

```
class optest
    {
    public static void main (String args[])
    {
       short x = 6;
       int y = 14;
       float a = 312.5f;
      float b = 27f;
      System.out.println("x is " + x + ", y is " + y);
```

NOTES

```
        System.out.println("x + y = " + (x + y));
        System.out.println("x - y = " + (x - y));
        System.out.println("x / y = " + (x / y));
        System.out.println("x % y = " + (x % y));
        System.out.println("a is " + a + ", b is " + b);
        System.out.println("a / b = " + (a / b));
    }
```

Assignment Operators

Assignment operators are used in expressions to assign results in another value, you can use them like:

```
m=n=o=10;
```

The right side of an assignment expression is always evaluated before the assignment takes place. This means that expressions such as x = x + 12, 12 is added to the value of x, and then that new value is reassigned to x.

Expression	Meaning
x += y	x = x + y
x –= y	x = x – y
x *= y	x = x * y
x /= y	x = x / y

Increment and Decrement Operators

As in C and C++, the ++ and—operators are used to increment or decrement a variable's value by 1. For example, x++ increments the value of x by 1 just as if you had used the expression x = x + 1. Similarly x—decrements the value of x by 1. Unlike C and C++, Java allows x to be floating point.

These increment and decrement operators can be prefixed or postfixed, that is, the ++ or—can appear before or after the value it increments or decrements. For simple increment or decrement expressions, which one you use isn't overly important. In complex assignments, where you are assigning the result of an increment or decrement expression, which one you use makes a difference.

Take, for example, the following two expressions:

```
y = x++;
y = ++x;
```

These two expressions yield very different results because of the difference between prefix and postfix. When you use postfix operators (x++ or

x—), y gets the value of x before x is changed; using prefix, the value of x is assigned to y after the change has occurred. Listing 3.2 is a Java example of how all this works.

Program Use of ++ and – – operator

NOTES

```
class incredecre
{
    public static void main (String args [] s)
    {
      int x = 6;
      int y = 7;
       System.out.println("x and y are " + x + " and " + y );
      x++;
       System.out.println("x++ results in " + x);
         ++x;
       System.out.println("++x results in " + x);
       System.out.println("Resetting x back to 0.");
         x = 0;
      System.out.println("————————");
      y = x++;
      System.out.println("y = x++ (postfix) results in:");
       System.out.println("x is " + x);
      System.out.println("y is " + y);
        System.out.println("————————");
          y = ++x;
  System.out.println("y = ++x (prefix) results in:");
 System.out.println("x is " + x);
 System.out.println("y is " + y);
 System.out.println("————————");
}
}
```

Comparisons Operators

Java has several expressions for testing equality and magnitude. All of these expressions return a boolean value that is, true or false.

NOTES

Given Table shows the comparison operators.

Operator	Meaning	Example
==	Equal	x == 3
!=	Not equal	x != 3
<	Less than	x < 3
>	Greater than	x > 3
<=	Less than or equal to	x <= 3
>=	Greater than or equal to	x >= 3

Program Use of comparison operator

```
class composer
 {
  public static void main (String args[] s)
 {
    int x =17, y = 13, z = 8;
    System.out.println("x = " + x);
    System.out.println("y = " + y);
    System.out.println("z = " + z);
    System.out.println("x < y = " + (x < y));
    System.out.println("x > z = " + (x > z));
    System.out.println("y <= z = " + (y <= z));
    System.out.println("x >= y = " + (x >= y));
    System.out.println("y == z = " + (y == z));
    System.out.println("x != y = " + (x != z));
  }
}
```

Logical Operators

Expressions that uses the comparison operators can be combined by using logical operators that represent the logical combinations AND, OR, XOR, and logical NOT.

For AND combinations, use either the & or && operators. The entire expression will be true only if both expressions on either side of the operator are also true; if either expression is false, the entire expression is false. For OR expressions, use either | or ||. OR expressions result in true if either or both of the expressions on either side is also true; if both expression operands are false, the expression is false. In addition, there is the XOR operator ^, which returns true only if its operands are different. For NOT, use the ! operator with a single expression argument.

The value of the NOT expression is the negation of the expression; if x is true, !x is false.

Program Use of logical with comparison operators

```
class logicaloper
{
  public static void main (String args[] s)
{
    int x =17, y = 13, z = 8;
    System.out.println("x = " + x);
    System.out.println("y = " + y);
    System.out.println("z = " + z);
    System.out.println("x < y  && x > z = " + (x < y && x>z));
    System.out.println("x > z || y <= z = " + (x > z || y <= z));
  }
}
```

NOTES

Bitwise Operators

The bitwise operators in Java are like C and C++ and used to perform operations on individual bits in integers.

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
<<	Left shift
>>	Right shift
>>>	Zero fill right shift
~	Bitwise complement
<<=	Left shift assignment (x = x << y)
>>=	Right shift assignment (x = x >> y)
>>>=	Zero fill right shift assignment (x = x >>> y)
x&=y	AND assignment (x = x & y)
x|=y	OR assignment (x = x | y)
x^=y	XOR assignment (x = x ^ y)

The bitwise AND, OR, and XOR operators

The bitwise AND, OR, and XOR operators (&, |, and ^) all act on the individual bits of an integer. These operators are sometimes useful when an integer is being used as a bit field. An example of this is

NOTES

when an integer is used to represent a group of binary flags. An int is capable of representing up to 32 different flags, because it is stored in 32 bits.

Program Use of AND, OR, and XOR operators

```
class Bitwiseop
  {
  public static void main (String args[])
    {
    int x = 5, y = 6;
    System.out.println("x = " + x);
    System.out.println("y = " + y);
    System.out.println("x & y = " + (x & y));
    System.out.println("x | y = " + (x | y));
    System.out.println("x ^ y = " + (x ^ y));
  }
}
```

The output of running Bitwiseop:

```
x = 5
y = 6
x & y = 4
x | y = 7
x ^ y = 3
```

To understand this output, you must first understand the binary equivalents of each decimal number. In Bitwise, the variables x and y are set to 5 and 6, which correspond to the binary numbers 0101 and 0110. The bitwise AND operation compares each bit of each number to see whether they are the same. It then sets the resulting bit to 1 if both bits being compared are 1; it sets the resulting bit to 0 otherwise. The result of the bitwise AND operation on these two numbers is 0100 in binary, or decimal 4. The same logic is used for both of the other operators, except that the rules for comparing the bits are different. The bitwise OR operator sets the resulting bit to 1 if either of the bits being compared is 1. For these numbers, the result is 0111 binary, or 7 decimal. Finally, the bitwise XOR operator sets resulting bits to 1 if exactly one of the bits being compared is 1, and 0 otherwise. For these numbers, the result is 0011 binary, or 3 decimal.

The Left-shift, Right-shift, and Zero-fill-right-shift Operators

The left-shift, right-shift, and zero-fill-right-shift operators (<<, >>, and >>>) shift the individual bits of an integer by a specified integer amount.

Following are some examples of how these operators are used:

```
x << 3;
y >> 7;
z >>> 2;
```

In the first example, the individual bits of the integer variable x are shifted to the left three places. In the second example, the bits of y are shifted to the right seven places. Finally, the third example shows z being shifted to the right two places, with zeros shifted into the two leftmost places.

The Negation and Bitwise Complement Operator

The negation unary integer operator (–) is used to change the sign of an integer value. This operator is as simple as it sounds, as indicated by the following example:

```
x =18;
y = –x;
```

In this example, x is assigned the literal value 18 and then is negated and assigned to y. The resulting value of y is –18.

The bitwise complement operator (~), performs a bitwise negation of an integer value. Bitwise negation means that each bit in the number is toggled. In other words, all the binary 0s become 1s and all the binary 1s become 0s. Take a look at an example very similar to the one for the negation operator:

```
x = 8;
y = ~x;
```

In this example, x is assigned the literal value 8 again, but it is bitwise complemented before being assigned to y. What does this mean? Well, without getting into the details of how integers are stored in memory, it means that all the bits of the variable x are flipped, yielding a decimal result of –9. This result has to do with the fact that negative numbers are stored in memory using a method known as two's complement.

Program Use of negation and bitwise complement operator

```
class Negationop
{
  public static void main (String args[])
    {
   int x = 18;
    System.out.println("x = " + x);
    int y = –x;
    System.out.println("y = " + y);
```

```
        }
    }
class BitwiseComplementop
 {
  public static void main (String args[])
    {
    int x = 8;
    System.out.println("x = " + x);
    int y = ~x;
    System.out.println("y = " + y);
  }
}
```

The Conditional Boolean Operator (?:)

The conditional boolean operator (?:) is the most unique of the boolean operators and is worth a closer look. This operator also is known as the ternary operator because it takes three items: a condition and two expressions. The syntax for the conditional operator follows:

Condition? Expression1: Expression2

The Condition, which itself is a boolean, is first evaluated to determine whether it is true or false. If Condition evaluates to a true result, Expression1 is evaluated. If Condition ends up being false, Expression2 is evaluated. To get a better feel for the conditional operator, check out the Conditional program:

Program Use of conditional boolean operator

```
class Conditionalop
  {
  public static void main (String args[])
    {
    int x = 10;
    boolean Even = false;
    System.out.println("x = " + x);
    x = Even ? 14 : 17;
    System.out.println("x = " + x);
  }
}
```

The results of the Conditional program follow:

```
x = 10
x = 17
```

The integer variable x is first assigned a value of 10. The boolean variable Even is assigned a value of false. Using the conditional operator, the value of Even is checked. Because it is false, the second expression of the conditional is used, which results in the value 17 being assigned to x.

2.6 PROGRAMMING CONTROLS

Program flow or programming constructs are used to decide the order in which a program executes its statements. Most program flow is sequential, meaning that the statements are executed one by one in the order in which they appear in the program or method. However, there are Java commands that make your program jump forward or backward, skipping over program code not currently required or to repeat some task again and again. These commands are said to control the program flow. Most programs need decision-making on given data. The program must analyze the data to decide what to do about it, and jump to the appropriate section of code. This decision-making process is very important in computer programming, no useful programs can be written without it.

When a program breaks the sequential flow and jumps to a new section of code, it is called branching. When this branching is based on a decision, the program is performing conditional branching. When no decision-making is involved and the program always branches when it encounters a branching instruction, the program is performing unconditional branching. Unconditional branching is rarely used in modern programs, so here we deals with conditional branching.

Sometime you need to perform a tasks again and again, say n number of times or based upon your choice, that is known as looping. A loop helps you to repeat some action again and again.

Decision-making

The if conditional statement is used when you want to execute different tasks based on a simple test. if conditions are nearly similar to if statements in C++ o C. They contain the keyword if, followed by a boolean test, followed by either a single statement or a block statement to execute if the test is true. Like in given example you will get the message x is smaller than y only if the value of x is less than the value of y:

```
if (x < y)
    System.out.println("x is smaller than y");
```

An else keyword provides the alternative statement to execute if the test is false:

```
if (x < y)
    System.out.println("x is smaller than y");
```

```
else
 System.out.println("y is bigger");
```

Program Use of simple if

```
class simpleif
 {
 public static void main (String[] s)
  {
  int balance = 700;
  if (balance < 1000)
     System.out.println("You cannot withdraw amount");
  }
 }
```

Program Use of simple if .. else

```
class simpleifelse
 {
 public static void main (String[] s)
  {
  int balance = 700;
  if (balance < 1000)
     System.out.println("You cannot withdraw amount");
  else
   System.out.println("You can withdraw amount");
 }
}
```

Program Use of simple if ..else if .. else

```
      class expifelifel
        {
     public static void main (String[] s)
         {
     int balance = 3000;
     if (balance < 1000)
      System.out.println("You cannot withdraw amount");
    else if (balance ==1000)
     System.out.println("You are on minimum limit of balance");
    else
     System.out.println("You have enough balance" + balance );
                              }
                                  }
```

Switch

NOTES

A common programming practice in any language is to test a variable against some value, and if it doesn't match that value, to test it again against a different value, and if it doesn't match that one to make yet another test, and so on until it matches with the right result. Using only if statements in such situation you have to use many if, that will make your program more complex like:

```
 if (oper == '+')
 result=x + y;
else if (oper == '–')
result=x – y;
   else if (oper == '*')
result=x * y;
   else if (oper == '/')
result=x / y;
```

switch is known as shorthand version of the nested if and it is simple as well as easier to read and allows you to group the tests and actions like the similar program can be write as:

```
    switch (oper)
      {
      case '+':
      result=x + y;
         break;
     case '–':
     result=x – y;
                 break;
     case '*':
      result=x * y;
                 break;
     case '/':
      result=x / y;
         break;
  }
```

Program Print even if x has a value of 2, 4, 6, or 8. For other values of x print odd

```
class evenodd
 {
 public static void main (String[] s)
   {
```

```
int x = 6;
 switch (x)
 {
 case 2:
 case 4:
 case 6:
 case 8:
  System.out.println("x is an even number.");
 break;
  default: System.out.println("x is an odd number.");
}
    }
        }
```

Loops

A loop in Java like in C or C++, repeats a statement or block of statements until a condition is matched. There are three types of loops in Java they are the for loop, while loop and the do while loop.

for Loop

for loops are frequently used for simple iterations in which you repeat a block of statements a certain number of times and then stop, but you can use for loops for just about any kind of loop.

The for loop in Java looks roughly like this:

```
for (initialization; test; increment)
{
    statements;
}
```

for loop has three parts:

initialization is an expression that initializes the start of the loop. If you have a loop index variable to keep track how many times the loop has occurred, this expression might declare and initialize it-for example, int i = 0.

Variables that you declare in this part of the for loop are local to the loop itself; they cease existing after the loop is finished executing.

test is the test that occurs before each pass of the loop. The test must be a boolean expression or function that returns a boolean value-for example, i < 15. If the test is true, the loop executes. Once the test is false, the loop stops executing.

increment is any expression or function call. Commonly, the increment is used to change the value of the loop index to bring the state of the loop closer to returning false and completing.

Program Use of for loop to compute sum of first 100 numbers

```
class testoffor
 {
 public static void main (String[] s)
  {
  for (int number = 0; number < 100; number++)
    {
    System.out.println("Number " + number + ".");
    int  sum=sum+number;
      }
    System.out.println("Sum of first 100 Numbers " + sum );
  }
}
```

NOTES

while Loop

The while loop is used to repeat a statement or block of statements as long as a particular condition is true. while loops look like this:

```
while (condition)
{
    bodyOfLoop;
}
```

Program Use of while loop

```
class expwhile
 {
 public static void main (String[] s)
  {
  int num = 0;
  while (num < 100)
    {
    System.out.println("Number " + num + ".");
    num++;
    }
  }
}
```

do...while Loop

The do loop is just like a while loop, except that do executes a given statement or block until the condition is false. The main difference is that while loops test the condition before looping, making it possible

NOTES

that the body of the loop will never execute if the condition is false the first time it's tested. do loops run the body of the loop at least once before testing the condition. do loops look like this:

```
do
{
 bodyOfLoop;
 increment or decrement
} while (condition)
```

Program Use of do..while loop

```
class expdowhile
{
public static void main (String[] s)
  {
  int number = 100;
  do
    {
   System.out.println("Number " + number + ".");
    number – – ;
    }
  while (number >= 1);
  }
}
```

Break

The normal way to exit a loop is for the tested condition to become false but there may be occasions you want to exit earlier and to do so we use the break statement.

Program Use of break

```
class expbreak
 {
 public static void main (String[] s)
  {
  int number = 0;
  do
    {
   number++;
    System.out.println("Number " + number);
```

```
    if (number == 55)
      break;
    }
   while (number < 100);
  }
}
```

Continue

Continue is similar to break except that instead of halting execution of the loop entirely, the loop starts over at the next iteration. For do and while loops, this means that the execution of the block starts over again; for loops, the increment and test expressions are evaluated and then the block is executed. continue is useful when you want to special-case elements within a loop.

Labeled Loops

Both break and continue can have an optional label that tells Java where to go after break. Without a label, break jumps outside the nearest loop to an enclosing loop or to the next statement outside the loop, and continue restarts the enclosing loop. To use a labeled loop, add the label before the initial part of the loop, with a colon between them. Then, when you use break or continue, add the name of the label after the break or continue.

Program Use of labeled loop

```
class LabelTest
  {
      public static void main (String [] s)
       {
            mk:
       for (int i = 1; i <= 5; i++)
        for (int j = 1; j <= 3; j++) {
           System.out.println("i is " + i + ", j is " + j);
          if (( i + j) > 4)
          break mk;
       }
      System.out.println("end of loops");
    }
  },
```

NOTES

2.7 ARRAYS IN JAVA

The basic way to store information in a computer is to use a variable, these are only as good as the number of variables we declare. Say suppose you want to record fee collected by 350 students. You should declare 350 variable to store the fee of each student or you can declare an array to store the 350 fee amount. Arrays in Java, as in other languages, are a way to store collections of items into a single unit. An array is a collection of items. Each slot in the array can hold an object or a primitive value. Arrays in Java are objects that can be treated just like other objects in the language.

To create an array in Java, you use three steps:

1. Declare a variable to hold the array.
2. Create a new array object and assign it to the array variable.
3. Store things in that array.

Declare and Create Arrays

You can declare arrays for any type of information that can be stored as a variable. The first step in creating an array is creating a variable that will hold the array, just as you would any other variable. Array variables indicate the type of object the array will hold (just as they do for any variable) and the name of the array, followed by empty brackets ([]). All the following statements are declaration of an array of variables.

```
string[] babyName;
int[] babyAge[];
```

The above statements created two arrays but there was nothing stored in them, to do so you must use the new statement and specify how many elements are in the array. The following statement creates an array called babyName and sets aside an area to store 15 entries.

```
string[] babyName = new string[15];
```

It is possible to assign values to an array when it is declared like a variable, the following statement creates an array called babyAge and assigns ages to the array.

```
int[] babyAge = {11, 14, 13, 15};
```

The values to be placed in the array are between the { and the } characters and separated by commas. Each element of the array must be of the same type, the number of elements in the array is not specified because the it is set to the number of elements set.

Creating Array Objects

To create an array object and assign it to that variable. There are two ways to do this:

1. Using new
2. Directly initializing the contents of that array.

The first way is to use the new operator to create a new instance of an array:

NOTES

```
String[] snames = new String[10];
```

That line creates a new array of Strings with 10 slots (sometimes called elements). When you create a new array object using new, you must indicate how many slots that array will hold. This line does not put actual String objects in the slots-you'll have to do that later.

Array objects can contain primitive types such as integers or booleans, just as they can contain objects like:

```
int[] tempmarks = new int[99];
```

When you create an array object using new, all its slots are initialized for you (0 for numeric arrays, false for boolean, '\0' for character arrays, and null for objects). You can then assign actual values or objects to the slots in that array. You can also create an array and initialize its contents at the same time. Instead of using new to create the new array object, enclose the elements of the array inside braces, separated by commas like:

String[] cities = {"jawalapur", "panchpuri", "kankhal", "haridwar", "mayapur"};

Accessing Array Elements

Once you have an array with initial values, you can test and change the values in each slot of that array. To get at a value stored within an array, use the array subscript expression ([]) like:

```
myMatrix[subscript];
```

The myMatrix part of this expression is a variable holding an array object, although it can also be an expression that results in an array. The subscript part of the expression, inside the brackets, specifies the number of the slot within the array to access. Array subscripts start with 0, as they do in C and C++. So, an array with 10 elements has 10 array slots accessed using subscript 0 to 9.

Note that all array subscripts are checked when your Java program is run to make sure that they are inside the boundaries of the array (greater than or equal to 0 but less than the array's length). Unlike in C, it is impossible in Java to access or assign a value to an array slot outside the boundaries of the array. Note the following two statements, for example:

```
String[] city  = new String[10];
city[10] = "Roorkee";
```

A program with that last statement in it produces an error at that line when you try to run it. The array stored in city has only 10 slots numbered from 0, the element at subscript 10 doesn't exist so you will get error. To assign an element value to a particular array slot, merely put an assignment statement after the array access expression like:

```
myMatrix[1] = 15;
sentence[0] = "The";
sentence[5] = sentence[0];
```

Program

```
class expArray
 {
 public static void main (String[] s)
  {
  int[] months = {1,2,3,4,5};
   System.out.println("There are " + months.length + "months");
  }
 }
```

Program

```
class MyArrayTest
      {
     String[] firstNames = { "Durgesh", "Mahesh", "Kunwar", "Krishan"};
     String[] lastNames = new String[firstNames.length];
     void printNames() {
       int i = 0;
      System.out.println(firstNames[i]
          + " " + lastNames[i]);
      i++;
       System.out.println(firstNames[i]
         + " " + lastNames[i]);
        i++;
         System.out.println(firstNames[i]
           + " " + lastNames[i]);
       i++;
       System.out.println(firstNames[i]
           + " " + lastNames[i]);
    }
   public static void main (String args[]) {
```

```
    ArrayTest a = new ArrayTest();
     a.printNames();
          System.out.println("————————");
           a.lastNames[0] = "Pant";
          a.lastNames[1] = "Sharma";
         a.lastNames[2] = "Singh";
           a.lastNames[3] = "Motla";
          a.printNames();
}
  }
```

Multidimensional Arrays

Sometimes in your programs there is a use of multidimensional arrays like Matrix operations. Java does not directly support multidimensional arrays. However, you can declare and create an array of arrays (and those arrays can contain arrays, and so on, for however many dimensions you need) and access the arrays as you do in C-style multidimensional arrays like:

```
int matrix[][] = new matrix[4][4];
matrix[0][0] = 11;
matrix[0][1] = 12;
```

Example: Creating a two-dimensional array

Suppose that you need a table-like array that can hold 80 integers in eight columns and 10 rows. First, you'd declare the array like this:

```
int numbers[][];
```

After declaring the array, you need to create it in memory, like this:

```
numbers = new int[8][10];
```

The last step is to initialize the array, probably using nested for loops:

```
for (int x=0; x<8; ++x)
    for (int y=0; y<10; ++y)
        numbers[x][y] = 0;
```

These lines initialize the numbers[][] array to all zeros.

2.8 CLASS

Programs are usually written to solve real-world problems, such as keeping track of employee records in an organization like Amrapali Institute or simulating the workings of a heating system. Although it is possible to solve complex problems by using programs written with only integers and characters data types, but it is more easier to solve

large, complex problems if you can create objects using base classes. In other words, simulating the workings of a heating system is easier if you can create variables that represent rooms, heat sensors, thermostats, and boilers. The closer these variables correspond to reality, the easier it is to write the program. To solve real world problems there is a need to create new data types also known as user defined data types, a class help us to design a new data type with attributes and operations or functions in a single unit. A class defines a data type, much like a struct C. In a computer science sense, a type consists of both a set of states and a set of operations which transition between those states. You can make a new data type by declaring a class. A class is just a collection of variables often of different types combined with a set of related functions. One way to think about a car is as a collection of wheels, doors, seats, windows, and so forth. Another way is to think about what a car can do: It can move, speed up, slow down, stop, park, and so on. A class enables you to encapsulate, or bundle, these various parts and various functions into one collection, which is called an object.

CAR class

wheels

doors

seats

windows

Functions

Move

Start

Stop

Park

CAR c1,c2;

Two objects of CAR , having same attributes and functions.

Encapsulating everything you know about a car into one class has a number of advantages for a programmer. Everything is now in one place, which makes it easy to refer to, copy, and manipulate the data using objects of class.

2.9 OBJECT

In Object oriented programming languages like Java, "object" usually means "an instance of a class." Thus a class defines the behavior of possibly many objects. You can say now an object is an individual instance of a class. You can define an object of your new class just as you define an integer variable. Like:

```
Car c1,              // One object of Car class
```

Employee e1,e2, //Two objects of Employee class
Student s1,s2,s3 // Three objects of student class

CAR class

wheels
doors
seats
windows
Functions
Move
Start
Stop
Park

CAR c1;

Wheels	Move
Doors	Start
Seats	Stop
Windows	Park

NOTES

Example

A class employee with following attributes empno, empname, empdept, empsalary and functions join, computesalary, printdata, printsalary.

Class employee

Empno—integer
Empname—string
Empdept—string
Empsalary—integer

Functions

Join()
Computesalary()
Printdata()

Example

A class bankaccount with following attributes accno, custname, custadd, openamount, closingamount and functions open, deposit, withdrawal, checkbalance, close.

Class bankaccount

Accno—integer
Custname—string
Custadd—string

NOTES

Openamount—integer

Closingamount—integer

Functions

Open()

Deposit()

Withdrawal()

Checkbalance()

2.10 JAVA CLASS

To declare a class, use the class keyword followed by an opening brace, and then list the data members and methods of that class. End the declaration with a closing brace and a semicolon. Here's the declaration of a class called Car:

```
class Car
{
int doors;
int wheels;
Start();
Stop();
};
```

Declaring this class doesn't allochilde memory for a Car. It just tells the compiler what a Car is, what data it contains (wheels and doors), and what it can do Start() and Stop(). It also tells the compiler the size of a Car in bytes, to know how much bytes the compiler must set aside for each Car object that you will create in future. In this example, if an integer is two bytes, a Car is only four bytes big: doors is two bytes, and wheels is another two bytes. Start() and Stop() takes up no room, because no storage space is set aside for member functions.

Program A simple Java class

```
public class Mydog
 {
 String name;
  public void speak()  //Method 1
  {
   System.out.println("Bho Bhon...");
  }
  public void sleep()  //Method 2
  {
  System.out.println("zzzzzzzzzzzzzzzzzz");
```

```
}
}
```

The Mydog class now has a public statement alongside it. The public statement means that the class is available for use by the public, in other words, by any programs you can use this class.

NOTES

The first part of the Mydog class creates a string variable called name. This variable is an attribute of the class and each object will use it, this is what distinguishes one dog from another. The second part of the dog class is a method or behavior called speak(). This method has one statement, the System.out.println statement to display 'Bho Bhon...'

Creation of Objects

To create a new object, you use the new operator with the name of the class you want to create an instance of, then parentheses after that. When you use the new operator, the new instance of the given class is created, and memory is allocated for it. In addition and most importantly, a special method defined in the given class is called to initialize the object and set up any initial values it needs. The following examples create new instances of the classes String, Random, and Motorcycle, and store those new instances in variables of the appropriate types like:

```
String str = new String();
Random r = new Random();
Motorcycle m2 = new Motorcycle();
```

Now as per dog class example If you wanted to use a dog object in your program then you could you the following statement

```
dog myDog = new dog();
```

You can now set the name of myDog

```
myDog.name = "Babul".
```

To make Barney speak by calling the speak method we could use the following:

```
myDog.speak();
```

Program Complete Dog class program

```
public class Mydog
 {
 String name;
 public void speak() //Method 1
  {
   System.out.println("Bho Bhon...");
  }
public void sleep() //Method 2
```

NOTES

```
 {
  System.out.println("zzzzzzzzzzzzzzzzzzz");
 }
}
class lesson5a
 {
 public static void main(String s[])
  {
  Mydog dog1 = new Mydog(); // Object declaration
  dog1.name = "Babul";
  System.out.println("Babul is Barking");
  dog1.speak();              //Calling of Method
  System.out.println("Babul is sleeping");
  Dog1.sleep();
  }
 }
```

Example

A class-object based program to add and subtract two integer numbers using data members and member functions outside the class.

Program

```
public class addsub
{
int x=12;
int y=8;              //private data members
public void twosum()   //public member function
{
int z=x+y;
  System.out.println("sum is ="+ z);
}
public void twosub();
{
int z=x-y ;
System.out.println("Difference is ="+ z);
}
}          // class defined
public static void main(String s []);
{
addsub s=new addsub();        //object declaration of addsub class
```

```
s.twosum();          // calling member function with object
s.twosub();
}
```

NOTES

2.11 JAVA METHODS

Java Methods in a class are used to define an object's behavior-what happens when that object is created and the various operations that object can perform during its lifetime, commonly known as functions in C++.

Defining Methods

A Java Method has four basic parts: -

The name of the method

The type of object or primitive type the method returns

A *list of parameters*

The body of the method

```
returntype methodname(type1 arg1, type2 arg2, ..)
{
  ...
}
```

The returntype is the type of value this method returns. It can be one of the primitive types, a class name, or void if the method does not return a value at all.

The method's parameter list is a set of variable declarations, separated by commas, inside parentheses. These parameters become local variables in the body of the method, whose values are the objects or values of primitives passed in when the method is called. Inside the body of the method you can have statements, expressions, method calls to other objects, conditionals, loops, and so on-everything you've learned about in the previous lessons.

Program

```
class Elevator
{
  boolean running = true;
  void shutDown()  //method of Elevator class
 {
   running = false;
  }
}
class FrontDesk
{
```

```
    private final int EVENING = 8;
    Elevator NorthElevator, SouthElevator;
    FrontDesk() {              // the class constructor
      NorthElevator = new Elevator();
      SouthElevator = new Elevator();
    }
    void maintenance(int time)
{
    if (time == EVENING)
       NorthElevator.shutDown();
    }
    void displayStatus()
    {
     // One more method
     System.out.print("North Elevator is ");
     if (!(NorthElevator.running ))
       System.out.print("not ");
     System.out.println("running.");
     System.out.print("South Elevator is ");
     if (!(SouthElevator.running ))
       System.out.print(" not ");
     System.out.println("running.");
     }
public class Hotel
{
   public static void main(String args[])
{
    FrontDesk lobby;
    lobby = new FrontDesk();
    System.out.println("It's 7:00. Time to check the elevators.");
    lobby.maintenance(7);
    lobby.displayStatus();
    System.out.println();
    System.out.println("It's 8:00. Time to check the elevators.");
    lobby.maintenance(8);
    lobby.displayStatus();
  }
}
```

Use of this Keyword

NOTES

In the body of a method definition, you may want to refer to the current object—the object in which the method is contained in the first place- to refer to that object's instance variables or to pass the current object as an argument to another method. To refer to the current object in these cases, you can use the this keyword. this can be used anywhere the current object might appear-in dot notation to refer to the object's instance variables, as an argument to a method, as the return value for the current method, and so on. Here's an example:

```
t = this.x;              // the x instance variable for this object
this.myMethod(this); // call the myMethod method, defined in
                         // this class, and pass it the current
                         // object
return this;             // return the current object
```

Variable Scope and Method Definitions

When you declare a variable, that variable always has a limited scope. Variable scope determines where that variable can be used. Variables with a local scope, for example, can only be used inside the block in which they were defined. Instance variables have a scope that extends to the entire class so they can be used by any of the methods within that class Like:

Program Use of scope of variables

```
class ScopeTest
{
int test = 10;
void printTest ()
{
 int test = 20;
  System.out.println("test = " + test);
}
public static void main (String args[])
{
ScopeTest st = new ScopeTest();
st.printTest();
}
}
```

In this class, you have two variables with the same name and definition, The first, an instance variable, has the name test and is initialized to the value 10. The second is a local variable with the same name, but with the value 20. Because the local variable hides the instance variable, the println() method will print that test is 20. The easiest way to get

around this problem is to make sure you don't use the same names for local variables as you do for instance variables. Another way to get around this particular problem, however, is to use **this.test** to refer to the instance variable, and just test to refer to the local variable. By referring explicitly to the instance variable by its object scope you avoid the conflict.

Passing Arguments by Value and Reference

When you call a method with object parameters, the variables you pass into the body of the method are passed by reference, which means that whatever you do to those objects inside the method affects the original objects as well. This includes arrays and all the objects that arrays contain; when you pass an array into a method and modify its contents, the original array is affected. Mostly the primitive types are passed by value.

Program Program to show use of Passing by Reference

```
class PassByReference
{
     int onetoZero(int arg[])
 {
       int count = 0;
     for (int i = 0; i < arg.length; i++)
        {
        if (arg[i] == 1)
          {
           count++;
arg[i] = 0;
}
      }
      return count;
    }
    public static void main (String arg[])
   {
      int arr[] = { 1, 3, 4, 5, 1, 1, 7 };
       PassByReference test = new PassByReference();
      int numOnes;
      System.out.print("Values of the array: [ ");
      for (int i = 0; i < arr.length; i++)
      {
       System.out.print(arr[i] + " ");
      }
```

```
        System.out.println("]");
        numOnes = test.onetoZero(arr);
        System.out.println("Number of Ones = " + numOnes);
        System.out.print("New values of the array: [ ");
        for (int i = 0; i < arr.length; i++)
    {
        System.out.print(arr[i] + " ");
    }
     System.out.println("]");
   }
}
```

Java Defined Class Methods

Just as you have class and instance variables, you also have class and instance methods, and the differences between the two types of methods are analogous. Class methods are available to any instance of the class itself and can be made available to other classes. Therefore, some class methods can be used anywhere, regardless of whether an instance of the class exists.

For example, the Java class libraries include a class called Math. The Math class defines a whole set of math operations that can be used in any program or the various number types like:

```
float root = Math.sqrt(453.0);
System.out.print("The larger of x and y is " + Math.max(x, y));.
```

Command-line Arguments

Because Java applications are standalone programs, it's useful to be able to pass arguments or options to a program to determine how the program is going to run, or to enable a generic program to operate on many different kinds of input. Command-line arguments can be used for many different purposes-for example, to turn on debugging input, to indicate a filename to read or write from, or for any other information that you might want your Java program to know.

Passing Arguments to Java Programs

How you pass arguments to a Java application varies based on the platform you're running Java on. On Windows and UNIX, you can pass arguments to the Java program via the command line like:

Program Passing Arguments to Java Programs

```
class PassArgs
{
    public static void main(String args[])
```

NOTES

```
{
    for (int i = 0; i < args.length; i++)
  {
      System.out.println("Argument " + i + ": " + args[i]);
   }
  }
}
```

Run the program as

java PassArgs 10 20 30 mks

An important thing to note about the arguments you pass into a Java program is that those arguments wsill be stored in an array of strings. This means that any arguments you pass to your Java program are strings stored in the argument array. To treat them as non-strings, you'll have to convert them to whatever type you want them to be. For example, suppose you have a very simple Java program called SumAverage that takes any number of numeric arguments and returns the sum and the average of those arguments like:

```
class SumofArgs
{
   public static void main (String args[])
  {
     int sum = 0;
     for (int i = 0; i < args.length; i++)
  {
     sum + = Integer.parseInt(args[i]);
    }
    System.out.println("Sum is: " + sum);
    System.out.println("Average is: " +
    (float)sum / args.length);
  }
   }
```

2.12 INHERITANCE

When you create a class and uses objects to work with class, with a set of attributes and functions, you have created something that is ready to pass these qualities on to it's children or subclass for reuse the main class to save your time and efforts on coding. This is called inheritance, every super class (parent) gives its qualities to its subclass (child). Inheritance

in programs made possible to reuse the attributes and functions of a parent class into a child class.

NOTES

The Family Inheritance

With all family trees we inherit the characteristics of our parents, grand parents and great grand parents. We can inherit that beautiful nose from our mothers side of the family, the buck teeth from our father , the long black hair from our great grandfather etc.

Inheriting Functions and Attributes

The functions and attributes of a class are the combination of two things, its own functions and attributes and the functions r and attributes of all its super classes.

A class which adds new functionality to an existing class is said to derive or inherited from that original class. The original class is said to be the new class's base class.

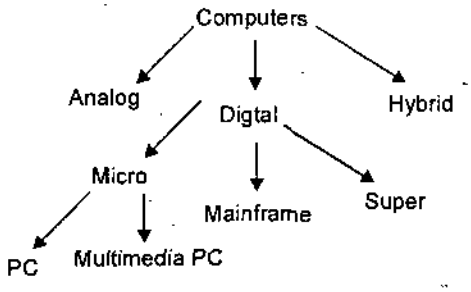

Fig 2.1. Example of Inheritance

Now From figure you can say a PC inherit the features of MICRO computer, while a MICRO computer inherits the features of a DIGITAL computer and the parent class for all is COMPUTERS.

Types of Inheritance

You can design type of Inheritance in Java following:

1. Single level (Parent—Child)
2. Multilevel (Grandparent—Parent—Child)
3. Multiple (Many parents—one child, not direct, using interface).

The declarations of these given classes in Java program can be like this:

```
class Vehicle
  {
   int registrationNumber;
  String  Ownername;  // (assuming that a Person class has been
defined)
```

NOTES

```
    void transferOwnership(String newOwner)
    {
    }
    ...
}
class Car extends Vehicle
{
 int numberOfDoors;
 ...
 }
 class Truck extends Vehicle
{
 int numberOfAxels;
 ...
}
 class Motorcycle extends Vehicle
 {
 boolean hasSidecar;
 ...
 }
```

Suppose that yourCar is a variable of type Car that has been declared and initialized with the statement.

Car yourCar = new Car();

Note that, as with any variable, it is OK to declare a variable and initialize it in a single statement. This is equivalent to the declaration "Car myCar;" followed by the assignment statement "myCar = new Car();".) Given this declaration, a program could refer to myCar.numberOfDoors, since numberOfDoors is an instance variable in the class Car. But since class Car extends class Vehicle, a car also has all the structure and behavior of a vehicle. This means that myCar.registrationNumber, myCar.owner, and myCar.transferOwnership() can be used by object of car class.

Now, in the real world, cars, trucks, and motorcycles are in fact vehicles. The same is true in a program. That is, an object of type Car or Truck or Motorcycle is automatically an object of Vehicle.

Extending Existing Classes

If you have created base classes you can extend features in your sub classes, You can use the given Java syntax in example. In day-to-day *programming*, especially for programmers who are just beginning to work with objects, sub classing is used mainly in one situation. There is an

existing class that can be adapted with a few changes or additions. This is much more common than designing groups of classes and subclasses from scratch. The existing class can be extended to make a subclass. The syntax for this is:

NOTES

```
class subclass-name extends existing-class-name
    {
      .
      .   // Changes and additions.
      .
    }
public class salary extends employee
  {
    public int getsalary()
  {
        // Returns the value of the salary
      int eno;
      netsal= 0;
      netsal= getnetsal();
       }
} // end class
```

Program Use of single level inheritance

```
class  Acalc
{
int num1,num2;
public int addNumbers()
  {
int sum=0;
sum=num1+num2;
return sum;
  }
 }
class sub extends Acalc
{
public int subNumbers()
  {
int sub=0;
sub=num1+num2;
```

NOTES

```
return sub;
  }
}
class expInherit
{
public static void main(String s[])
{
sub objsub=new sub();
objsub.num1=35;
objsub.num2=15;
int a=objsub.addNumbers();
int b=objsub.subNumbers();
System.out.println(a);
System.out.println(b);
}
}
```

Abstract Methods and Classes

Java supports the declaration of abstract methods. Additionally, Java takes this a step further and introduces the concept of an abstract class. Instances of Java abstract classes cannot be created with new. Abstract members and classes are identified by the use of the abstract keyword, as shown in the following:

```
abstract class Shapes{
...
 abstract void createshape();
...
}
```

A class is considered abstract if it has one or more methods that are abstract. In the case of the Species class, the method createshape() is specified as abstract because some shapes have uniqueness.

Multiple Inheritance Using Interfaces

If you design a class that is entirely abstract, then that class is what Java refers to as an interface. A Java interface is similar to a class in that it defines a new type that contains both methods and variables. However, because an interface is completely abstract, its methods are not implemented within the interface. Instead, classes that are derived from an interface implement the methods of the interface.

An interface is declared in the same manner as a class except that instead of class, the keyword interface is used. Java, like C++, does not support multiple class inheritance. In other words, a class may have only one immediate superclass because only a single class name can follow extends in a Java class declaration. Fortunately, class inheritance and interface inheritance can be combined when deriving a new Java class, and a subclass can implement more than one interface. For example, you can do the following:

Program

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
public class SimpleButtonEvent extends Applet  implements ActionListener
{
  private Button b;
  public void init()
 {
    b = new Button("Press me");
     b.addActionListener(this);
    add(b);
 }
  public void actionPerformed(ActionEvent e)
 {
    // If the target of the event was our Button
    // In this example, the check is not
    //   truly necessary as we only listen to
    //   a single button
    if ( e.getSource() == b )
   {
       getGraphics().drawString("hello dear ",20,20);
    }
  }
}
```

Inheritance and Finalize( )

When you use composition to create a new class, you never worry about finalizing the member objects of that class. Each member is an independent object and thus is garbage collected and finalized regardless of whether it happens to be a member of your class. With inheritance, however, you must override **finalize( )** in the derived class if you have

NOTES

any special cleanup that must happen as part of garbage collection.

When you override **finalize( )** in an inherited class, it's important to remember to call the base-class version of **finalize( )**, since otherwise the base-class finalization will not happen.

2.13 JAVA PACKAGES

Packages, are a way of organizing groups of classes in Java. A package contains any number of classes that are related in purpose, in scope, or by inheritance. If your programs are small and use a limited number of classes, you may find that you don't need to explore packages at all. But the more Java programming you do, the more classes you'll find you have. And although those classes may be individually well designed, reusable, encapsulated, and with specific interfaces to other classes, you may find the need for a bigger organizational entity that allows you to group your packages. Although a package is most typically a collection of classes, packages can also contain other packages, forming yet another level of organization somewhat analogous to the inheritance hierarchy. Each "level" usually represents a smaller, more specific grouping of classes. The Java class library itself is organized along these lines. The top level is called java; the next level includes names such as io, net, util, and awt. The last of these has an even lower level, which includes the other packages like image.

Packages are useful for several broad reasons:

They allow you to organize your classes into units. Just as you have folders or directories on your hard disk to organize your files and applications, packages allow you to organize your classes into groups so that you only use what you need for each program.

They reduce problems with conflicts in names. As the number of Java classes grows, so does the likelihood that you'll use the same class name as someone else, opening up the possibility of naming clashes and errors if you try to integrate groups of classes into a single program. Packages allow you to "hide" classes so that conflicts can be avoided.

They allow you to protect classes, variables, and methods in larger ways than on a class-by-class basis and can be used to identify your classes. For example, if you implemented a set of classes to perform some purpose, you could name a package of those classes with a unique identifier that *identifies you or your organization.*

Some Useful Java Packeges

1. Java.lang
2. Java.util
3. Java.net
4. Java.io
5. Java.awt

How to Use Packages?

You've been using packages in many programs. Every time you use the import command, and every time you refer to a class by its full package name like java.awt.Color.

To use a class contained in a package, you can use one of three mechanisms:- If the class you want to use is in the package java.lang (for example, System or Date), you can simply use the class name to refer to that class. The java.lang classes are automatically available to you in all your programs. If the class you want to use is in some other package, you can refer to that class by its full name, including any package names for example, java.awt. Font.

For classes that you use frequently from other packages, you can import individual classes or a whole package of classes. After a class or a package has been imported, you can refer to that class by its class name.

What about your own classes in your own programs that don't belong to any package? The rule is that if you don't specifically define your classes to belong to a package, they're put into an unnamed default package. You can refer to those classes simply by class name from anywhere in your code.

import

To import classes from a package, use the import command, as you've used throughout the examples in this book. You can either import an individual class, like this:

import java.util.Vector.

You can import an entire package of classes, using an asterisk (*) to replace the individual class names like:

import java.awt.*

Writing import java.awt.* imports all the public classes in that package, but does not import sub packages such as image and peer. To import all the classes in a complex package hierarchy, you must explicitly import each level of the hierarchy by hand. Also, you cannot indicate partial class names for example, M* to import all the classes that begin with M.

2.14 INTERFACES

Interfaces, like the abstract classes and methods, provide templates of behavior that other classes are expected to implement. Interfaces, however, provide far more functionality to Java and to class and object design than do simple abstract classes and methods. An interface is a prototype for a class and is useful from a logical design perspective. Unlike an abstract class is a class that has been left partially unimplemented

because it uses abstract methods, which are themselves unimplemented. Interfaces are abstract classes that are left completely unimplemented. Completely unimplemented in this case means that no methods in the class have been implemented. Additionally, interface member data is limited to static final variables, which means that they are constant.

The benefits of using interfaces are much the same as the benefits of using abstract classes. Interfaces provide a means to define the protocols for a class without worrying about the implementation details. This seemingly simple benefit can make large projects much easier to manage; once interfaces have been designed, the class development can take place without worrying about communication among classes.

Another important use of interfaces is the capacity for a class to implement multiple interfaces. This is a violation of the concept of multiple inheritance, which is supported in C++ but not in Java. Multiple inheritance enables you to derive a class from multiple parent classes. Although powerful, multiple inheritance is a complex and often tricky feature of C++ that the Java designers decided they could do without. Their workaround was to allow Java classes to implement multiple interfaces. The major difference between inheriting multiple interfaces and true multiple inheritance is that the interface approach enables you to inherit only method descriptions, not implementations. If a class implements multiple interfaces, that class must provide all the functionality for the methods defined in the interfaces. Although this approach is certainly more limiting than multiple inheritance, it is still a very useful feature. It is this feature of interfaces that separates them from abstract classes.

Declaring Interfaces

The syntax for creating interfaces is as follows:

```
interface nameofinterface
{
  InterfaceBody
}
```

nameofinterface is the name of the interface and InterfaceBody refers to the abstract methods and static final variables that make up the interface. Because it is assumed that all the methods in an interface are abstract, it isn't necessary to use the abstract keyword. You cannot declare a method inside an interface to be either private or protected. So, in next example, here's aimca interface with one method explicitly declared public and abstract grow() and one implicitly declared as such growBigger().

```
public interface aimca
 {
      public abstract void grow();   // explicitly public and abstract
      void growBigger();             // effectively public and abstract
 }
```

Note that, as with abstract methods in classes, methods inside interfaces do not have bodies. Remember, an interface is pure design; there is no implementation involved.

NOTES

Implementing Interfaces

Because an interface is a prototype, or template, for a class, you must implement an interface to arrive at a usable class. Implementing an interface is similar to deriving from a class, except that you are required to implement any methods defined in the interface. To implement an interface, you use the implements keyword. The syntax for implementing a class from an interface follows:

```
class classname  implements Interfacename
{
  ClassBody
}
```

Identifier refers to the name of the new class, Interface is the name of the interface you are implementing, and ClassBody is the new class body.

To use an interface, you include the implements keyword as part of your class definition. When you will learn applets, you will use Animation, Images, and Sound for that you must use threads and applet class method at same time, or you can say multiple inheritance with Runnable interface in your applet definition like:

```
public class Textmove extends java.applet. Applet implements Runnable
    {
  // body of program
    }
```

Program

The line-drawing applet using the Runnable interface.

```
import java.awt.*;
import java.lang.*;
import java.applet.Applet;
 public class LineRun extends Applet implements Runnable
{
    Thread t;  // This is the thread!
    // Set the size of the applet...
    public void init() {
        resize(300, 300);
    }
    // Entering the Applet. Start the thread...
```

```
public void start() {
  if (t == null) {
   t = new Thread(this);
   t.start();
  } // end if
}
// Leaving the Applet. Stop the thread...
public void stop() {
  if (t != null) {
   t.stop();
   try {
      t.join();
   }
    catch (InterruptedException e) { }
  } // end if
  t = null;
}
  // Run the thread. Lines everywhere!
public void run()
{
   // Get dimension data about the applet...
    double width = (double)size().width;
    double height = (double)size().height;
    // Loop and draw lines forever...
    while (true)
    {
       Graphics g = getGraphics();
       // Randomly select a color...
       Color c = new Color((int)(255.0 * Math.random()),
         (int)(255.0 * Math.random()),
         (int)(255.0 * Math.random()) );
          g.setColor(c);
          g.drawLine((int)(width * Math.random()),
          (int)(height * Math.random()),
          (int)(width * Math.random()),
          (int)(height * Math.random()) );
         // Sleep some...
```

```
            try
             t.sleep(100);
            }
            catch (InterruptedException e) { }
        }
    }
}
```

NOTES

Implementing Multiple Interfaces

Unlike the singly inherited class hierarchy, you can include as many interfaces as you need in your own classes, and your class will implement the combined behavior of all the included interfaces. To include multiple interfaces in a class, just separate their names with commas like:

```
public class UseofMulti extends java.applet.Applet
implements Runnable, Eatable, Sortable, Observable
{
program body
}
```

2.15 EXCEPTION

As the name implies, an exception is an exceptional condition. An exception is something out of the ordinary. Most often, exceptions are used as a way to report error conditions like file not found or divide by zero . Exceptions can be used as a means of indicating other situations as well. Java has exceptions to deal with managing, creating, and expecting errors and other unusual situations.

Through a combination of special language features, consistency checking at compile time and a set of extensible exception classes, errors and other unusual conditions in Java programs can be much more easily managed. Given these features, you can now add a whole new dimension to the behavior and design of your classes, of your class hierarchy, and of your overall system. Your class and interface definitions describe how your program is supposed to behave given the best circumstances. By integrating exception handling into your program design, you can consistently describe how the program will behave when circumstances are not quite as good, and allow people who use your classes to know what to expect in those cases.

Throw, Try, and Catch Blocks

To respond to an exception, the call to the method that produces it must be placed within a try block. A try block is a block of code beginning

NOTES

with the try keyword followed by a left and right curly brace. Every try block is associated with one or more catch blocks. Here is a try block:

```
try
    {
    // method calls go here
    }
```

If a method is to catch exceptions thrown by the methods it calls, the calls must be placed within a try block. If an exception is thrown, it is handled in a catch block. Different catch blocks handle different types of exceptions. This is a try block and a catch block set up to handle exceptions of type Exception like:

```
try
    {
    // method calls go here
    }
    catch( Exception e )
    {
    // handle exceptions here
    }
```

When any method in the try block throws any type of exception, execution of the try block ceases. Program control passes immediately to the associated catch block. If the catch block can handle the given exception type, it takes over. If it cannot handle the exception, the exception is passed to the method's caller. In an application, this process goes on until a catch block catches the exception or the exception reaches the main() method uncaught and causes the application to terminate.

Program

```
import java.io.*;
class calc
{
        DataInputStream in = new DataInputStream(System.in);
        int num1=0;
        int num2=0;
            int sum1,product,subtract,div,modl=0;
void input()
  {
                    try
                      {
                        System.out.println("Enter First number=");
```

```
                num1=Integer.parseInt(in.readLine());
            System.out.println("Enter Second number=");
            num2=Integer.parseInt(in.readLine());
                }
catch (IOException e ){ }
 }
void add()
{
sum1=num1+num2;
System.out.println("sum="+sum1);
}
}
   public class calculate
   {
public static void main (String s[])
                {
 calc mk=new calc();
        mk.input();
          mk.add();
                }
}
```

NOTES

Multiple Catch Blocks

In some cases, a method may have to catch different types of exceptions. Java supports multiple catch blocks. Each catch block must specify a different type of

exception Like:

```
try
   {
   // method calls go here
   }
 catch( SomeExceptionClass e )
   {
    // handle SomeExceptionClass exceptions here
   }
 catch( SomeOtherExceptionClass e )
   {
    // handle SomeOtherExceptionClass exceptions here
}
```

When an exception is thrown in the try block, it is caught by the first catch block of the appropriate type. Only one catch block in a given set will be executed. Notice that the catch block looks a lot like a method declaration. The exception caught in a catch block is a local reference to the actual exception object. You can use this exception object to help determine what caused the exception to be thrown in the first place.

The Finally Clause

Java introduces a new concept in exception handling: the finally clause. The finally clause sets apart a block of code that is always executed. Here's an example of a finally clause:

```
import java.io.*;
import java.lang.Exception;
public class MultiThrow {
    public static void main( String[] args)
    {
        try
            {
            alpha() ;
            }
        catch( Exception e )
            {
             System.out.println( "Caught exception " );
            }
     finally()
            {
             System.out.println( "Finally. " );
            }
    }
}
```

In normal execution that is, when no exceptions are thrown, the finally block is executed immediately after the try block. When an exception is thrown, the finally block is executed before control passes to the caller.

Types of Exceptions

The methods of the Java API and the language itself also throw exceptions. These exceptions can be divided into two classes: Exception and Error. Both the Exception and Error classes are derived from Throwable. Exception and its subclasses are used to indicate conditions that may be recoverable. Error and its subclasses indicate conditions that are generally not recoverable and that should cause your applet to terminate. The various packages included in the Java Development Kit throw different kinds of Exception and Error exceptions, as described in the following sections.

The java.io exceptions

Exception	Cause
CharConversionException	Root class for character conversion exceptions
IOException	Root class for I/O exceptions
EOFException	End of file
FileNotFoundException	Unable to locate file
InterruptedIOException	I/O operation was interrupted; contains a member bytes Transferred that indicates how many bytes were transferred before the operation was interrupted
InvalidClassException	Class is not valid for serialization
InvalidObjectException	Class explicitly forbids serialization
NotActiveException	Serialization not active
NotSerializableException	Class may not be serialized
ObjectStreamException	Root class for object stream exceptions
OptionalDataException	Contains data members to indicate end of file or optional data to read
StreamCorruptedException	Stream fails internal consistency test
SyncFailedException	Synchronization failed
UTFDataFormatException	Malformed UTF-8 string
UnsupportedEncodingException	Character-encoding mechanism not supported
WriteAbortExceptionException	in stream

Built-in Exceptions

In the above example see how the automatic exceptions in Java work. This application creates a method and forces it to divide by zero. The method does not have to explicitly throw an exception because the division operator throws an exception when required.

Program

```
import java.io.*;
import java.lang.Exception;
public class DivideBy0
    {
    public static void main( String[] args)
    {
```

```
    int a = 2;
    int b = 3;
    int c = 5;
    int d = 0;
    int e = 1;
    int f = 3;
    try
        {
        System.out.println( a+"/"+b+" = "+div( a, b ) );
        System.out.println( c+"/"+d+" = "+div( c, d ) );
        System.out.println( e+"/"+f+" = "+div( e, f ) );
        }
     catch( Exception except)
        {
         System.out.println( "Caught exception " +
                                except.getMessage() );
        }
    }
    static int div( int a, int b ) {
        return (a/b);
    }
}
```

The output of this application is shown here:

```
2/3 = 0
Caught exception / by zero
```

The first call to div() works fine. The second call fails because of the divide-by-zero error. Even though the application did not specify it, an exception was thrown—and caught. So you can use arithmetic in your code without writing code that explicitly checks bounds.

2.16 STREAMS

A stream is a path of communication between the source of some information and its destination. This information can come from a file, the computer's memory, or even from the Internet. In fact, the source and destination of a stream are completely arbitrary producers and consumers of bytes, respectively-you don't need to know about the source of the information when reading from a stream, and you don't need to know about the final destination when writing to one. A stream is a path of communication between a source of information and its destination. For example, an

input stream allows you to read data from a source, and an output stream allows you to write data to a destination.

2.17 JAVA IO

NOTES

All the classes that are used in Java stream programming either input, output or file are part of the package java.io. To use any of these classes in your own programs, you will need to import each individual class or to import the entire java.io package, like this:

```
import java.io.InputStream;
import java.io.FilteredInputStream;
import java.io.FileOutputStream;
import java.io.*;
```

The foundations of this stream framework in the Java class hierarchy are the two abstract classes, InputStream and OutputStream. Inheriting from these classes is a virtual cornucopia of categorized subclasses, demonstrating the wide range of streams in the system, but also demonstrating an extremely well-designed hierarchy of relationships between these streams-one well worth learning from. All the methods you will use in stream based programs are declared to throw IOExceptions. This new subclass of Exception conceptually embodies all the possible I/O errors that might occur while using streams. Several subclasses of it define a few, more specific exceptions that can be thrown as well. For now, it is enough to know that you must either catch an IOException, or be in a method that can "pass it along," to be a well-behaved user of streams.

2.18 INPUT STREAM AND READER CLASSES

The Java input model is based on the concept of an input stream. More practically speaking, Java uses input streams as the means of reading data from an input source, such as the keyboard. The basic input stream classes supported by Java as follows:

InputStream

BufferedInputStream

DataInputStream

FileInputStream

StringBufferInputStream

Java version 1.1 introduces support for character input streams, which are virtually identical to the input streams just listed except that they operate on characters rather than on bytes. The character input stream classes are called readers instead of input streams. Corresponding

reader classes implement methods similar to the input stream classes just listed except for the DataInputStream class. Following are the basic reader classes provided by Java:

Reader

BufferedReader

FileReader

StringReader

The InputStream Class

The InputStream class is an abstract class that serves as the base class for all other input stream classes. InputStream defines a basic interface for reading streamed bytes of information. The methods defined by the InputStream class will become very familiar to you because they serve a similar purpose in every InputStream-derived class. This design approach enables you to learn the protocol for managing input streams once and then apply it to different devices using an InputStream-derived class.

The typical scenario when using an input stream is to create an InputStream-derived object and then tell it you want to input information by calling an appropriate method. If no input information is currently available, the InputStream uses a technique known as blocking to wait until input data becomes available. An example of when blocking takes place is the case of using an input stream to read information from the keyboard. Until the user types information and presses Return or Enter, there is no input available to the InputStream object. The InputStream object then waits (blocks) until the user presses Return or Enter, at which time the input data becomes available and the InputStream object can process it as input.The InputStream class defines the following methods:

```
abstract int read()
int read(byte b[])
int read(byte b[], int off, int len)
long skip(long n)
int available()
synchronized void mark(int readlimit)
synchronized void reset()
boolean markSupported()
void close()
```

InputStream defines three different read() methods for reading input data in various ways. The first read() method takes no parameters and simply reads a byte of data from the input stream and returns it as an integer. This version of read() returns -1 if the end of the input stream is reached. Because this version of read() returns a byte of input as an int, you must cast it to a char if you are reading characters. The second

NOTES

version of read() takes an array of bytes as its only parameter, enabling you to read multiple bytes of data at once. The data that is read is stored in this array. You have to make sure that the byte array passed into read() is large enough to hold the information being read or an IOException is thrown. This version of read() returns the actual number of bytes read, or -1 if the end of the stream is reached. The last version of read() takes a byte array, an integer offset, and an integer length as parameters. This version of read() is very similar to the second version except that it enables you to specify where in the byte array you want to place the information that is read. The off parameter specifies the offset into the byte array to start placing read data, and the len parameter specifies the maximum number of bytes to read.

The skip() method is used to skip over bytes of data in the input stream. skip() takes a long value n as its only parameter, which specifies how many bytes of input to skip. It returns the actual number of bytes skipped or -1 if the end of the input stream is reached.

The available() method is used to determine the number of bytes of input data that can be read without blocking. available() takes no parameters and returns the number of available bytes. This method is useful if you want to ensure that there is input data available (and therefore avoid the blocking mechanism).

The mark() method marks the current position in the stream. You can later return to this position using the reset() method. The mark() and reset() methods are useful in situations in which you want to read ahead in the stream but not lose your original position. An example of this situation is verifying a file type, such as an image file. You would probably read the file header first and mark the position at the end of the header. You would then read some of the data to make sure that it follows the format expected for that file type. If the data doesn't look right, you can reset the read pointer and try a different technique.

Notice that the mark() method takes an integer parameter, readlimit. readlimit specifies how many bytes can be read before the mark becomes invalidated. In effect, readlimit determines how far you can read ahead and still be able to reset the marked position. The markSupported() method returns a boolean value representing whether or not an input stream supports the mark/reset functionality.

Finally, the close() method closes an input stream and releases any resources associated with the stream. It is not necessary to explicitly call close() because input streams are automatically closed when the InputStream object is destroyed. Although it is not necessary, calling close() immediately after you are finished using a stream is a good programming practice. The reason is that close() causes the stream buffer to be flushed, which helps avoid file corruption.

NOTES

The System.in Object

The keyboard is the standard device for retrieving user input. The System class contained in the language package contains a member variable that represents the keyboard, or standard input stream. This member variable is called in and is an instance of the InputStream class.

Program

```
import java.io.*;
class ReadKeys1
{
  public static void main (String args[])
 {
    StringBuffer s = new StringBuffer();
    char c;
    try
      {
       Reader in = new InputStreamReader(System.in);
      while ((c = (char)in.read()) != '\n')
     {
        s.append(c);
      }
    }
     catch (Exception e) {
      System.out.println("Error: " + e.toString());
    }
    System.out.println(s);
  }
}
```

The ReadKeys1 class first creates a StringBuffer object called s. It then creates a reader object connected to the standard input stream and enters a while loop that repeatedly calls the read() method until a newline character (\n) is detected (that is, the user presses Return). Notice that the input data returned by read() is cast to a char type before being stored in the character variable c. Each time a character is read, it is appended to the string buffer using the append() method of StringBuffer.

Program

```
import java.io.*;
class ReadKeys2
{
  public static void main (String args[])
```

```
    {
    char buf[] = new char[80];
    try {
       Reader in = new InputStreamReader(System.in);
      in.read(buf, 0, 80);
    }
    catch (Exception e)
    {
       System.out.println("Error: " + e.toString());
    }
    String s = new String(buf);
    System.out.println(s);
  }
}
```

In ReadKeys2, an array of characters 80 characters long is created. A reader object is created and connected to the standard input stream, and a single read() method call is performed that reads everything the user has typed. The input is blocked until the user presses Return, at which time the input becomes available and the read() method fills the character array with the new data. A String object is then created to hold the constant string previously read. Notice that the constructor used to create the String object takes an array of characters (buf) as the first parameter. Finally, println() is again used to output the string.

Program

```
import java.io.*;
class ReadKeys3
{
  public static void main (String args[])
{
    char buf[] = new char[10];
    try {
       Reader in = new InputStreamReader(System.in);
      in.read(buf, 0, 10);
    }
    catch (Exception e)
    {
       System.out.println("Error: " + e.toString());
    }
```

```
            String s = new String(buf);
            System.out.println(s);
        }
    }
```

NOTES

ReadKeys3 is very similar to ReadKeys2, with one major difference: The third version of the read() method is used to limit the maximum number of characters read into the array. The size of the character array is also shortened to 10 characters to show what this version of read() does when more data is available than the array can hold. Remember that this version of read() can also be used to read data into a specific offset of the array. In this case, the offset is specified as 0 so that the only difference is the maximum number of characters that can be read (10).

Buffered Input Stream Class

As its name implies, the BufferedInputStream class provides a buffered stream of input. This means that more data is read into the buffered stream than you might have requested so that subsequent reads come straight out of the buffer rather than from the input device. This arrangement can result in much faster read access because reading from a buffer is really just reading from memory. BufferedInputStream implements all the same methods defined by InputStream. As a matter-of fact, it doesn't implement any new methods of its own. However, the BufferedInputStream class does have two different constructors, which follow:

```
BufferedInputStream(InputStream in)
BufferedInputStream(InputStream in, int size)
```

To support buffered input, the BufferedInputStream class also defines a handful of member variables, which follow:

```
byte buf[]
int count
int pos
int markpos
int marklimit
```

The buf byte array member is the buffer in which input data is actually stored. The count member variable keeps up with how many bytes are stored in the buffer. The pos member variable keeps up with the current read position in the buffer. The markpos member variable specifies the current mark position in the buffer as set using the mark() method. markpos is equal to -1 if no mark has been set. Finally, the marklimit member variable specifies the maximum number of bytes that can be read before the mark position is no longer valid. marklimit is set by the readlimit parameter passed into the mark() method. Because all these member variables are specified as protected, you will probably never

actually use any of them. However, seeing these variables should give you some insight into how the BufferedInputStream class implements the methods defined by InputStream.The BufferedReader class is very similar to the BufferedInputStream class except that it deals with characters instead of bytes.

Program

```
import java.io.*;
class ReadKeys4
{
  public static void main (String args[])
   {
   Reader in = new BufferedReader(new InputStreamReader(System.in));
    char buf[] = new char[10];
    try {
      in.read(buf, 0, 10);
    }
    catch (Exception e)
    {
      System.out.println("Error: " + e.toString());
    }
    String s = new String(buf);
    System.out.println(s);
  }
}
```

The BufferedReader object is created by passing the System.in input stream into an InputStreamReader object. This approach is necessary because the BufferedReader() constructor requires a Reader-derived object. From there on, the program is essentially the same as ReadKeys3, except that the read() method is called on the BufferedReader object rather than on an InputStreamReader object.

Data Input Stream Class

The DataInputStream class is useful for reading primitive Java data types from keyboard or other an input stream in a portable fashion. There is only one constructor for DataInputStream, which simply takes an InputStream object as its only parameter. This constructor is defined as follows:

DataInputStream(InputStream in)

DataInputStream implements the following useful methods beyond those defined by InputStream:

final int skipBytes(int n)

NOTES

final void readFully(byte b[])
final void readFully(byte b[], int off, int len)
final boolean readBoolean()
final byte readByte()
final int readUnsignedByte()
final short readShort()
final int readUnsignedShort()
final char readChar()
final int readInt()
final long readLong()
final float readFloat()
final double readDouble()

The skipBytes() method works in a manner very similar to skip(); the exception is that skipBytes() blocks until all bytes are skipped. The number of bytes to skip is determined by the integer parameter n. There are two readFully() methods implemented by DataInputStream. These methods are similar to the read() methods except that they block until all data has been read. The normal read() methods block only until some data is available, not all. The readFully() methods are to the read() methods what skipBytes() is to skip(). The rest of the methods implemented by DataInputStream are variations of the read() method for different fundamental data types. The type read by each method is easily identifiable by the name of the method.

Program

```
import java.io.*;
public class kinput1
{
public static void main (String s[])
                    {
DataInputStream in = new DataInputStream(System.in);
            int x=0;
        int rem=0;
                try
                  {
x=Integer.parseInt(in.readLine());
System.out.println(x);
          rem=x%2;
                    if (rem==0)
                  {
```

```
                    System.out.println("Even Number");
                    }
                    else
                    {
                    System.out.println("Odd Number");
                    }
                    }
catch (IOException e ){ }
                    }
}
```

Program

```
import java.io.*;
class kin2
{
   DataInputStream in = new DataInputStream(System.in);
            int x=0;
         int rem=0;
   void input()
   {
try
                    {
x=Integer.parseInt(in.readLine());
System.out.println(x);
           rem=x%2;
         if (rem==0)
                    {
                    System.out.println("Even Number");
                    }
                    else
                    {
                      System.out.println("Odd Number");
                      }
                         }
catch (IOException e ){ }
                    }
void output()
{
```

NOTES

```
System.out.println("numberis ="+x + "reminder is"+rem);
}
}
    public class kinput2
   {
public static void main (String s[])
                    {
 kin mk=new kin();
        mk.input();
       mk.output();
                    }
}
```

Program

```
import java.io.*;

class calc
{
   DataInputStream in = new DataInputStream(System.in);
        int num1=0;
        int num2=0;
    int sum1,product,subtract,div,modl=0;
     void input()
  {
                  try
                    {
System.out.println("Enter First number=");
num1=Integer.parseInt(in.readLine());
System.out.println("Enter Second number=");
num2=Integer.parseInt(in.readLine());
                    }
catch (IOException e ){ }
}
void add()
{
sum1=num1+num2;
System.out.println("sum="+sum1);
```

```
}
}
    public class calculate
   {
public static void main (String s[])
                    {
 calc mk=new calc();
         mk.input();
           mk.add();
                    }
}
```

2.19 THREAD

In the early days of computing, computers were single tasking—that is, they ran a single job at a time. The big, lumbering machine would start one job, run that job to completion, then start the next job, and so on. When engineers became overly frustrated with these batch-oriented systems, they rewrote the programs that ran the machines and thus was born the modern multitasking operating system.

Multitasking refers to a computer's capability to perform multiple jobs concurrently. For the most part, modern operating systems like Windows XP or Linux can run two or more programs at the same time. While you are using Netscape to download a big file, you can be play a song and also play Solitaire in a different window; both programs are running at the same time.

Multithreading is an extension of the multitasking paradigm. But rather than multiple programs, multithreading involves multiple threads of control within a single program. Not only is the operating system running multiple programs, each program can run multiple threads of control—think of threads as subprograms—within the program. For example, using a Web browser you can open multiple web sites in a single browser, you can print one Web page, download another, and fill out a form in a third all at the same time.

Threads in Java

If you want to write your first Java thread program, just follow the given steps:

1. Create a new class and extend it from Thread class
2. Each thread class must have a run() method, so write a run() method
3. Now your main() will start your thread so create object of your thread class in main()

NOTES

4. Now call start() method with the help of Object.

Program

```
class MyFirstThread extends Thread
{
    int from, to;
    public MyFirstThread (int from, int to)
    {
        this.from = from;
        this.to = to;
    }
                // the run() method is must so write it
    public void run()
    {
        for (int i=from; i<to; i++)
        {
            System.out.println("i == " + i);
        }
    }
    public static void main(String[] args)
    {
        // create  5 threads, each of wich counts 200 numbers
        for (int i=0; i<5; i++)
              {
          MyFirstThread t = new MyFirstThread (i*200, (i+1)*200);
            // starting a thread will launch a separate sequence
            // of control and execute the run() method of the thread
           t.start();
        }
    }
}
```

Program

```
public class PingPONG extends Thread
 {
    private String word; // What word to print
```

```
    private int delay; // how long to pause
    public PingPONG(String whatToSay, int delayTime)
      {
        word = whatToSay;
        delay = delayTime;
    }
    public void run()
       {
       try
         {
           for (;;)
            {
                System.out.print(word + " ");
                sleep(delay); // wait until next time
            }
        } catch (InterruptedException e)
        {
            return; // end this thread;
      }
    }
    public static void main(String[] args)
    {
       new PingPONG("Ping", 33).start();                    // 1/30 second
       new PingPONG("PONG",100).start();                    // 1/10 second
    }
}
```

Implementing thread using the Runnable Interface

There are situations in which it is not convenient to create a Thread subclass. For example, you may want to add a run() method to a preexisting class that does not inherit from Thread. The Java Runnable interface makes this possible.

Program

```
    import java.awt.*;
    import java.lang.*;
    import java.applet.Applet;
    public class LineRun extends Applet implements Runnable
     {
      Thread t;  // This is the thread!
```

NOTES

```
// Set the size of the applet
public void init() {
      resize(300,300);
}
    // Entering the Applet. Start the thread
public void start()
{
  if (t == null)
  {
   t = new Thread(this);
   t.start();
  } f
}
    // Leaving the Applet. Stop the threa
public void stop()
    {
  if (t != null)
   {
   t.stop();
   try {
      t.join();
   }
    catch (InterruptedException e) { }
  }
  t = null;
}
    // Run the thread.
public void run()
 {
   // Get dimension data about the applet
   double width = (double)size().width;
   double height = (double)size().height;
   // Loop and draw lines forever
   while (true)
    {
      Graphics g = getGraphics();
      // Randomly select a color
```

```
        Color c = new Color((int)(255.0 * Math.random()),
          (int)(255.0 * Math.random()),
          (int)(255.0 * Math.random()) );
        g.setColor(c);
        g.drawLine((int)(width * Math.random()),
          (int)(height * Math.random()),
          (int)(width * Math.random()),
          (int)(height * Math.random()) );
        // Sleep some
        try
        {
          t.sleep(100);
        }
        catch (InterruptedException e) { }
      }
    }
  }
```

NOTES

2.20 THREAD STATES

NEW BORN state:

When a Thread object is first created, it is in the NEW state. At this point, the thread is not executing.

RUNABLE state:

When you invoke the Thread's start() method, the thread changes to the RUNABLE state.When a Java thread is RUNABLE, it is eligible for execution. However, a thread that is RUNABLE is not necessarily running. RUNABLE implies that the thread is alive and that it can be allocated CPU time by the system when the CPU is available—but the CPU may not always be available. On single-processor systems, Java threads must share the single CPU; additionally, the Java virtual machine task (or process) must also share the CPU with other tasks running on the system.

NOT RUNABLE state:

When certain events happen to a RUNABLE thread, the thread may enter the NOT RUNABLE state. When a thread is NOT RUNABLE, it is still alive, but it is not eligible for execution. The thread is not allocated time on the CPU. Some of the events that may cause a thread to become NOT RUNABLE include the following:

NOTES

The thread is waiting for an I/O operation to complete. The thread has been put to sleep for a certain period of time (using the sleep() method). The wait() method has been called or the thread has been suspended using the suspend() method A NOT RUNABLE thread becomes RUNABLE again when the condition that caused the thread to become NOT RUNABLE ends (I/O has completed, the thread has ended its sleep() period, and so on). During the lifetime of a thread, the thread may frequently move between the RUNABLE and NOT RUNABLE states.

DEAD state : When a thread terminates, it is said to be DEAD. Threads can become DEAD in a variety of ways. Usually, a thread dies when its run() method returns. A thread may also die when its stop() or destroy() method is called. A thread that is DEAD is permanently DEAD—there is no way to resurrect a DEAD thread.

Naming of Threads

public final String getName();

public final void setName(String name);

Every Java thread has a name. The name can be set during construction or with the setName() method. If you fail to specify a name during construction, the system generates a unique name of the form. Thread-N, where N is a unique integer; the name can be changed later using setName().

The name of a thread can be retrieved using the getName() method. Thread names are important because they provide the programmer with a useful way to identify particular threads during debugging. You should name a thread in such a way that you (or others) will find the name helpful in identifying the purpose or function of the thread during debugging.

Starting and Stopping Threads

To start and stop threads once you have created them, you need the following methods:

public void start();

public final void stop();

public final void stop(Throwable obj);

public void destroy();

To begin a new thread, create a new thread object and call its start() method. An exception is thrown if start() is called more than once on the same thread. As discussed in "Thread States," earlier in this chapter, there are two main ways a thread can terminate: The thread can return from its run() method, ending gracefully. Or the thread can be terminated by the stop() or destroy() method.

When invoked on a thread, the stop() method causes that thread to terminate by throwing an exception to the thread (a ThreadDeath exception). Calling stop() on a thread has the same behaviour as executing throw new

ThreadDeath() within the thread, except that stop() can also be called from other threads (whereas the throw statement affects only the current thread).

NOTES

To understand why stop() is implemented this way, consider what it means to stop a running thread. Active threads are part of a running program, and each runnable thread is in the middle of doing something. It is likely that each thread is consuming system resources: file descriptors, graphics contexts, monitors (to be discussed later), and so on. If stopping a thread caused all activity on the thread to cease immediately, these resources might not be cleaned up properly. The thread would not have a chance to close its open files or release the monitors it has locked. If a thread were stopped at the wrong moment, it would be unable to free these resources; this leads to potential problems for the virtual machine running out of open file descriptors, for example.

Program

```
class DeadThread extends Thread
  {
  // main(), this class is an application
  public static void main(String[] args)
   {
     Thread t = new DeadThread();          // create the thread
     t.start();                            // start the thread
     // wait for a while
      try { Thread.sleep(100); } catch (InterruptedException e) { }
     t.stop();                             // now stop the thread
  }
  // run(), this class is also a Thread
  public void run()
   {
     int n = 0;
     PrintStream ps = null;
     try {
          ps = new PrintStream(new FileOutputStream("big.txt"));
         while (true)
          {                              // forever
            ps.println("n == " + n++);
             try { Thread.sleep(5); } catch (InterruptedException e) {
}
         }
```

NOTES

```
        } catch (ThreadDeath td)
          {                    // watch for the stop()
            System.out.println("Cleaning up.");
            ps.close();                          // close the open file
            throw td;
        } catch (IOException e) {
        }
    }
}
```

2.21 APPLETS VS APPLICATIONS

In short, Java applications are standalone Java programs that can be run by using just the Java interpreter, for example, from a command line. Most everything you've used up to this point in the book has been a Java application, albeit a simple one. Java applets, however, are run from inside a World Wide Web browser. A reference to an applet is embedded in a Web page using a special HTML tag. When a reader, using a Java-enabled browser, loads a Web page with an applet in it, the browser downloads that applet from a Web server and executes it on the local system the one the browser is running on. The Java interpreter is built into the browser and runs the compiled Java class file from there. Because Java applets run inside a Java browser, they have access to the structure the browser provides: an existing window, an event-handling and graphics context, and the surrounding user interface. One final significant difference between Java applets and applications—probably the biggest difference-is the set of restrictions placed on how applets can operate in the name of security. Given the fact that Java applets can be downloaded from any site on the World Wide Web and run on a client's system, Java-enabled browsers and tools limit what can be done to prevent a rogue applet from causing system damage or security breaches. Without these restrictions in place, Java applets could be written to contain viruses or trojan horses (programs that seem friendly but do some sort of damage to the system), or be used to compromise the security of the system that runs them. The restrictions on applets include the following:

Applets can't read or write to the reader's file system, which means they cannot delete files or test to see what programs you have installed on the hard drive. Applets can't communicate with any network server other than the one that had originally stored the applet, to prevent the applet from attacking another system from the reader's system. Applets can't run any programs on the reader's system. For UNIX systems, this includes forking a process. Applets can't load programs native to the local platform, including shared libraries such as DLLs.

NOTES

All these rules are true for Java applets running Netscape Navigator or Microsoft Internet Explorer. Other Java-enabled browsers or tools may allow you to configure the level of security you want-for example, the appletviewer tool in the JDK allows you to set an access control list for which directories an applet canread or write.

How to Run an Applet on Web?

Although Java is a general-purpose programming language suitable for a large variety of tasks, the task most people use it for is applet programming. Java is a programming language designed for networked computers and the World Wide Web. Java applets are downloaded over a network to appear on a Web page. Part of learning Java is learning to program applets and other Graphical User Interface programs. GUI programs are event-driven. That is, user actions such as clicking on a button or pressing a key on the keyboard generate events, and the program must respond to these events as they occur. Current uses of applets include these are:

- Tickertape-style news and sports headline updates
- Animated graphics
- Video games
- Student tests
- Imagemaps that respond to mouse movement
- Advanced text displays
- Database reports.

Java applets are typically run within a Web browser. As of July 1996, most of the popular Web browsers on the market support embedded Java applets in HTML pages. These browsers include:

- Sun HotJava 1.0
- Netscape Navigator 2.0 (or greater)
- Microsoft Internet Explorer 5.0 or greater
- Oracle PowerBrowser 1.5.

Of course, the Applet Viewer tool included with the Java Developer's Kit can be used to test and run Java applets as well. However, it will not be included on all client machines; therefore, it is not really an option to applet developers. Many beginning Java developers wonder how an applet can run within a browser if the Java runtime libraries are not installed on the client system. To run a simple Java applet, "Installing Java," it was first necessary to download the JDK from the JavaSoft Web site. After installing the JDK on the system, the reader could then run any sample Java applet included with the JDK.

NOTES

2.22 APPLET LIFE CYCLE

Each applet has four major events in its lifetime:

1. Initialization
2. Starting
3. Stopping
4. Destroying.

These four events correspond directly to four methods within the Applet class, init(), start(), stop(), and destroy(). The following discussion provides some information on each of these methods.

public void init()

The init() method is called when the applet is initially loaded. This method is used to do one-time setup features such as add components to the layout manager, set screen colors, and connect to a host database.

public void start()

The start() method is called after the applet has been initialized, and also each time the applet is restarted after being stopped. Applets can be stopped when the user changes to another Web page. If the user returns at a later time to the page with the applet on it, the applet will be restarted by the browser. Therefore, the start() method can be called many times during an applet's life cycle. Common operations that occur during an applet's start() method are the initialization of threads within the applet and the updating of the screen display.

public void stop()

The stop() method is called whenever the user leaves the current page. Note that by default when the user leaves a page, the applet will continue to run. This can result in an enormous consumption of system resources if many applets have been loaded and these applets are performing some resource-intensive task such as animation. In fact, it is quite common to see poorly written applets loaded from the Internet that obviously did not implement this method. They never stop running!, The stop() method is used to temporarily suspend the execution of the applet until the start() method is called again.

public void destroy()

The destroy() method is called whenever it is time to completely finish the applet's execution. This method is generally called when the browser is exiting or the applet is being reloaded from the host server. The destroy() method is used to free up allocated resources such as threads or database connections.

Example

```
import java.awt.*;
import java.applet.*;

public class HelloWorldApplet extends Applet
{
        // An applet that simply displays the Hello World!
  public void paint(Graphics g)
  {
     g.drawString("Hello World!", 110, 130);
  }
        } // end of class HelloWorldApplet
```

NOTES

Now write HTML file to run this file as given:

```
<HTML>
<HEAD>
<TITLE>This is the example of first applet!</TITLE>
</HEAD>
<BODY>
<H1>Now be ready with hello </H1>
<BR>
<APPLET CODE=" HelloWorldApplet.class" WIDTH=600 HEIGHT=50>
</APPLET>
</BODY>
</HTML>
```

Note: Your Java class file and html file must be in same dir to run your first applet.

Program

```
import java.awt.*;
import java.applet.*;
public class LifeCycleofApplet extends java.applet.Applet
{
  Font theFont = new Font("Helvetica", Font.BOLD, 20);
 int  i;
 String String1, String2;
 public void paint(Graphics g)
 {
```

NOTES

```
        g.setFont(theFont);
        g.setColor(Color.blue);
        g.drawString(String1 + String2, 5, 30);
    }
    public void init()
    {
      i = 0;
      String1 = "";
      String2 = "The applet is initializing!";
      repaint();
      showStatus("The applet is initializing!");
    }
    public void start()
    {
      i = 1;
      String1 = String2;
      String2 = "The applet is starting!";
      repaint();
      showStatus("The applet is starting!");
    }
    public void stop()
    {
      i = 2;
      String1 = String2;
      String2 = "The applet is stopping!";
      repaint();
      showStatus("The applet is stopping!");
    }
    public void destroy()
    {
      i = 3;
      String1 = String2;
      String2 = "The applet is being destroyed!";
      repaint();
      showStatus("The applet is being destroyed!");
    }
}
```

Now write HTML file to run this file as given :-

```
<HTML>
<HEAD>
<TITLE>This is the example of Life Cycle of an applet!</TITLE>
</HEAD>
<BODY>
<H1>Now be ready !</H1>
<BR>
<APPLET CODE="LifeCycleofApplet.class" WIDTH=600 HEIGHT=50>
If you can see this, your browser does not support Java applets.
</APPLET>
</BODY>
</HTML>
```

NOTES

Program

```
import java.awt.*;
import java.applet.*;
 public class HelloWorldApplet2 extends Applet
{
   public void init()
        {
          // Initialize the applet by setting it to use blue
          // and yellow as background and foreground colors.
       setBackground(Color.blue);
       setForeground(Color.yellow);
    }
    public void paint(Graphics g) {
        g.drawString("Hello Amrapali MCA4th students !", 110,
130);
    }
} // end of class HelloWorldApplet2.
```

2.23 PARAMETERS IN APPLET

In Java applications, you pass parameters to your main() routine by using arguments on the command line.You can then parse those arguments inside the body of your class, and the application acts accordingly, based on the arguments it is given. Applets, however, don't have a command line. How do you pass in different arguments to an applet?

NOTES

Applets can get different input from the HTML file that contains the <APPLET> tag through the use of applet parameters. To set up and handle parameters in an applet, you need two things:

1. A special parameter tag in the HTML file
2. Code in your applet to parse those parameters

Applet parameters come in two parts:

1. a parameter name, which is simply a name you pick,
2. a value which is the actual value of that particular parameter.

In the HTML file that contains the embedded applet, you indicate each parameter using the <PARAM> tag, which has two attributes for the name and the value, called (surprisingly enough) NAME and VALUE. The <PARAM> tag goes inside the opening and closing <APPLET> tags like:

```
<APPLET CODE="MyApplet.class" WIDTH=100 HEIGHT=100>
<PARAM NAME=font VALUE="TimesRoman">
<PARAM NAME=size VALUE="36">
</APPLET>
```

Program

```
import java.awt.* ;
 public class ParamHelloApplet extends java.applet.Applet
{
 Font f = new Font("TimesRoman", Font.BOLD, 36);
String name;
 public void init()
 {
      name = getParameter("name");
     if (name = = null)
     name = "Vama";
     name = "Hello " + name + "!";
   }
     public void paint(Graphics g)
     {
      g.setFont(f);
      g.setColor(Color.red);
     g.drawString(name, 5, 40);
  }
}
```

Now let's create the HTML file that contains this applet

```
HTML>
<HEAD>
<TITLE>Hello applet with parameter !</TITLE>
</HEAD>
<BODY>
<P>
  <APPLET CODE="ParamHelloApplet.class" WIDTH=200
HEIGHT=50>
<PARAM NAME=name VALUE="Suhani">
Hello to all!
</APPLET>
</BODY>
</HTML>
```

2.24 COLORS, FONTS, SHAPES IN APPLETS

Colors

Java is designed to work with the RGB color system. An RGB color is specified by three numbers that give the level of red, green, and blue, respectively, in the color. A color in Java is an object of the class, java.awt.Color. You can construct a new color by specifying its red, blue, and green components. For example,

```
myColor = new Color(r,g,b);
```

There are two constructors that you can call in this way. In the one that I almost always use, r, g, and b are integers in the range 0 to 255. In the other, they are numbers of type float in the range 0.0F to 1.0F. You might recall that a literal of type float is written with an "F" to distinguish it from a double number. Often, you can avoid constructing new colors altogether, since the Color class defines several named constants representing common colors: Color.white, Color.black, Color.red, Color.green, Color.blue, Color.cyan, Color.magenta, Color.yellow, Color.pink, Color.orange, Color.lightGray, Color.gray, and Color.darkGray. An alternative to RGB is the HSB color system.

```
myColor = Color.getHSBColor(h,s,b);
```

For example, you could make a random color that is as bright and as saturated as possible with like:

```
myColor = Color.getHSBColor( (float)Math.random(), 1.0F, 1.0F ).
```

NOTES

Fonts

A font represents a particular size and style of text. The same character will appear different in different fonts. In Java, a font is characterized by a font name, a style, and a size. The available font names are system dependent, but you can always use the following four strings as font names: "Serif", "SansSerif", "Monospaced", and "Dialog". In the original Java 1.0, the font names were "TimesRoman", "Helvetica", and "Courier". You can still use the older names if you want.

The style of a font is specified using named constants that are defined in the Font class. You can specify the style as one of the four values:-

- Font.PLAIN,
- Font.ITALIC,
- Font.BOLD, or
- Font.BOLD + Font.ITALIC.

The size of a font is an integer. Size typically ranges from about 10 to 36, although larger sizes can also be used. The size of a font is usually about equal to the height of the largest characters in the font, in pixels, but this is not a definite rule. The size of the default font is 12.Java uses the class named java.awt.Font for representing fonts. You can construct a new font by specifying its font name, style, and size in a constructor like:

```
Font plainFont = new Font("Serif", Font.PLAIN, 12);
Font bigBoldFont = new Font("SansSerif", Font.BOLD, 24);
```

Shapes

The Graphics class includes a large number of instance methods for drawing various shapes, such as lines, rectangles, and ovals. The shapes are specified using the (x, y) coordinate system described above. They are drawn in the current drawing color of the graphics context. The current drawing color is set to the foreground color of the component when the graphics context is created, but it can be changed at any time using the setColor() method.

Here is a list of some of the most important drawing methods. With all these commands, any drawing that is done outside the boundaries of the component is ignored. Note that all these methods are in the Graphics class, so they all must be called through an object of type Graphics.

drawString(String str, int x, int y) — Draws the text given by the string str. The string is drawn using the current color and font of the graphics context. x specifies the position of the left end of the string. y is the y-coordinate of the baseline of the string. The baseline is a horizontal line on which the characters rest. Some parts of the characters, such as the tail on a y or g, extend below the baseline.

```
    screen.drawPolygon(xPoints,yPoints,points);
}
}
```

Program

NOTES

```
import java.awt.*;
import javax.swing.*;
public class Javastring extends JApplet
{
   String message;
   Font font1, font2, font3, font4, font5;
   Display drawingSurface;
   public void init() {
      message = getParameter("message");
      if (message == null)
        message = "Java!";
      font1 = new Font("Serif", Font.BOLD, 14);
      font2 = new Font("SansSerif", Font.BOLD + Font.ITALIC, 24);
      font3 = new Font("Monospaced", Font.PLAIN, 20);
      font4 = new Font("Dialog", Font.PLAIN, 30);
      font5 = new Font("Serif", Font.ITALIC, 36);
      drawingSurface = new Display();
       drawingSurface.setBackground(Color.black);
      setContentPane(drawingSurface);
   }
   class Display extends JPanel
    {
      public void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        int width = getSize().width;
       int height = getSize().height;
       for (int i = 0; i < 25; i++)
       {
       int fontNum = (int)(5*Math.random()) + 1;
           switch (fontNum)
           {
              case 1:
```

NOTES

```
                    g.setFont(font1);
                    break;
                case 2:
                    g.setFont(font2);
                    break;
                case 3:
                    g.setFont(font3);
                    break;
                case 4:
                    g.setFont(font4);
                    break;
                case 5:
                    g.setFont(font5);
                    break;
            }
            float hue = (float)Math.random();
            g.setColor( Color.getHSBColor(hue, 1.0F, 1.0F) );
            int x,y;
            x = -50 + (int)(Math.random()*(width+40));
            y = (int)(Math.random()*(height+20));
            g.drawString(message,x,y);
        }
    }
}
```

```
<APPLET code="Javastring.class" WIDTH=400 HEIGHT=350>
<PARAM NAME="message" VALUE="Hello Amrapali!">
</APPLET>
```

2.25 EVENTS IN APPLET

A GUI program doesn't have a main() routine that outlines what will happen when the program is run, in a step-by-step process from beginning to end. Instead, the program must be prepared to respond to various kinds of events that can happen at unpredictable times and in an order that the program doesn't control. The most basic kinds of events are generated by the mouse and keyboard. The user can press any key on the keyboard, move the mouse, or press a button on the mouse. The user can do any of these things at any time, and the computer has to respond appropriately.

NOTES

In Java, events are represented by objects. When an event occurs, the system collects all the information relevant to the event and constructs an object to contain that information. Different types of events are represented by objects belonging to different classes. For example, when the user presses one of the buttons on a mouse, an object belonging to a class called MouseEvent is constructed. The object contains information such as the GUI component on which the user clicked, the (x, y) coordinates of the point in the component where the click occurred, and which button on the mouse was pressed. When the user presses a key on the keyboard, a KeyEvent is created. After the event object is constructed, it is passed as a parameter to a designated subroutine. By writing that subroutine, the programmer says what should happen when the event occurs.

For an event to have any effect, a program must detect the event and react to it. In order to detect an event, the program must "listen" for it. Listening for events is something that is done by an object called an event listener. An event listener object must contain instance methods for handling the events for which it listens. For example, if an object is to serve as a listener for events of type MouseEvent, then it must contain the following method like:

public void mousePressed(MouseEvent evt)

Every event in Java is associated with a GUI component. For example, when the user presses a button on the mouse, the associated component is the one that the user clicked on. Before a listener object can "hear" events associated with a given component, the listener object must be registered with the component.

Now you can yhink that to add an event and listen , there is a large number of details you have to use like:

1. *Put the import specification "import java.awt.event.*;" at the* beginning of your source code
2. Declare that some class implements the appropriate listener interface, such as MouseListener
3. Provide definitions in that class for the subroutines from the interface
4. Register the listener object with the component that will generate the events by calling a method such as addMouseListener() in the component.

Program

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
 public class websites extends Applet implements Runnable, ActionListener
```

NOTES

```
String[] pageTitle = new String[4];
URL[] pageLink = new URL[4];
int current = 0;
Thread runner;
 public void init()
 {
 pageTitle[0] = "Sun Java";
 pageLink[0] = getURL("http://java.sun.com");
 pageTitle[1] = "Microsoft";
 pageLink[1] = getURL("http://www.microsoft.com");
 pageTitle[2] = "Google";
 pageLink[2] = getURL("http://www.google.com");
 pageTitle[3] = "Netscape";
 pageLink[3] = getURL("http://www.netscape.com");
 Button goButton = new Button("Go");
 goButton.addActionListener(this);
 add(goButton);
}
URL getURL(String urlText)
{
URL pageURL = null;
try { pageURL = new URL(getDocumentBase(),
                        urlText);
}
 catch (MalformedURLException m)
  {
  }
 return pageURL;
}
public void paint(Graphics screen)
{
 screen.drawString(pageTitle[current], 5, 60);
 screen.drawString("" + pageLink[current], 5, 80);
}
public void start()
{
 if (runner == null)
```

```
    {
     runner = new Thread(this);
     runner.start();
    }
  }
  public void run()
  {
   while (true)
    {
     repaint();
     current++;
     if (current > 5)
       current = 0;
     try
       {
       Thread.sleep(10000);
       }
      catch (InterruptedException e)
       {
       }
    }
  }
  public void stop()
  {
  if (runner != null)
    {
     runner.stop();
     runner = null;
    }
  }
  public void actionPerformed(ActionEvent evt)
  {
  runner.stop();
   AppletContext browser = getAppletContext();
  if (pageLink[current] != null)
      browser.showDocument(pageLink[current]);
  }
}
```

NOTES

NOTES

2.26 JAVA AWT

The Java programming language provides a class library called the Abstract Window Toolkit (AWT) that contains a number of common graphical widgets. You can add these widgets to your display area and position them with a layout manager.

AWT Basics

All graphical user interface objects items of AWT comes from a common superclass, Component. To create a Graphical User Interface (GUI), you need to add components to a Container object. Because a Container is also a Component, containers may be nested arbitrarily. Most often, you will use a Panel when creating nested GUIs.

Each AWT component uses *native code* to display itself on your screen. When you run a Java application under Microsoft Windows, buttons are *really* Microsoft Windows buttons. When you run the same application on a Macintosh, buttons are *really* Macintosh buttons. When you run on a UNIX machine that uses Motif, buttons are *really* Motif buttons.

Applications versus applets

As you know l that an Applet is a Java program that runs in a web page, while an application is one that runs from the command line. An Applet is a Panel that is automatically inserted into a web page. The browser displaying the web page instantiates and adds *the Applet to the proper part of the web page. The browser tells the Applet when to create its GUI (by calling the init() method of Applet) and when to start() and stop() any special processing.*

Applications run from a command prompt. When you execute an application from the command prompt, the interpreter starts by calling the application's main() method.

Basic AWT—GUI logics

There are three steps you take to create any GUI application or applet:-

1. Compose your GUI by adding components to Container objects.
2. Setup event handlers to respond to user interaction with the GUI.
3. Display the GUI (automatically done for applets, you must explicitly do this for applications).

When you display an AWT—GUI, the interpreter starts a new thread to watch for user interaction with the GUI. This new thread sits and waits until a user presses a key, clicks or moves the mouse, or any other system-level event that affects the GUI. When it receives such an event, it calls one-of the event handlers you set up for the GUI.

Program

```
import java.applet.Applet;
import java.awt.*;
public class AButton extends Applet
 {
  public void init() {
    // STEP 1: Compose the GUI
     Button startButton = new Button("Start with beep sound");
     add(beepButton);
     // STEP 2: Setup Event handlers
     beepButton.addActionListener(new Beeper());
     // STEP 3: Display the GUI (automatic—this is an applet)
  }
}
```

NOTES

Now that program need a event handling that can be set up by adding an instance of a *listener* class to the button. When the button is pressed, a certain method in the listener class is called. In this example, the listener class implements ActionListener (because Button requires it). When the button is pressed, the *button calls* the actionPerformed() method of the listener class. Event handling is discussed in detail later in this module. Suppose you want to produce a "beep" sound when the button is pressed. You can define your event handler under above program as follows:

```
public class Beeper implements ActionListener
{
  public void actionPerformed(ActionEvent event)
   {
    Component c = (Component)event.getSource();
    c.getToolkit().beep();
   }
}
```

When actionPerformed() is called, it produces a beep sound on the computer now your system must be sound capable.

To try this applet, create a simple HTML page and write the given code follows:

```
<html>
  <applet code=AButton.class width=200 height=200>
  </applet>
</html>
```

Then test the HTML page by running *appletviewer* or by loading the

NOTES

HTML file in a browser that supports the Java Runtime Environment (JRE). Note that in this case, the browser must support at least version 1.1 of the JRE, as the example uses the event handling capabilities introduced with that release.

AWT Components

All AWT components are extended from class Component. You can think Component as the parent class for all AWT components. Having this single class is rather useful, as the library designers can put a lot of common code into it. You have to import the related package of a components in your program.

Buttons

A Button has a single line label and may be "pushed" or 'clicked " with a mouse click.

```
import java.awt.*;
import java.applet.Applet;
public class ButtonTest extends Applet
 {
  public void init()
    {
     Button button = new Button("OK");
     add(button);
     }
  }
```

Note that in the above example there is no event handling added; pressing the button will not do anything for that you have to write action listener for this. The AWT button has no direct support for images as labels so you can not show images on a button.

Canvas

A Canvas is a graphical component representing a region where you can draw things such as rectangles, circles, and text strings. The name comes from a painter's canvas. You subclass Canvas to override its default paint() method to define your own components.

```
import java.awt.*;
class DrawingRegion extends Canvas
{
   public DrawingRegion()
  {
    setSize(100, 50);
  }
  public void paint(Graphics g)
```

```
    {
    g.drawRect(0, 0, 99, 49); // draw border
    g.drawString("A Canvas", 20,20);
    }
}
public class CanvasPaintTest extends Applet
  {
    public void init()
    {
    DrawingRegion region = new DrawingRegion();
    add(region);
    }
}
```

The Canvas class is frequently extended to create new component types, for example image buttons. However, starting with the JRE 1.1, you can now directly subclass Component directly to create lightweight, transparent widgets.

Checkbox

A Checkbox is a label with a small pushbutton. The state of a Checkbox is either true (button is checked) or false (button not checked). The default initial state is false. Clicking a Checkbox toggles its state. For example:

Program

```
import java.awt.*;
import java.applet.Applet;
public class CheckboxSimpleTest extends Applet
{
  public void init()
  {
    Checkbox m = new Checkbox("Allow Mixed Case");
   add(m);
  }
}
```

Program

```
import java.awt.*;
import java.applet.Applet;
public class CheckboxSimpleTest2 extends Applet
 {
  public void init()
```

NOTES

```
    {
      Checkbox m = new Checkbox("Label", true);
      add(m);
    }
}
```

CheckboxGroup

A CheckboxGroup is used to control the behavior of a group of Checkbox objects each of which has a true or false state. Exactly one of the Checkbox objects is allowed to be true at one time. Checkbox objects controlled with a CheckboxGroup are usually referred to as "radio buttons". The following example illustrates the basic idea behind radio buttons.

Program

```
import java.awt.*;
import java.applet.Applet;
public class CheckboxGroupTest extends Applet
{
  public void init()
  {
    // create button controller
    CheckboxGroup cbg = new CheckboxGroup();
      Checkbox cb1 =
       new Checkbox("Show lowercase only", cbg, true);
      Checkbox cb2 =
       new Checkbox("Show uppercase only", cbg, false);
      add(cb1);
      add(cb2);
  }
}
```

Choice

Choice objects are drop-down lists. The visible label of the Choice object is the currently selected entry of the Choice.

Program

```
import java.awt.*;
import java.applet.Applet;
public class ChoiceSimpleTest extends Applet
{
  public void init()
   {
```

```
    Choice rgb = new Choice();
    rgb.add("Red");
    rgb.add("Green");
    rgb.add("Blue");
    add(rgb);
  }
}
```

Label

A Label is a displayed Label object. It is usually used to help indicate what other parts of the GUI do, such as the purpose of a neighboring text field.

Program

```
import java.awt.*;
import java.applet.Applet;
 public class LabelTest extends Applet
 {
  public void init()
  {
    add(new Label("A label"));
    // right justify next label
     add(new Label("Another label", Label.RIGHT));
  }
}
```

List

A List is a scrolling list box that allows you to select one or more items. Multiple selections may be used by passing true as the second argument to the constructor.

Program

```
import java.awt.*;
import java.applet.Applet;
  public class ListSimpleTest extends Applet {
  public void init()
  {
    List list = new List(5, false);
    list.add("Saattaal");
    list.add("Nainitaal");
    list.add("Bhimtaal");
```

NOTES

```
        list.add("Khrupataal");
        list.add("Naukchiataal");
        list.add("Sadiataal");
        add(list);
    }
}
```

Scrollbar

A Scrollbar is a "slider" widget with characteristics specified by integer values that are set during Scrollbar construction. Both horizontal and vertical sliders are available.

Program

```
import java.awt.*;
import java.applet.Applet;
// A simple example that makes a Scrollbar appear
public class ScrollbarSimpleTest extends Applet
{
  public void init()
  {
    Scrollbar sb =
      new Scrollbar(Scrollbar.HORIZONTAL,
                    0, // initial value is 0
                    5, // width of slider
                 -100, 105); // range -100 to 100
    add(sb);
  }
}
```

The maximum value of the Scrollbar is determined by subtracting the Scrollbar width from the maximum setting (last parameter).

TextField

A TextField is a scrollable text display object with one row of characters. The preferred width of the field may be specified during construction and an initial string may be specified.

Program

```
import java.awt.*;
import java.applet.Applet;
public class TextFieldSimpleTest extends Applet
  {
```

```
  public void init()
  {
      TextField f1 =
       new TextField("type something");
    add(f1);
  }
}
```

NOTES

Tips:

- Call setEditable(false) to make the field read-only.
- The constructor has an optional width parameter.
- This does not control the number of characters in the TextField, but is merely a suggestion of the preferred width on the screen. Note that layout managers may choose to respect or ignore this preferred width.
- For password fields use field.setEchoChar('?');
- To clear/reset use: field.setEchoChar((char)0);

TextArea

A TextArea is a multi-row text field that displays a single string of characters, where newline ('\n' or '\n\r' or '\r', depending on platform) ends each row. The width and height of the field is set at construction, but the text can be scrolled up/down and left/right.

Program

```
import java.awt.*;
import java.applet.Applet;
public class TextAreaSimpleTest extends Applet
{
  TextArea disp;
  public void init()
{
    disp = new TextArea("Code goes here", 10, 30);
    add(disp);
  }
}
```

There is no way, for example, to put the cursor at beginning of row five, only to put the cursor at single dimension position 50. There is a four-argument constructor that accepts a fourth parameter of a scrollbar policy. The different settings are the class constants:

1. SCROLLBARS_BOTH

NOTES

2. SCROLLBARS_HORIZONTAL_ONLY
3. SCROLLBARS_NONE
4. SCROLLBARS_VERTICAL_ONLY.

Program

```
import java.awt.*;
import java.applet.Applet;
public class TextAreaScroll extends Applet
{
  String s =
    "This is a very long message " +
    "It should wrap when there is " +
    "no horizontal scrollbar.";
  public void init()
 {
    add(new TextArea (s, 4, 15,
      TextArea.SCROLLBARS_NONE));
    add(new TextArea (s, 4, 15,
      TextArea.SCROLLBARS_BOTH));
    add(new TextArea (s, 4, 15,
      TextArea.SCROLLBARS_HORIZONTAL_ONLY));
    add(new TextArea (s, 4, 15,
      TextArea.SCROLLBARS_VERTICAL_ONLY));
  }
}
```

Common Component Methods

All AWT components share the 100-plus methods inherited from the Component class. Some of the most useful and commonly-used methods are given for you :

getSize()—Gets current size of component

```
Dimension d = someComponent.getSize();
int height = d.height;
int width = d.width;
```

getLocation() - Gets position of component, *relative to containing component.*

```
Point p = someComponent.getLocation();
int x = p.x;
int y = p.y;
```

NOTES

getLocationOnScreen()—Gets the position of the component *relative to the upper-left corner of the computer screen*

```
Point p = someComponent.getLocationOnScreen();
int x = p.x;
int y = p.y;
```

getBounds()—Gets current bounding of component.

```
Rectangle r = someComponent.getBounds();
int height = r.height;
int width = r.width;
int x = r.x;
int y = r.y;
```

setEnabled(boolean)—*Toggles the state of the component. If set to* true, the component will react to user input and appear normal. If set to false, the component will ignore user interaction, and usually appear *ghosted* or *grayed-out.*

setVisible(boolean)—Toggles the visibility state of the component. If set to true, the component will appear on the screen *if* it is contained in a visible container. If false, the component will not appear on the screen. Note that if a component is marked as not visible, any layout manager that is responsible for that component will usually proceed with the layout algorithm *as though the component were not in the parent container!* This means that making a component invisible *will not* simply make it disappear while reserving its space in the GUI. Making the component invisible will cause the layout of its sibling components to readjust!

setBackground(Color)/setForeground(Color)—Changes component background/foreground colors.

setFont(Font)—Changes font of text within a component.

Containers

A Container is a Component, so may be nested. Class Panel is the most commonly-used Panel and can be extended to partition GUIs. Class Applet is a specialized Panel for running programs within a browser.

Common Container Methods

Besides the 100-plus methods inherited from the Component class, all Container subclasses inherit the behavior of about 50 common methods of Container (most of which just override a method of Component). While the most common method of Container used add(), has already been briefly discussed, if you need to access the list of components within a container, you may find the getComponentCount(), getComponents(), and getComponent(int) methods helpful.

NOTES

ScrollPane

The ScrollPane container was introduced with the 1.1 release of the Java Runtime Environment (JRE) to provide a new Container with automatic scrolling of any one *large Component. That large object could be anything from an image that is too big for the display area to a bunch of spreadsheet cells. All the event handling mechanisms for scrolling are managed for you. Also, there is no LayoutManager for a ScrollPane since there is only a single object within it.*

The following example demonstrates the scrolling of a large image. Since an Image object is not a Component, the image must be drawn by a component such as a Canvas.

Program

```
import java.awt.*;
import java.applet.*;
 class ImageCanvas extends Component
{
  private Image image;
  public ImageCanvas(Image i)
 {
    image = i;
 }
  public void paint(Graphics g)
{
    if (image != null)
      g.drawImage(image, 0, 0, this);
 }
}
 public class ScrollingImage extends Applet
 {
  public void init() {
     setLayout(new BorderLayout());
    ScrollPane sp = new ScrollPane(ScrollPane.SCROLLBARS_ALWAYS);
     Image im = getImage(getCodeBase(), "./images/abc.gif");
     sp.add(new ImageCanvas(im));
     add(sp, BorderLayout.CENTER);
 }
}
```

2.27 EVENT HANDLING

NOTES

Events

Beginning with the 1.1 version of the JRE, objects register as *listeners* for events. If there are no listeners when an event happens, nothing happens. If there are twenty listeners registered, each is given an opportunity to process the event, in an undefined order. With a Button, for example, activating the button notifies any registered ActionListener objects. Consider SimpleButtonEvent applet which creates a Button instance and registers itself as the listener for the button's action events like:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
 public class SimpleButtonEvent extends Applet
       implements ActionListener
{
  private Button b;
  public void init()
 {
    b = new Button("Press me");
    b.addActionListener(this);
   add(b);
 }

  public void actionPerformed(ActionEvent e)
 {
   // If the target of the event was our Button
   // In this example, the check is not
   //   truly necessary as we only listen to
   //   a single button
   if ( e.getSource() == b )
  {
     getGraphics().drawString("hello dear ",20,20);
   }
 }
}
```

NOTES

Notice that any class can implement ActionListener, including, in this case, the applet itself. All listeners are always notified. If you don't want an event to be processed further, you can call the AWTEvent.consume() method. However each listener would then need to check for consumption using the isConsumed() method. Consuming events primarily stops events from being processed by the system, after every listener is notified. So, if you want to reject keyboard input from the user, you can consume() the KeyEvent. All the KeyListener implementers will still be notified, but the character will not be displayed. (Consumption only works for InputEvent and its subclasses.

So, here is how everything works:

- Components generate subclasses of AWTEvent when something interesting happens.
- Event sources permit any class to be a listener using the addABCListener () method, where ABC is the event type you can listen for, for example addActionListener(). You can also remove listeners using the removeABCListener() methods. If there is an add/removeABCListener() pair, then the component is a source for the event when the appropriate action happens.
- In order to be an *event handler* you have to implement the listener type, otherwise, you cannot be added, ActionListener being one such type.
- Some listener types are special and require you to implement multiple methods. For instance, if you are interested in key events, and register a KeyListener, you have to implement three methods, one for key press, one for key release, and one for both, key typed. If you only care about key typed events, it doesn't make sense to have to stub out the other two methods. There are special classes out there called adapters that implement the listener interfaces and stub out all the methods. Then, you only need to subclass the adapter and override the necessary method(s).

2.28 AWT EVENTS

Events subclass the AWTEvent class. And nearly every event-type has an associated Listener *interface, PaintEvent and InputEvent do not. (With PaintEvent, you just override paint() and update(), for InputEvent, you listen for subclass events, since it is abstract.*

Low-level Events

Low-level events represent a low-level input or window operation, like a key press, mouse movement, or window opening. The following table displays the different low-level events, and the operations that generate each event (each operation corresponds to a method of the listener interface) like:

NOTES

ComponentEvent	Hiding, moving, resizing, showing
ContainerEvent	Adding/removing component
FocusEvent	Getting/losing focus
KeyEvent	Pressing, releasing, or typing (both) a key
MouseEvent	Clicking, dragging, entering, exiting, moving, pressing, or releasing
WindowEvent	Iconifying, deiconifying, opening, closing, really closed, activating, deactivating.

For instance, typing the letter 'A' on the keyboard generates three events, one for pressing, one for releasing, and one for typing. Depending upon your interests, you can do something for any of the three events.

Semantic Events

Semantic events represent interaction with a GUI component; for instance selecting a button, or changing the text of a text field. Which components generate which events is shown in the next section.

ActionEvent	Do the command
AdjustmentEvent	Value adjusted
ItemEvent	State changed
TextEvent	Text changed

Event Sources

The following table represents the different event sources. Keep in mind the object hierarchy. For instance, when Component is an event source for something, so are all its subclasses:

Low-level Events	
Component	ComponentListenerFocus
	ListenerKeyListenerMouse
	ListenerMouseMotionListener
Container	ContainerListener
Window	WindowListener

Semantic Events	
ButtonList	
MenuItem	ActionListener
Text Field	
Choice	

NOTES

Checkbox	
Checkbox	
Checkbox	ItemListener
MenuItem	
List	
Scrollbar	AdjustmentListener
TextArea	TextListener
TextField	

Notice that although there is only one MouseEvent class, the listeners are spread across two interfaces. This is for performance issues. Since motion mouse events are generated more frequently, if you have no interest in them, you can ignore them more easily, without the performance hit.

Event Listeners

Each listener interface is paired with one event type and contains a method for each type of event the event class embodies. For instance, the KeyListener contains three methods, one for each type of event that the KeyEvent has: keyPressed(), keyReleased(), and keyTyped().

Summary of Listener interfaces and their methods

Interface	Method(s)
ActionListener	actionPerformed(ActionEvent e)
AdjustmentListener	adjustmentValueChanged (AdjustmentEvent e)
ComponentListener	componentHidden (ComponentEvent e)
	componentMoved(ComponentEvent e)
	componentResized(ComponentEvent e)
	componentShown(ComponentEvent e)
ContainerListener	componentAdded(ContainerEvent e)
	componentRemoved(ContainerEvent e)
FocusListener	focusGained(FocusEvent e)
	focusLost(FocusEvent e)
ItemListener	itemStateChanged(ItemEvent e)
KeyListener	keyPressed(KeyEvent e)
	keyReleased(KeyEvent e)
	keyTyped(KeyEvent e)
MouseListener	mouseClicked(MouseEvent e)
	mouseEntered(MouseEvent e)
	mouseExited(MouseEvent e)

NOTES

	mousePressed(MouseEvent e)
	mouseReleased(MouseEvent e)
MouseMotionListener	mouseDragged(MouseEvent e)
	mouseMoved(MouseEvent e)
TextListener	textValueChanged(TextEvent e)
WindowListener	windowActivated(WindowEvent e)
	windowClosed(WindowEvent e)
	windowClosing(WindowEvent e)
	windowDeactivated(WindowEvent e)
	windowDeiconified(WindowEvent e)
	windowIconified(WindowEvent e)
	windowOpened(WindowEvent e)

Event Adapters

Since the low-level event listeners have multiple methods to implement, there are event adapter classes to ease the pain. Instead of implementing the interface and stubbing out the methods you do not care about, you can subclass the appropriate adapter class and just override the one or two methods you are interested in. Since the semantic listeners only contain one method to implement, there is no need for adapter classes.

Program

```
public class MyKeyAdapter extends KeyAdapter
{
  public void keyTyped(KeyEvent e)
 {
    System.out.println("User typed: " +
       KeyEvent.getKeyText(e.getKeyCode()));
 }
}
```

Button Pressing Example

The following code demonstrates the basic concept a little more beyond the earlier example. There are three buttons within a Frame, their displayed labels may be internationalized so you need to preserve their purpose within a command *associated with the button. Based upon which button is pressed, a different action occurs.*

Program

```
import java.awt.*;
import java.awt.event.*;
```

NOTES

```
public class Activator
{
  public static void main(String[] args)
  {
    Button b;
    ActionListener al = new MyActionListener();
    Frame f = new Frame("Hello Java");
    f.add(b = new Button("Hola"),
                BorderLayout.NORTH);
    b.setActionCommand("Hello");
    b.addActionListener(al);
    f.add(b = new Button("Aloha"),
                BorderLayout.CENTER);
    b.addActionListener(al);
    f.add(b = new Button("Adios"),
                BorderLayout.SOUTH);
    b.setActionCommand("Quit");
    b.addActionListener(al);
    f.pack();
    f.show();
  }
}
class MyActionListener implements ActionListener
{
  public void actionPerformed(ActionEvent e)
{
    // Action Command is not necessarily label
    String s = e.getActionCommand();
    if (s.equals("Quit")) {
      System.exit(0);
    }
    else if (s.equals("Hello")) {
      System.out.println("Bon Jour");
    }
    else {
      System.out.println(s + " selected");
    }
}
}
```

Since this is an application, you need to save the source (as Activator.java), compile it, and run it outside the browser. Also, if you wanted to avoid checking which button was selected, you can associate a different ActionListener to each button, instead of one to all. This is actually how many Integrated Development Environments (IDEs) generate their code.

Adapters Example

The following code demonstrates using an adapter as an anonymous inner class to draw a rectangle within an applet. The mouse press signifies the top left corner to draw, with the mouse release the bottom right.

Program

```
import java.awt.*;
import java.awt.event.*;
public class Draw extends java.applet.Applet
{
  public void init()
{
    addMouseListener(
      new MouseAdapter()
     {
         int savedX, savedY;
          public void mousePressed(MouseEvent e)
        {
          savedX = e.getX();
          savedY = e.getY();
        }
          public void mouseReleased(MouseEvent e)
        {
          Graphics g = Draw.this.getGraphics();
          g.drawRect(savedX, savedY,
                    e.getX()-savedX,
                    e.getY()-savedY);
        }
     }
    }
}
```

NOTES

AWT Applications and Menus

GUI-based Applications

To create a window for your application, define a subclass of Frame (a Window with a title, menubar, and border) and have the main method construct an instance of that class. Applications respond to events in the same way as applets do. The following example, BasicApplication, responds to the native window toolkit quit, or closing, operation:

Program

```
import java.awt.*;
import java.awt.event.*;

public class BasicApplication extends Frame
{
   public BasicApplication()
{
     super("BasicApplication Title");
    setSize(200, 200);
    // add a demo component to this frame
     add(new Label("Application Template...",
                                  Label.CENTER),
     BorderLayout.CENTER);
      addWindowListener(new .WindowAdapter()
    {
       public void windowClosing(WindowEvent e)
   {
         setV..ible(false); dispose();
        System.exit(0);
      }
    }
  }
public static void main(String[] args)
 {
    BasicApplication app =
                    new BasicApplication();
    app.setVisible(true);
  }
}
```

Now consider an application that displays the x, y location of the last mouse click and provides a button to reset the displayed x, y coordinates to 0, 0:

Program

```
import java.awt.*;
import java.awt.event.*;
public class CursorFrame extends Frame
{
  TextField a, b;
  Button btn;
  public CursorFrame()
  {
     super("CursorFrame");
    setSize(400, 200);
     setLayout(new FlowLayout());
     add(new Label("Click the mouse..."));
    a = new TextField("0", 4);
    b = new TextField("0", 4);
    btn = new Button("RESET");
    add(a); add(b); add(btn);
     addMouseListener(new MouseAdapter() {
       public void mousePressed(MouseEvent e) {
         a.setText(String.valueOf(e.getX()));
         b.setText(String.valueOf(e.getY()));
      }
    }
 addWindowListener(new WindowAdapter()
    {
       public void windowClosing(WindowEvent e)
{
        setVisible(false);
        dispose();
       System.exit(0);
      }
    }
      btn.addActionListener(new ActionListener()
{
       public void actionPerformed(ActionEvent e)
{
```

- NOTES

NOTES

```
            a.setText("0");
            b.setText("0");
        }
    }
}
public static void main(String[] args)
{
    CursorFrame app = new CursorFrame();
    app.setVisible(true);
}
}
```

This application provides anonymous classes to handle mouse events, application window closing events, and the action event for resetting the text fields that report mouse coordinates. When you have a very common operation, such as handling application window closing events, it often makes sense to abstract out this behavior and handle it elsewhere. In this case, it's logical to do this by extending the existing Frame class, creating the specialization AppFrame like:

Program

```
import java.awt.*;
import java.awt.event.*;
public class AppFrame extends Frame
    implements WindowListener
{
 public AppFrame(String title)
{
    super(title);
    addWindowListener(this);
 }
  public void windowClosing(WindowEvent e)
{
    setVisible(false);
    dispose();
   System.exit(0);
 }
  public void windowClosed(WindowEvent e) {}
  public void windowDeactivated(WindowEvent e) {}
  public void windowActivated(WindowEvent e) {}
```

```
  public void windowDeiconified(WindowEvent e) {}
  public void windowIconified(WindowEvent e) {}
  public void windowOpened(WindowEvent e) {}
}
```

NOTES

Dialog Boxes

A Dialog is a window that requires input from the user. Components may be added to the Dialog like any other container. Like a Frame, a Dialog is initially invisible. You must call the method setVisible() to activate the dialog box like:

```
import java.awt.*;
import java.awt.event.*;
public class DialogFrame extends AppFrame
{
  Dialog d;
public DialogFrame()
  {
    super("DialogFrame");
   setSize(200, 100);
   Button btn, dbtn;
   add(btn = new Button("Press for Dialog Box"),
     BorderLayout.SOUTH);
   d = new Dialog(this, "Dialog Box", false);
   d.setSize(150, 150);
   d.add(new Label("This is the dialog box."),
     BorderLayout.CENTER);
   d.add(dbtn = new Button("OK"),
     BorderLayout.SOUTH);
    btn.addActionListener(new ActionListener()
   {
      public void actionPerformed(ActionEvent e)
  {
       d.setVisible(true);
     }
    }
     dbtn.addActionListener(new ActionListener()
  {
      public void actionPerformed(ActionEvent e)
```

NOTES

```
{
        d.setVisible(false);
      }
    }
    d.addWindowListener(new WindowAdapter()
 {
       public void windowClosing(WindowEvent e)
{
        d.setVisible(false);
      }
    }
  }
  public static void main(String[] args)
{
    DialogFrame app = new DialogFrame();
    app.setVisible(true);
  }
}
```

AWT Menus

An application can have a MenuBar object containing Menu objects that are comprised of MenuItem objects. Each MenuItem can be a string, menu, checkbox, or separator (a line across the menu).

To add menus to any Frame or subclass of Frame:

Create a MenuBar

MenuBar mb = new MenuBar();

Create a Menu

Menu m = new Menu("File");

Add each Menu to the MenuBar in the order you want them to appear, from left to right.

mb.add(m); // add File menu to bar

Add the MenuBar to the Frame by calling the setMenuBar() method.

setMenuBar(mb); // set menu bar of your Frame create your MenuItem choices and add each to the Menu, in the order you want them to appear, from top to bottom Like:

```
import java.awt.*;
import java.awt.event.*;
// Make a main window with two top-level menus: File and Help.
// Help has a submenu and demonstrates a few interesting menu items.
```

```
public class MainWindow extends Frame
{
  public MainWindow()
  {
    super("Menu System Test Window");
    setSize(200, 200);
     // make a top level File menu
    FileMenu fileMenu = new FileMenu(this);
     // make a top level Help menu
    HelpMenu helpMenu = new HelpMenu(this);
     // make a menu bar for this frame
    // and add top level menus File and Menu
    MenuBar mb = new MenuBar();
    mb.add(fileMenu);
    mb.add(helpMenu);
    setMenuBar(mb);
     addWindowListener(new WindowAdapter() {
       public void windowClosing(WindowEvent e) {
        exit();
      }
    }
  }
  public void exit()
{
    setVisible(false); // hide the Frame
     dispose(); // tell windowing system to free resources
    System.exit(0); // exit
  }

  public static void main(String args[])
{
    MainWindow w = new MainWindow();
    w.setVisible(true);
  }
}
// Encapsulate the look and behavior of the File menu
class FileMenu extends Menu implements ActionListener
```

NOTES

```
{
  MainWindow mw;  // who owns us?
  public FileMenu(MainWindow m)
{
    super("File");
    mw = m;
    MenuItem mi;
    add(mi = new MenuItem("Open"));
    mi.addActionListener(this);
    add(mi = new MenuItem("Close"));
    mi.addActionListener(this);
    add(mi = new MenuItem("Exit"));
    mi.addActionListener(this);
  }
  // respond to the Exit menu choice
  public void actionPerformed(ActionEvent e)
 {
    String item = e.getActionCommand();
    if (item.equals("Exit"))
      mw.exit();
    else
      System.out.println("Selected FileMenu " + item);
  }
}
// Encapsulate the look and behavior of the Help menu
class HelpMenu extends Menu implements ActionListener
{
  MainWindow mw;  // who owns us?
  public HelpMenu(MainWindow m)
{
    super("Help");
    mw = m;
    MenuItem mi;
    add(mi = new MenuItem("Fundamentals"));
    mi.addActionListener(this);
    add(mi = new MenuItem("Advanced"));
    mi.addActionListener(this);
```

NOTES

```
    addSeparator();
    add(mi = new CheckboxMenuItem("Have Read The Manual"));
    mi.addActionListener(this);
    add(mi = new CheckboxMenuItem("Have Not Read The Manual"));
    mi.addActionListener(this);
    // make a Misc sub menu of Help menu
    Menu subMenu = new Menu("Misc");
    subMenu.add(mi = new MenuItem("Help!!!"));
    mi.addActionListener(this);
    subMenu.add(mi = new MenuItem("Why did that happen?"));
    mi.addActionListener(this);
    add(subMenu);
  }
  // respond to a few menu items
   public void actionPerformed(ActionEvent e)
{
    String item = e.getActionCommand();
    if (item.equals("Fundamentals"))
      System.out.println("Fundamentals");
    else if (item.equals("Help!!!"))
      System.out.println("Help!!!");
    // etc...
  }
}
```

Menu Shortcuts

One nice feature of the MenuItem class is its ability to provide menu shortcuts or speed keys. For instance, in most applications that provide printing capabilities, pressing Ctrl-P initiates the printing process. When you create a MenuItem you can specify the shortcut associated with it. If the user happens to press the speed key, the action event is triggered for the menu item.

The following code creates two menu items with speed keys, Ctrl-P for Print and Shift-Ctrl-P for Print Preview:

```
file.add (mi = new MenuItem ("Print",
new MenuShortcut('p')));
file.add (mi = new MenuItem ("Print Preview",
new MenuShortcut('p', true)));
```

The example above uses Ctrl-P and Shift-Ctrl-P shortcuts on Windows/

NOTES

Motif. The use of *Ctrl* for the shortcut key is defined by the Toolkit method getMenuShortcutKeyMask(). For the Macintosh, this would be the *Command* key. An optional boolean parameter to the constructor determines the need for the *Shift* key appropriate to the platform.

Pop-up Menus

One restriction of the Menu class is that it can only be added to a Frame. If you want a menu in an Applet, you are out of luck (unless you use the Swing component set). While not necessarily a perfect solution, you can associate a pop-up menu with any Component, of which Applet is a subclass. A PopupMenu is similar to a Menu in that it holds MenuItem objects. However, instead of appearing at the top of a Frame, you pop the popup menu up over any component, usually when the user generates the appropriate mouse event.

The actual mouse interaction to generate the event is platform specific so there is the means to determine if a MouseEvent triggers the pop-up menu using the MouseEvent.isPopupTrigger() method. It is then your responsibility to position and display the PopupMenu.

The following program, PopupApplication, demonstrates this portable triggering of a pop-up menu, as well as activating a pop-up menu from a command button:

```
import java.awt.*;
import java.awt.event.*;
public class PopupApplication extends AppFrame
{
  Button btn; TextField msg; PopupAppMenu m;
  public PopupApplication() {
    super("PopupApplication");
   setSize(200, 200);
   btn = new Button("Press for pop-up menu...");
   add(btn, BorderLayout.NORTH);
   msg = new TextField();
   msg.setEditable(false);
   add(msg, BorderLayout.SOUTH);
   m = new PopupAppMenu(this);
   add(m);
       btn.addActionListener(new ActionListener()
     {
     public void actionPerformed(ActionEvent e)
      {
      m.show(btn, 10, 10);
   }
```

```
    });
     addMouseListener(new MouseAdapter()
      {
       public void mousePressed(MouseEvent e)
       {
         if (e.isPopupTrigger())
            m.show(e.getComponent(), e.getX(), e.getY());
       }
       public void mouseReleased(MouseEvent e)
        {
         if (e.isPopupTrigger())
           m.show(e.getComponent(), e.getX(); e.getY());
       }
     });
  }
  public static void main(String[] args)
    {
    PopupApplication app = new PopupApplication();
    app.setVisible(true);
  }
}
class PopupAppMenu extends PopupMenu
  implements ActionListener
  {
  PopupApplication ref;
  public PopupAppMenu(PopupApplication ref)
   {
    super("File");
    this.ref = ref;
    MenuItem mi;
    add(mi = new MenuItem("Copy"));
    mi.addActionListener(this);
    add(mi = new MenuItem("Cut"));
    mi.addActionListener(this);
    add(mi = new MenuItem("Paste"));
    mi.addActionListener(this);
  }
  public void actionPerformed(ActionEvent e)
  {
    String item = e.getActionCommand();
```

NOTES

```
ref.msg.setText("
                Selected menu item: " + item);
    }
}
```

NOTES

SUMMARY

- Java is just a small, simple, safe, object-oriented, interpreted or dynamically optimized, byte-coded, architecture-neutral, garbage-collected, multithreaded programming language with a strongly typed exception-handling mechanism for writing distributed, dynamically extensible programs.
- A statement in Java indicates the simplest tasks you can do in Java, a statement forms a single Java operation. All the following are simple Java statements:

```
int poo= 1;
import java.io.*;
System.out.println("This is a " + name " " + name1);
Train.engine = true;
```

- Variables are locations in primary memory (RAM) in which your program values can be stored. Each one has a name, a type, and a value.
- Java actually has three kinds of variables:
  1. Instance variables
  2. Class variables
  3. Local variables.
- Most of the expressions in Java use operators. Operators are special symbols used to perform operations like arithmetic, assignment, increment and decrement, logical operations etc.
- A loop in Java like in C or C++, repeats a statement or block of statements until a condition is matched. There are three types of loops in Java they are the for loop, while loop and the do while loop.
- Arrays in Java, as in other languages, are a way to store collections of items into a single unit. An array is a collection of items.
- Java Methods in a class are used to define an object's behaviour-what happens when that object is created and the various operations that object can perform during its lifetime, commonly known as functions in C++.
- Packages, are a way of organizing groups of classes in Java. A package contains any number of classes that are related in purpose, in scope, or by inheritance.

- Exceptions can be used as a means of indicating other situations as well. Java has exceptions to deal with managing, creating, and expecting errors and other unusual situations.
- A stream is a path of communication between the source of some information and its destination. This information can come from a file, the computer's memory, or even from the Internet.
- Java is a programming language designed for networked computers and the World Wide Web. Java applets are downloaded over a network to appear on a Web page.
- A Dialog is a window that requires input from the user. Components may be added to the Dialog like any other container. Like a Frame, a Dialog is initially invisible.

REVIEW QUESTIONS

1. Why Java considered as platform neutral?
2. Write a small Java application to print your University name with your course and study center name in three lines.
3. What is java development kit?
4. What is a variable and a constant? What are the different rules to declare a variable?
5. Name the eight data types used in Java.
6. Suppose you need to write a Java program that calculates a labour salary by given days and per day salary of work. Write declarations for the variables you will use.
7. List all the probable operators in java.
8. List all the white spaces that are allowed in java.
9. How can you identify a comment in java?
10. What are various data types allowed in jāva?
11. Write Java program to print sum of first 50 prime numbers.
12. How do you know when to use a for loop?
13. What are the three parts of a for loop?
14. When does a for loop stop looping?
15. How can a for loop count backward?
16. How can a for loop count by tens?
17. Is it possible to create an infinite loop with a for loop?

NOTES

18. How many times is a while loop guaranteed to execute?
19. Why is it important to initialize a loop control variable?
20. What's an infinite loop?
21. Compare and contrast while and do-while loops?
22. What is an array?
23. What is an array subscript? How is a subscript like an index?
24. If you had an array of 30 integers, what is the largest valid subscript?
25. What happens if you try to access a nonexistent array element?
26. Describe why a for loop is appropriate for accessing an array?
27. What are class variables and methods.
28. Write differences between objects and the primitive data types.
29. Write a program to show examples of calling a method with a different number of arguments.
30. What is inheritance?
31. What are different types of inheritance, describe with examples.
32. How do you show the declaration of a multiple class inheritance?
33. How do you add a class or interface to a package?
34. How does the complete name of a package relate to the package's storage on your disk?
35. Do you have to catch all types of exceptions that might be thrown by Java?
36. How do you pass an exception up from a called method to the calling method?
37. What are the two main types of exceptions that Java may throw?
38. What are Java streams, whish is base class or all streams.
39. Write a Java program to add 5 float numbers entererd through keyboard.
40. How are threads similar to multitasking?
41. What Java interface must be implemented by all threads?
42. How can you take advantage of an applet's life cycle in order to retain a thread's state as the user switches between Web pages.
43. How can someone without a Java-compatible browser run applets?
44. What does the optional codebase attribute do?
45. Explain the client/server relationship as it applies to Java applets.
46. How will the client/server focus of the Internet change as applications start to flow two ways, both from and to a remote computer?
47. What are the five life-cycle stages of an applet?
48. What two attributes of a label object can be set using the class's methods?

49. How can you change the label displayed in a button control?
50. When a user clicks a button control, what arguments are received by the action() method?
51. In an applet with several button controls, how can you determine which button was clicked?
52. What happens when the user clicks a label?
53. What are the three arguments required by the Checkbox class's constructor?
54. What's another name for checkboxes that are set for exclusive mode?
55. What are the two arguments needed by the TextField class's constructor?
56. What method do you call in order to change the state of a checkbox?
57. What is the difference between the nonexclusive and exclusive modes for a checkbox control?
58. What additional object do you need to group the items in a checkbox control?
59. When would you use echo characters with a textfield control?
60. How do you set a textfield control's echo character?
61. How can you determine which checkbox in an applet generated an event?

FURTHER READINGS

- *'Internet and Java programming'*, by Harish Kumar Taluja. Firewall Media.
- *'Programming Engineering Computation in Java'*. by Dr. Raja Subramaniam, Laxmi Publications (p) Ltd.
- *'Internet its Applications with HTML and VB Script'*, by Shashi Bansal, University Science Press.

NOTES

UNIT 3 JAVA SWINGS

★ STRUCTURE ★

3.0 Learning Objectives
3.1 Introduction
3.2 Swing Components
3.3 JDBC
- *Summary*
- *Review Questions*
- *Further Readings*

3.0 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- illustrate features of Java Swings.
- create a swing applit.
- define about all components of Java Swings.
- describe JDBC.

3.1 INTRODUCTION

Main Features of Swings are:

- *Lightweight:* Not built on native window-system windows.
- *Much bigger set of built-in controls:* Trees, image buttons, tabbed panes, sliders, toolbars, color choosers, tables, text areas to display HTML or RTF, etc.
- *Much more customizable:* Can change border, text alignment, or add image to almost any control. Can customize how minor features are drawn. Can separate internal representation from visual appearance.
- *"Pluggable" look and feel:* Can change look and feel at runtime, or design own look and feel.
- *Many miscellaneous new features:* Double-buffering built in, tool tips, dockable tool bars, keyboard accelerators, custom cursors, etc.

3.2 SWING COMPONENTS

NOTES

JApplet

In simple AWT applications you need to write an applet , while in swing you will write Japplet like:

```
import java.awt.*;
import javax.swing.*;
public class JAppletExample extends JApplet
{
  public void init()
{
    WindowUtilities.setNativeLookAndFeel();
    Container content = getContentPane();
    content.setBackground(Color.white);
    content.setLayout(new FlowLayout());
    content.add(new JButton("Button 1"));
    content.add(new JButton("Button 2"));
    content.add(new JButton("Button 3"));
  }
}
```

JFrame

JFrame is like AWT frame , it is starting point for graphical applications. You need to create a JFrame as a container for other graphical objects. Components go in the "content pane", not directly in the frame. Changing other properties (layout manager, background color, etc.) also apply to the content pane. Access content pane via getContentPane, or if you want to replace the content pane with your container (*e.g.*, a JPanel), use setContentPane. JFrames close automatically when you click on the close button (unlike AWT Frames). However, closing the last JFrame does not result in your program exiting Java. So your "main" JFrame still needs a WindowListener.

Program

```
import java.awt.*;
import javax.swing.*;
public class JFrameExample
{
  public static void main(String[] args)
{
    WindowUtilities.setNativeLookAndFeel();
```

NOTES

```
        JFrame f = new JFrame("This is a test");
        f.setSize(400, 150);
        Container content = f.getContentPane();
        content.setBackground(Color.white);
        content.setLayout(new FlowLayout());
        content.add(new JButton("Button 1"));
        content.add(new JButton("Button 2"));
        content.add(new JButton("Button 3"));
        f.addWindowListener(new ExitListener());
        f.setVisible(true);
    }
}
class ExitListener extends WindowAdapter
{
    public void windowClosing(WindowEvent event)
{
      System.exit(0);
    }
}
```

JButton

Simple uses of JButton are very similar to Button. You create a JButton with a String as a label, and then drop it in a window. Events are normally handled just as with a Button: you attach an ActionListener via the addActionListener method. The most obvious new feature is the ability to associate images with buttons. Swing introduced a utility class called ImageIcon that lets you very easily specify an image file (jpeg or GIF, including animated GIFs). Many Swing controls allow the inclusion of icons. The simplest way to associate an image with a JButton is to pass the ImageIcon to the constructor, either in place of the text or in addition to it. However, a JButton actually allows seven associated images like:

1. the main image (use setIcon to specify it if not supplied in the constructor),
2. the image to use when the button is pressed (setPressedIcon),
3. the image to use when the mouse is over it (setRolloverIcon, but you need to call setRolloverEnabled(true) first),
4. the image to use when the button is selected and enabled (setSelectedIcon),
5. the image to use when the button is disabled (setDisabledIcon),
6. the image to use when it is selected but disabled (setDisabledSelectedIcon), and

7. the image to use when the mouse is over it while it is selected (setRolloverSelectedIcon).

You can also change the alignment of the text or icon in the button (setHorizontalAlignment and setVerticalAlignment; only valid if button is larger than preferred size), and change where the text is relative to the icon (setHorizontalTextPosition, setVerticalTextPosition). You can also easily set keyboard mnemonics via setMnemonic. This results in the specified character being underlined on the button, and also results in ALT-*char* activating the button.

NOTES

```
import java.awt.*;
import javax.swing.*;
public class JButtons extends JFrame
{
  public static void main(String[] args)
{
    new JButtons();
  }
  public JButtons()
 {
    super("Using JButton");
     WindowUtilities.setNativeLookAndFeel();
     addWindowListener(new  ExitListener());
    Container content = getContentPane();
     content.setBackground(Color.white);
    content.setLayout(new  FlowLayout());
   JButton button1 = new JButton("Java");
    content.add(button1);
   ImageIcon cup = new ImageIcon("cup.gif");          //you must
have cup.gif file .or use other icon file
   JButton button2 = new JButton(cup);
    content.add(button2);
   JButton button3 = new JButton("Java", cup);
    content.add(button3);
   JButton button4 = new JButton("Java", cup);
    button4.setHorizontalTextPosition(SwingConstants.LEFT);
    content.add(button4);
   pack();
    setVisible(true);
  }
}
```

NOTES

JLabel

In many cases, JLabel is used exactly like Label: as a way to display text. However, since a JLabel can have an image instead of or in addition to the text, it is alsofrequently used as a way to put an image in a display. A JLabel has three major features that Label does not. The first is the ability to display images, usually by supplying an ImageIcon eiher to the constructor or via a call to setIcon. The use of icons in JLabel is just like the use in JButton,The second new feature is the ability to place borders around the labels. The third new feature, and the one I am focusing on here, is the ability to use HTML to format the label. The idea is that, if the string for the label begins with "<html>", then the string is interpreted as HTML rather than taken literally. This lets you make multi-line labels, labels with mixed colors and fonts, and various other fancy effects.

```
import java.awt.*;
import javax.swing.*;
public class JLabels extends JFrame
{
  public static void main(String[] args)
{
    new JLabels();
  }
  public JLabels()
{
    super("Using HTML in JLabels");
     WindowUtilities.setNativeLookAndFeel();
     addWindowListener(new ExitListener());
    Container content = getContentPane();
    Font font = new Font("Serif", Font.PLAIN, 30);
    content.setFont(font);
    String labelText =
      "<html><FONT COLOR=RED>Red</FONT> and " +
      "<FONT COLOR=BLUE>Blue</FONT> Text</html>";
    JLabel coloredLabel = new JLabel(labelText, JLabel.CENTER);
     coloredLabel.setBorder
        (BorderFactory.createTitledBorder("Mixed Colors"));
     content.add(coloredLabel, BorderLayout.NORTH);
    labelText =
      "<html><B>Bold</B> and <I>Italic</I> Text</html>";
```

```
    JLabel boldLabel =
      new JLabel(labelText, JLabel.CENTER);
    boldLabel.setBorder
       (BorderFactory.createTitledBorder("Mixed Fonts"));
    content.add(boldLabel, BorderLayout.CENTER);
    labelText =
      "<html>The Applied Physics Laboratory is a division " +
      "of the Johns Hopkins University." +
      "<P>" +
      "Major JHU divisions include:" +
      "<UL>" +
      "<LI>The Applied Physics Laboratory" +
      "<LI>The Krieger School of Arts and Sciences" +
      "<LI>The Whiting School of Engineering" +
      "<LI>The School of Medicine" +
      "<LI>The School of Public Health" +
      "<LI>The School of Nursing" +
      "<LI>The Peabody Institute" +
      "<LI>The Nitze School of Advanced International Studies" +
      "</UL>";
    JLabel fancyLabel =
      new JLabel(labelText,
                 new ImageIcon("images/JHUAPL.gif"),
                 JLabel.CENTER);
    fancyLabel.setBorder
       (BorderFactory.createTitledBorder("Multi-line HTML"));
    content.add(fancyLabel, BorderLayout.SOUTH);
    pack();
    setVisible(true);
  }
}
```

JPanel

In the simplest case, you use a JPanel exactly the same way as you would a Panel. Allocate it, drop components in it, then add the JPanel to some Container. However, JPanel also acts as a replacement for Canvas (there is no JCanvas). When using JPanel as a drawing area in lieu of a Canvas, there are two additional steps you usually need to

NOTES

follow. First, you should set the preferred size via setPreferredSize (recall that a Canvas' preferred size is just its current size, while a Panel and JPanel determine their preferred size from the components they contain). Secondly, you should use paintComponent for drawing, not paint

```
import java.awt.*;
import javax.swing.*;
public class JPanels extends JFrame
{
  public static void main(String[] args)
{
    new JPanels();
  }
  public JPanels()
 {
    super("Using JPanels with Borders");
     WindowUtilities.setNativeLookAndFeel();
     addWindowListener(new ExitListener());
    Container content = getContentPane();
     content.setBackground(Color.lightGray);
    JPanel controlArea = new JPanel(new GridLayout(3, 1));
    String[] colors = { "Red", "Green", "Blue",
                        "Black", "White", "Gray" };
     controlArea.add(new SixChoicePanel("Color", colors));
    String[] thicknesses = { "1", "2", "3", "4", "5", "6" };
     controlArea.add(new SixChoicePanel("Line Thickness",
                                        thicknesses));
    String[] fontSizes = { "10", "12", "14", "18", "24", "36" };
     controlArea.add(new SixChoicePanel("Font Size",
                                        fontSizes));
     content.add(controlArea, BorderLayout.EAST);
    JPanel drawingArea = new JPanel();
    // Preferred height is irrelevant, since using WEST region
    drawingArea.setPreferredSize(new Dimension(400, 0));
     drawingArea.setBorder(BorderFactory.createLineBorder
(Color.blue, 2));
     drawingArea.setBackground(Color.white);
     content.add(drawingArea, BorderLayout.WEST);
```

```
    pack();
    setVisible(true);
  }
}
```

NOTES

SixChoicePanel program

```
import java.awt.*;
import javax.swing.*;
public class SixChoicePanel extends JPanel
{
  public SixChoicePanel(String title, String[] buttonLabels)
{
    super(new GridLayout(3, 2));
    setBackground(Color.lightGray);
    setBorder(BorderFactory.createTitledBorder(title));
    ButtonGroup group = new ButtonGroup();
    JRadioButton option;
    int halfLength = buttonLabels.length/2;  // Assumes even length
    for(int i=0; i<halfLength; i++)
    {
      option = new JRadioButton(buttonLabels[i]);
      group.add(option);
      add(option);
      option = new JRadioButton(buttonLabels[i+halfLength]);
      group.add(option);
      add(option);
    }
  }
}
```

JCheckBox

Similar to Checkbox (but note the capital B in JCheckBox). You can attach either an ActionListener or an ItemListener to monitor events. If you use an ActionListener, you'll want to call is Selected to distinguish a selection from a deselection. If you use an ItemListener, the ItemEvent itself has this information: call getStateChange and compare the result to ItemEvent.SELECTED or ItemEvent.DESELECTED. Unless you are sure that the background color of the JCheckBox matches the background color of the Container, you should call setContentAreaFilled(false). You can supply an icon to replace the normal square with a check in

NOTES

it (via setIcon), but if you do, be sure to also supply an icon to be displayed when the checkbox is selected (setSelectedIcon).

JRadioButton

A JRadioButton is somewhat similar to a Checkbox when the Checkbox is inside a CheckboxGroup. Create several JRadioButtons, add them to a ButtonGroup, and also add them to a Container. Like JCheckBox, you can attach either an ActionListener or an ItemListener. However, only the radio button that is clicked will get an ActionEvent, while both the one clicked and the one that becomes deselected as a result get an ItemEvent. Unless you are sure that the background color of the JRadioButton matches the background color of the Container, you should call setContentAreaFilled(false). You can supply an icon to replace the normal square with a check in it (via setIcon), but if you do, be sure to also supply an icon to be displayed when the checkbox is selected (setSelectedIcon).

JFileChooser

This control lets users interactively select a filename by browsing directories. Normal use involves allocating a JFileChooser (pass a String for the directory to the constructor, or "." to indicate the current directory, or leave it blank for home directory), setting a title via setDialogTitle, optionally specifying a default choice (setSelectedFile), optionally writing and attaching a FileFilter to limit the file types displayed, popping it up via showOpenDialog, passing in the parent Frame (this returns an int), then finally, if the int matches JFileChooser.APPROVE_OPTION (*i.e.*, user didn't cancel), calling getSelectedFile. Note, however, that with the first two releases of Java 1.1 for Windows from Sun, the constructor call causes an error if used on a PC that has a removable drive (*e.g.*, IOMega Zip Drive) with no disk in it. Sun promises this will be fixed in JDK 1.2.2.

JTextField

The basic use of this widget works almost exactly like the AWT's TextField, including size options to the constructor, ActionEvents on ENTER and TextEvents on regular keys. However, it does not play double duty as a password field; use JPasswordField instead. You can set the text alignment via setHorizontalAlignment; supply JTextField.LEFT, JTextField.CENTER, or JTextField.RIGHT.

JTextArea

This is very similar to the AWT TextArea, but two things should be noted. First, unlike TextArea it does not directly have scrolling behaviour. Instead, like other Swing components, scrolling behavior is obtained by

wrapping it in a JScrollPane. Second, JTextArea is only for simple text, but Swing also provides JTextPane and JEditorPane, which support much more complex options.

NOTES

JSlider

In the AWT, the Scrollbar class played double duty as a control for interactively selecting numeric values and a widget used to control scrolling. This was inconvenient, and resulted in poor looking sliders. Swing gives you a "real" slider: JSlider. You create a JSlider in a similar manner to Scrollbar: the zero-argument constructor creates a horizontal slider with a range from 0 to 100 and an initial value of 50. You can also supply the orientation (via JSlider.HORIZONTAL or JSlider.VERTICAL) and the range and initial value to the constructor. You handle events by attaching a ChangeListener. Its stateChanged method normally calls getValue to look up the current JSlider value.

```
import java.awt.*;
import javax.swing.*;
public class JSliders extends JFrame {
  public static void main(String[] args) {
    new JSliders();
  }
  public JSliders() {
    super("Using JSlider");
    // Comment out next line for Java LAF
     WindowUtilities.setNativeLookAndFeel();
     addWindowListener(new ExitListener());
    Container content = getContentPane();
     content.setBackground(Color.white);
    JSlider slider1 = new JSlider();
   slider1.setBorder(BorderFactory.createTitledBorder("JSlider without
Tick Marks"));
     content.add(slider1, BorderLayout.NORTH);
   JSlider slider2 = new JSlider();
     slider2.setBorder(BorderFactory.createTitledBorder("JSlider with
Tick Marks"));
    slider2.setMajorTickSpacing(20);
    slider2.setMinorTickSpacing(5);
    slider2.setPaintTicks(true);
     content.add(slider2, BorderLayout.CENTER);
```

NOTES

```
        JSlider slider3 = new JSlider();
        slider3.setBorder(BorderFactory.createTitledBorder("JSlider with Tick
Marks & Labels"));
        slider3.setMajorTickSpacing(20);
        slider3.setMinorTickSpacing(5);
        slider3.setPaintTicks(true);
        slider3.setPaintLabels(true);
        content.add(slider3, BorderLayout.SOUTH);
      pack();
       setVisible(true);
  }
}
```

JToolBar

Swing provides a very nice new component not available in the AWT: JToolBar. In the most basic use it is little more than a JPanel, acting as a container to hold small buttons. However, the main distinction is that JToolBar is *dockable* (or *floatable*), meaning that it can be dragged out of the original window and kept as a standalone window. It can also be dragged back into the window, or dropped into the side of the window even if it was originally placed at the top. To build a JToolBar, you simply call the empty constructor (for a horizontal toolbar), or pass in JToolBar. VERTICAL. You typically place a horizontal toolbar in the north or south region of a container that uses BorderLayout, and a vertical toolbar in the east or west region. What little complexity there is with JToolBar comes from the buttons you put in it. You could just drop normal JButtons in, or call add on an Action (a special subclass of ActionListener that includes info on labels and icons), which automatically creates a JButton. The problem in both cases is that a graphical button for a toolbar differs in two ways from normal JButtons like:

1.A toolbar button should be very small, while JButton maintains relatively large margins. Solution: call setMargin with an Insets object with all values zero (or at least small).

2.A JButton puts text labels to the right of the icon, but toolbars usually have the label below. Solution: call setVerticalTextPosition(BOTTOM) and setHorizontalTextPosition(CENTER

```
import java.awt.*;
import javax.swing.*;
public class ToolBarButton extends JButton
{
  private static final Insets margins=
  new Insets(0, 0, 0, 0);
```

```
  public ToolBarButton(Icon icon) {
    super(icon);
    setMargin(margins);
     setVerticalTextPosition(BOTTOM);
    setHorizontalTextPosition(CENTER);
  }
  public ToolBarButton(String imageFile) {
    this(new ImageIcon(imageFile));
  }
  public ToolBarButton(String imageFile, String text) {
    this(new ImageIcon(imageFile));
    setText(text);
  }
}
```

NOTES

More Toolbars

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class JToolBarExample extends JFrame
                        implements ItemListener {
  private BrowserToolBar toolbar;
  private JCheckBox labelBox;
  public static void main(String[] args) {
    new JToolBarExample();
  }
  public JToolBarExample() {
    super("JToolBar Example");
     WindowUtilities.setNativeLookAndFeel();
     addWindowListener(new ExitListener());
    Container content = getContentPane();
     content.setBackground(Color.white);
    toolbar = new BrowserToolBar();
    content.add(toolbar, BorderLayout.NORTH);
    labelBox = new JCheckBox("Show Text Labels?");
     labelBox.setHorizontalAlignment(SwingConstants.CENTER);
     labelBox.addItemListener(this);
     content.add(new JTextArea(10, 30), BorderLayout.CENTER);
```

NOTES

```
        content.add(labelBox, BorderLayout.SOUTH);
        pack();
        setVisible(true);
    }
    public void itemStateChanged(ItemEvent event) {
        toolbar.setTextLabels(labelBox.isSelected());
        pack();
    }
}
```

JEditorPane

JEditorPane is sort of a fancy text area that can display text derived from different file formats. The built-in version supports HTML and RTF (Rich Text Format) only, but you can build "editor kits" to handle special purpose applications. In principle, you choose the type of document you want to display by calling setContentType and specify a custom editor kit via setEditorKit. Note that unless you extend it, legal choices are "text/html" (the default), "text/plain" (which is also what you get if you supply an unknow type), and "text/rtf".

```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
public class Browser extends JFrame implements HyperlinkListener ,
ActionListener
{
  public static void main(String[] args)
{
    if (args.length == 0)
       new Browser("http://www.apl.jhu.edu/~hall/");
    else
       new Browser(args[0]);
  }
  private JIconButton homeButton;
  private JTextField urlField;
  private JEditorPane htmlPane;
  private String initialURL;
```

```
public Browser(String initialURL)
{
  super("Simple Swing Browser");
  this.initialURL = initialURL;
   addWindowListener(new ExitListener());
   WindowUtilities.setNativeLookAndFeel();
  JPanel topPanel = new JPanel();
   topPanel.setBackground(Color.lightGray);
  homeButton = new JIconButton("home.gif");
   homeButton.addActionListener(this);
  JLabel urlLabel = new JLabel("URL:");
  urlField = new JTextField(30);
   urlField.setText(initialURL);
   urlField.addActionListener(this);
   topPanel.add(homeButton);
  topPanel.add(urlLabel);
   topPanel.add(urlField);
   getContentPane().add(topPanel, BorderLayout.NORTH);
  try {
      htmlPane = new JEditorPane(initialURL);
      htmlPane.setEditable(false);
      htmlPane.addHyperlinkListener(this);
      JScrollPane scrollPane = new JScrollPane(htmlPane);
       getContentPane().add(scrollPane, BorderLayout.CENTER);
  } catch(IOException ioe)
  {
     warnUser("Can't build HTML pane for " + initialURL
           + ": " + ioe);
  }
   Dimension screenSize = getToolkit().getScreenSize();
  int width = screenSize.width * 8 / 10;
  int height = screenSize.height * 8 / 10;
  setBounds(width/8, height/8, width, height);
  setVisible(true);
}
public void actionPerformed(ActionEvent event)
{
```

NOTES

NOTES

```
String url;
if (event.getSource() == urlField)
  url = urlField.getText();
else  // Clicked "home" button instead of entering URL
  url = initialURL;
try {
  htmlPane.setPage(new URL(url));
  urlField.setText(url);
} catch(IOException ioe) {
  warnUser("Can't follow link to " + url + ": " + ioe);
}
}
public void hyperlinkUpdate(HyperlinkEvent event)
{
if (event.getEventType() == HyperlinkEvent.EventType.ACTIVATED)
{
  try {
    htmlPane.setPage(event.getURL());
    urlField.setText(event.getURL().toExternalForm());
  } catch(IOException ioe) {
    warnUser("Can't follow link to "
          + event.getURL().toExternalForm() + ": " + ioe);
  }
}
}
private void warnUser(String message)
{
JOptionPane.showMessageDialog(this, message, "Error",
                              JOptionPane.ERROR_MESSAGE);
}
}
```

3.3 JDBC

Java Database Connectivity or JDBC for short is set of Java API's that enables the developers to create platform and database independent applications in java. The biggest advantage of programming in Java is its platform independence. An application written to access the MS Access database on Win 95/Win NT platform can work on Linux against Oracle database, only by changing the name of driver, provided none of the database calls it makes are vendor specific.

NOTES

JDBC Drivers

JDBC Drivers are set of classes that enables the Java application to communicate with databases. Java.sql that ships with JDK contains various classes for using relational databases. But these classes do not provide any implementation, only the behaviours are defined. The actual implementations are done in third-party drivers. Third-party vendors implements the java.sql.Driver interface in their database driver.

JDBC Drivers Types

Sun has defined four JDBC driver types. These are:

1. Type 1: JDBC-ODBC Bridge Driver

The first type of JDBC dirver is JDBC-ODBC Bridge which provide JDBC access to any ODBC complaint databases through ODBC drivers. Sun's JDBC-ODBC bridge is example of type 1 driver.

2. Type 2: Native-API Partly-Java Driver

Type 2 drivers are developed using native code libraries, which were originally designed for accessing the database through C/C++. Here a thin code of Java wrap around the native code and converts JDBC commands to DBMS-specific native calls.

3. Type 3: JDBC-Net Pure Java Driver

Type 3 drivers are a three-tier solutions. This type of driver communicates to a middleware component which in turn connects to database and provide database connectivity.

4. Type 4: Native-Protocol Pure Java Driver

Type 4 drivers are entirely written in Java that communicate directly with vendor's database through socket connection. Here no translation or middleware layer, are required which improves performance tremendously.

Benefits of JDBC

The JDBC API provides a set of implementation-independent generic database access methods for the above mentioned SQL-compliant databases. JDBC abstracts much of the vendor-specific details and generalizes the most common database access functions. Thus resulted a set of classes and interfaces of the java.sql package that can be used with any database providing JDBC connectivity through a vendor-specific JDBC driver in a consistent way. Thus if our application conforms to the most commonly available database features, we should be able to reuse an application with another database simply by switching to a new JDBC driver. In other words, JDBC enables us to write applications that access relational databases without any thought as to which particular database we are using.

NOTES

Also database connectivity is not just connecting to databases and executing statements. In an enterprise-level application environment, there are some important requirements to be met, such as optimizing network resources by employing connection pooling, and implementing distributed transactions. JDBC has all these features in accomplishing advanced database programming.

Steps for connecting to a MS Access Database via JDBC

First, set the DriverManager to understand ODBC data sources by using:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Then, create a connection, use the DSN say "test";

```
String dbURL = "jdbc:odbc:" + dataSourceName;
Connection con = DriverManager.getConnection(dbURL, "","");
```

Next, create a java.sql.Statement Object so you could run some queries:

```
Statement s = con.createStatement();
```

Then write the following SQl statements:

```
s.execute("create table mytest (myage number(2)");          // create a table
s.execute("insert into mytest  values(33)");                // insert some data into the table
s.execute("select myage  from mytest");          // select the data from the table
```

The next part might be a little strange—when you ran **select** query, it produced a java.sql.ResultSet. A ResultSet is a Java object that contains the resulting data from the query that was run—in this case, all the data from the column myage in the table mytest.

```
ResultSet rs = s.getResultSet();                // get any ResultSet that came from our query
if (rs != null)                                 // if rs == null, then there is no ResultSet to view
while ( rs.next() )                             // this will step through our data row-by-row
{
/* the next line will get the first column in our current row's ResultSet as a String ( getString( columnNumber) ) and output it to the screen */
System.out.println("Data from column_name: "+ rs.getString(1) );
}
```

As you can see, if the ResultSet object **rs** equals null, then we just skip by the entire **while** loop. But since we should have some data in there, we do this *while (rs.next())* bit.

Lastly, we need to close the Statement and Connection objects. This tells the database that we are done using them and that the database can free those resources up for someone else to use. **It is very important to close your connections—failure to do so can over time crash your database!** While this isn't too important with a MS-Access database, the same rules apply for any data base (like Oracle, MS SQL, etc.)

NOTES

```
s.close();                    // close the Statement to let the database know
we're done with it
con.close();                  // close the Connection to let the database know
we're done with it
```

Sample Programs

```
import java.sql.*;
public class disp
{
public static void main(String argv[])
{
try
 {
// Load the JDBC-ODBC bridge
Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
// specify the ODBC data source's URL
String url = "jdbc:odbc:mk";
// connect
Connection con = DriverManager.getConnection(url,"","");
// create and execute a SELECT
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery
("SELECT cof_name,sup_id,price,sales FROM coffees");
System.out.println("Class is SelectFromPer\n");
// traverse through results
System.out.println("Found row:");
while (rs.next()) {
System.out.print ("in loop");
// get current row values
String Surname = rs.getString(1);
int a = rs.getInt(2);
int Category = rs.getInt(3);
int c=rs.getInt(4);
// print values
System.out.print (" Surname=" + Surname);
```

NOTES

```
System.out.print (" FirstName=" + a);
System.out.print (" Category=" + Category);
System.out.print (" C=" + c);
System.out.print(" \n");
}
// close statement and connection
stmt.close();
con.close();
} catch (java.lang.Exception ex) {
}
}
}
```

Example

```
import java.sql.*;
public class Test
{
public static void main(String[] args)
    {
        try
        {
         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        String dataSourceName = "TEST";
        String dbURL = "jdbc:odbc:" + dataSourceName;
         Connection con = DriverManager.getConnection(dbURL, "","");
        Statement s = con.createStatement();
       s.execute("create table mytest(myage number(2)"); // create a table
        s.execute("insert into mytest values(31)"); // insert some data
        s.execute("select myage from mytest"); // select the data
        ResultSet rs = s.getResultSet(); // get any ResultSet
        if (rs != null) // if rs == null, then there is no ResultSet
        while ( rs.next() )
            {
          System.out.println("Data from column_name: " + rs.getString(1)
);
        }
        s.close(); // close the Statement
        con.close(); // close the Connection
    }
```

```
        catch (Exception err) {
        System.out.println("ERROR: " + err);
    }
}
}
```

NOTES

```
//save this code into a file called Test.java and compile it
```

How to Use JDBC with Swings to connect with oracle

```
import javax.swing.*
import java.awt.*;
import java.sql.*;
public class Applet1 extends JApplet
  {
  private JButton DbConnect = new JButton();
  private JTextArea empValues = new JTextArea();
  private Connection con;
  private JLabel jLabel1 = new JLabel();
  public Applet1()
 {
 }
  public void init()
   {
   try
    {
     jbInit();
    } catch(Exception e)
       {
      e.printStackTrace();
      }
 }
  private void jbInit() throws Exception
 {
    this.getContentPane().setLayout(null);
    DbConnect.setText("Click to Connect to DB");
   DbConnect.setBounds(new Rectangle(90, 225, 175, 40));
    DbConnect.addActionListener(new ActionListener()
     {
```

NOTES

```
                public void actionPerformed(ActionEvent e)
                {
                 connectToDB(e);
                }
              }
              );
             empValues.setBounds(new Rectangle(40, 50, 310, 150));
             jLabel1.setText("Emp Records :");
             jLabel1.setBounds(new Rectangle(45, 20, 155, 20));
             this.getContentPane().add(jLabel1, null);
             this.getContentPane().add(empValues, null);
             this.getContentPane().add(DbConnect, null);
           }
                 private void connectToDB(ActionEvent e)
           {
            try {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
con = DriverManager.getConnection
("jdbc:oracle:thin:@localhost:1521:ora9202","scott","tiger");
empValues.setText("Connected to the Database.
Fetching Values from DEPT Tables.\n");
              fetchValues();
             } catch (SQLException ex)
             {
               System.out.println("Connection Error =  "  + ex.toString());
             }
           }
           public void fetchValues()
           {
           try
             {
             Statement stmt = con.createStatement();
             StringBuffer allRowValues = new StringBuffer();
            int counter = 1;
           ResultSet rset = stmt.executeQuery("SELECT ENAME, EMPNO
FROM EMP");
            while (rset.next())
            {
             allRowValues.append("ROW " + counter +  ":
```

```
ENAME = " + rset.getString(1) + " &  ENO = " + rset.getString(2)
+ "\n");
            counter++;
          }
           empValues.setText(allRowValues.toString());
          rset.close();
          stmt.close();
          con.close();
        } catch (SQLException ex)
        {
            System.out.println("Error While Fetching Values = " +
ex.toString());
        }
        }
      }
```

NOTES

SUMMARY

- Main Features of Swings are:
  - Lightweight.
  - Much bigger set of built-in controls.
  - Much more customizable.
  - "Pluggable" look and feel.
  - Many miscellaneous new features.
- Swing have some components, such as:

  JApplet, JFrame, JButton, JLabel and JPanel etc.
- Java Database Connectivity or JDBC for short is set of Java API's that enables the developers to create platform and database independent applications in java.
- JDBC Drivers are set of classes that enables the Java application to communicate with databases. Java.sql that ships with JDK contains various classes for using relational databases.

REVIEW QUESTIONS

1. Write a java JApplet to print the message *Hello World* with the following characteristics

   Font-Helvetica, Size-48,

   Style-Bold, Italics Color-Cyan.
2. Modify the above program to print the same message with the

NOTES

same characteristics. The string to be displayed is taken from the corresponding html file.

3. Write a java JApplet to display an image.
4. Write a java program using swings to display a rotating line. Initially the line is pointing North, and then it points East, then South, then West and so on.
5. Create a table student data with following columns roll no., enroll no., s. name, father name, marks, now write Java program to connect this table and perform following operations:

   Add few records

   Display all records

   Search on given roll no.

   Delete records.

FURTHER READINGS

- *Design your web world,* by Sunil Jalota, Firewall Media.
- *Programming Engineering Computation in Java,* by Dr. Raja Subramaniam, Laxmi Publications (P) Ltd.

UNIT 4 JAVA BEANS AND RMI

NOTES

★ STRUCTURE ★

4.0 Learning Objectives
4.1 Introduction
4.2 Types of EJB
4.3 Applications of EJB
- *Summary*
- *Review Questions*
- *Fruther Readings*

4.0 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- define Java Beans.
- describe types of EJB.
- state application of EJB.

4.1 INTRODUCTION

EJB stands for "Enterprise JavaBeans" which are distributed network aware components for developing secure, scalable, transactional and multi-user components in a J2EE environment. Above definition actually describes EJBs from functional point of view like. what they do. A more structural definition would be: "EJBs are collection of Java classes, interfaces and XML files adhering to given rules".

What do EJBs Provide?

In J2EE all the components run inside their own containers. JSP, Servlets and JavaBeans have their own web container. Similarly EJBs run inside EJB container. The container provides certain built-in services to EJBs which the EJBs use to function. The services that EJB container provides are:

- Component pooling
- Resource management
- Transaction management
- Security

NOTES

- Persistence
- Handling of multiple clients.

Component pooling

The EJB container handles the pooling of EJB components. If there are no requests for a particular EJB then the container will probably contain zero or one instance of that component in memory. If need arises then it will increase component instances to satisfy all incoming requests. Then again if number of requests decrease, container will decrease the component instances in the pool. The best thing is that the client is absolutely unaware of this component pooling and the container handles all this for you.

Resource management.

The container is also responsible for maintaining database connection pools. It provides you a standard way of obtaining and returning database connections. The container also manages EJB environment references and references to other EJBs. The container manages following types of resources and makes them available to EJBs :

- JDBC 2.0 Data Sources
- JavaMail Sessions
- JMS Queues and Topics
- URL Resources
- Legacy Enterprise Systems via J2EE Connector Architecture.

Transaction management

This is probably the single most important factor of all. A transaction is a single unit of work, composed of one or more steps. If all the steps succeed then the transaction is committed otherwise it is rolled back. There are different types of transactions and it is absolutely unimaginable to not to use transactions in today's business environments. We will learn more about transactions in a separate article.

Security

The EJB container provides it's own authentication and authorization control, allowing only specific clients to interact with the business process. There is no need for you to create a security architecture of your own, you are provided with a built-in system, all you have to do is to use it.

Persistence

The container if desired can also maintain persistent data of our application. The container is then responsible for retrieving and saving the data for

us while taking care of concurrent access from multiple clients and not corrupting the data.

Handling of multiple clients

NOTES

The EJB container handles multiple clients of different types. A JSP based thin client can interact with EJBs with same ease as that of GUI based thick client. The container is smart enough to allow even non-Java clients like COM based applications to interact with the EJB system. Like before the EJB container handles it all for you.

4.2 TYPES OF EJB

There are three types of EJBs:

- Session Beans.
- Entity Beans.
- Message-driven Beans.

Session Beans

Session beans are the brain of our EJB based application. Their methods should decide what should or should not happen. They maintain the business logic. The client directly interacts with them, they are of two types:

- **Stateless session beans:** They do not maintain state in class level variables. They are thus fastest and most efficient of all. Since they do not save their state in class level variables, they are called as "Stateless".
- **Stateful session beans:** They do maintain state in class level variables and each client is provided a different and specific Stateful Session bean. This is different from Stateless Session beans in which case the client can be provided any Stateless Session bean between requests since the Stateless Session beans haven't managed their state, to the client, all of them are equal. Due to this fact, Stateful Session beans are heavy beans, while Stateless Session beans are light and efficient.

Entity Beans

Entity beans represent data in an EJB system. That data can reside in any data source including database. Entity beans are also of two types:

- **Container-managed persistence entity beans:** CMP bean's persistence is managed by the EJB container which is responsible for saving and retrieving the data from the underlying database

NOTES

for us. CMP beans are easier to develop and work well with database access.

- **Bean-managed persistence entity beans:** BMP bean's persistence has to be managed by the EJB itself. While it gives more control to the EJB developer to save and retrieve data, they are harder to build as more coding is required. One use of BMP beans would be to retrieve and save data from non-database data sources like XML files, JMS resources etc.

Message-driven Beans

They are new in EJB 2.0 specification and provide a mechanism for EJBs to respond to asynchronous JMS messages from different sources. Message-driven beans and JMS have opened a new paradigm for J2EE developers to create MOM (Message Oriented Middleware) based applications. We will learn more and more about this in separate articles on Stardeveloper.

4.3 APPLICATIONS OF EJB

Developers have shown that good online applications can be created using JSP, Servlets and JavaBeans alone, without using EJBs. This has raised the question that why should we use EJB? People who ask this question bring two important points in favor of not using EJBs:

- EJB Containers are expensive.
- EJB based systems turn out to more complex to develop and maintain.

I'll try to answer these questions now. First of all the reason EJB containers are expensive and their license fees turn out to be thousands of dollars for a single instance of that container is simple, the whole business process depends on that EJB container (application server) to run. Now if due to any reason there is a bug or if the application server crashes all of a sudden then it is going to halt the business application, something which business people wouldn't want to see happening even in dreams. Due to the dependency of the whole business process on these application server, the application server vendors spend lot of money on developing truly robust, secure and scalable application servers which are fault-tolerant and work in clusters to provide fail-safe operations. They also provide certain additional features like load balancing to prevent one server from becoming bobbed down from load. Current application servers also act as transactional processing monitors and work in the transactional system as heart in the body.

Installing and Running JBoss Server / Bean server

You can download JBoss from JBoss web site. Current stable version is 2.4.3. Download it from their web site. Once you have downloaded it,

unzip the JBoss zip file into some directory *e.g.,* C:\JBoss. The directory structure should be something like following:

C:\JBoss

admin

bin

client

conf

db

deploy

lib

log

tmp

This is it as far as installation of JBoss is concerned. Now to start JBoss with default configuration go to JBoss/bin directory and run the following command at the DOS prompt:

C:\JBoss\bin>run

run.bat is a batch file which starts the JBoss Server. Once JBoss Server starts, you should see huge lines of text appearing on your command prompt screen. These lines show that JBoss Server is starting. Once JBoss startup is complete you should see a message like following on your screen:

[Default] JBoss 2.4.3 Started in 0m:11s

Well done, you just installed and ran JBoss successfully on your system for the first time. To stop JBoss, simply press Ctrl + C on the command prompt and JBoss will stop, again after displaying huge lines of text.

Installing, Configuring and Running Tomcat Server

First download latest stable release of Tomcat Server from Tomcat web site. Current latest stable release is 4.0.1. If you are on Windows platform then you have the luxury of downloading Tomcat in an executable (.exe) format. So from the different types of Tomcat files available for download, select following:

jakarta-tomcat-4.0.1.exe

Once download is complete, double click this file to start the process of Tomcat installation. Installation is quite straight forward. During setup if you are running Windows NT/2K/XP, you'll be asked if you want to install it as a service, select 'yes' if you know what as a service is and how to start and stop it. Basically allowing the Tomcat to be installed a service means that you will be able to auto-start Tomcat when Windows starts and secondly Tomcat will run in the background and you won't have to keep one command prompt window open while

NOTES

it is running. Both of these features are great if you are using Tomcat on production machines. During development I will suggest that you don't install Tomcat as a service.

It will also ask you where you want it to be installed. Give it any location you want or simply allow it to be installed on the default location of C:\Program Files\Apache Tomcat 4.0.

Before starting Tomcat, let's first configure it a bit on the next page.

Configuring and Running Tomcat

Create a new folder under the main C:\ drive and name it "Projects". Now create a new sub-folder in the C:\Projects folder and name it "TomcatJBoss". The directory structure should look like following:

C:\Projects

    TomcatJBoss

Now open conf/Server.xml file from within the Tomcat directory where you have installed it. By default this location will be:

C:\Program Files\Apache Tomcat 4.0\conf\server.xml

Somewhere in the middle where you can see multiple <Context> tags, add following lines between other <Context> tags:

```
<!— Tomcat JBoss Context —>
<Context path="/jboss" docBase="C:\Projects\TomcatJBoss\" debug="0"
    reloadable="true" />
```

Now save Server.xml file. Go to Start -> Programs -> Apache Tomcat 4.0 -> Start Tomcat, to start Tomcat Server. If everything has been setup correctly, you should see following message on your command prompt.

Starting service Tomcat-Standalone

Apache Tomcat/4.0.1.

Developing your First Session EJB

As we learned in "An Introduction to Enterprise JavaBeans" article, Session EJBs are responsible for maintaining logic in our J2EE applications. What we did not say in that article was that Session beans are also the simplest EJBs to develop. So that's why our first EJB will be a Session bean.

As you will see in a moment, every EJB class file has two accompanying interfaces and one XML file. The two interfaces are Remote and Home interfaces. Remote interface is what the client gets to work with, in other words Remote interface should contain the methods you want to expose to your clients. Home interface is actually EJB builder and should contain methods used to create Remote interfaces for your EJB. By default Home interface must contain at least one create() method. The actual implementation of these interfaces, our Session EJB class file remains hidden from the clients. The XML file we talked about is named as

"ejb-jar.xml" and is used to configure the EJBs during deployment. So in essence our EJB, which we are going to create (we will call it FirstEJB from now onwards) consists of following files:

NOTES

```
com
    stardeveloper
        ejb
            session
                First.java
                FirstHome.java
                FirstEJB.java
META-INF
    ejb-jar.xml
```

First.java will be the Remote interface we talked about. FirstHome.java is our Home interface and FirstEJB.java is the actual EJB class file. The ejb-jar.xml deployment descriptor file goes in the META-INF folder.

Create a new folder under the C:\Projects folder we had created earlier, and name it as "EJB". Now create new sub-folder under C:\Projects\EJB folder and name it as "FirstEJB". Create a new folder "src" for Java source files in the 'FirstEJB' folder. The directory structure should look like following:

```
C:\Projects
    TomcatJBoss
    EJB
        FirstEJB
            src
```

Now create directory structure according to the package com. stardeveloper. ejb.session in the "src" folder. If you know how package structure is built, it shouldn't be a problem. Anyhow, the final directory structure should like following:

```
C:\Projects
    TomcatJBoss
    EJB
        FirstEJB
            src
                com
                    stardeveloper
                        ejb
                            session
```

First.java:

Now create a new First.java source file in com/stardeveloper/ejb/session folder. Copy and paste following code in it:

```
/* First.java */
package com.stardeveloper.ejb.session;
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
public interface First extends EJBObject
{
    public String getTime() throws RemoteException;
}
```

The starting two lines are import statements for importing required classes.

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
```

Then comes interface declaration line which tells that this is an interface with name "First" which extends an existing interface javax.ejb.EJBObject.

Now you have to declare methods in Remote interface which we want to be called by the client (which the client can access and call). For simplicity, we will only declare a single method, getTime(); which will return a String object containing current time.

```
public String getTime() throws RemoteException;
```

Notice that there are no () parenthesis with this method as it has been declared in an interace. Remote interface method's must also throw RemoteException because EJBs are distributed components and during the call to an EJB, due to some network problem, exceptional events can arise, so all Remote interface method's must declare that they can throw RemoteException in Remote interfaces.

FirstHome.java

Let's now create the Home interface for our FirstEJB. Home interface is used to create and get access to Remote interfaces. Create a new FirstHome.java source file in com/stardeveloper/ejb/session package. Copy and paste the following code in it:

```
/* FirstHome.java */
package com.stardeveloper.ejb.session;
import javax.ejb.EJBHome;
import javax.ejb.CreateException;
import java.rmi.RemoteException;
public interface FirstHome extends EJBHome
{
    public First create() throws CreateException, RemoteException;
```

}

Hit the 'save' button to save FirstHome.java source file.

Explanation:

First few lines are package and import statements. Next we declare our FirstHome interface which extends javax.ejb.EJBHome interface.

Note: *All Home interfaces *must* extend EJBHome interface.*

```
public interface FirstHome extends EJBHome
{
```

We continue exploring FirstHome.java on the next page.

FirstHome.java:

Then we declare a single create() method which returns an instance of First Remote interface. Notice that all methods in Home interfaces as well must also declare that they can throw RemoteException. One other exception that they *must* declare that they can throw is CreateException.

```
public First create() throws CreateException, RemoteException;
```

We are done with creating Remote and Home interfaces for our FirstEJB. These two are the only things which our client will see, the client will remain absolutely blind as far as the actual implementation class, FirstEJB is concerned.

FirstEJB.java:

FirstEJB is going to be our main EJB class. Create a new FirstEJB.java source file in com/stardeveloper/ejb/session folder. Copy and paste following text in it:

```
/* FirstEJB.java */
package com.stardeveloper.ejb.session;
import javax.ejb.SessionBean;
import javax.ejb.EJBException;
import javax.ejb.SessionContext;
import java.rmi.RemoteException;
import java.util.Date;
public class FirstEJB implements SessionBean
{
    public String getTime()
{
        return "Time is : " + new Date().toString();
    }
    public void ejbCreate() {}
    public void ejbPassivate() {}
```

NOTES

NOTES

```
    public void ejbActivate() {}
    public void ejbRemove() {}
    public void setSessionContext(SessionContext context) {}
}
```

Hit the 'save' button to save FirstEJB.java source file.

Explanation:

First few lines are package and import statements. Next we declare our FirstEJB class and make it implement javax.ejb.SessionBean interface.

Note: All Session bean implementation classes *must* implement SessionBean interface.

public class FirstEJB implements SessionBean

Our first method is getTime() which had declared in our First Remote interface. We implement that method here in our FirstEJB class. It simply returns get date and time as you can see below :

```
    public String getTime()
{
        return "Time is : " + new Date().toString();
    }
```

Then come 5 callback methods which are part of SessionBean interface and since we are implementing SessionBean interface, we have to provide empty implementations of these methods. Later in other articles when will build some really useful Session beans, we will then see some use of these callback methods there, for now we don't need them.

ejb-jar.xml:

Let's now create the EJB deployment descriptor file for our FirstEJB Session bean. Create a new ejb-jar.xml file in the FirstEJB/META-INF folder. Copy and paste following text in it:

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems,
    Inc.//DTD Enterprise JavaBeans 2.0//EN"
    "http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">
<ejb-jar>
    <description></description>
    <enterprise-beans>
      <session>
          <display-name>FirstEJB</display-name>
          <ejb-name>First</ejb-name>
          <home>com.stardeveloper.ejb.session.FirstHome</home>
          <remote>com.stardeveloper.ejb.session.First</remote>
```

```
        <ejb-class>com.stardeveloper.ejb.session.FirstEJB</ejb-class>
        <session-type>Stateless</session-type>
        <transaction-type>Container</transaction-type>
      </session>
    </enterprise-beans>
    <assembly-descriptor>
      <container-transaction>
        <method>
          <ejb-name>First</ejb-name>
          <method-name>*</method-name>
        </method>
        <trans-attribute>Supports</trans-attribute>
      </container-transaction>
      <security-role>
        <description>Users</description>
        <role-name>users</role-name>
      </security-role>
    </assembly-descriptor>
</ejb-jar>
```

NOTES

ejb-jar.xml:

One or more EJBs are packaged inside a JAR (.jar) file. There should be only one ejb-jar.xml file in an EJB JAR file. So ejb-jar.xml contains deployment description for one or more than one EJBs. Now as we learned in an earlier, there are 3 types of EJBs so ejb-jar.xml should be able to contain deployment description for all 3 types of EJBs.

Our ejb-jar.xml file for FirstEJB contains deployment description for the only EJB we have developed; FirstEJB. Since it is a Session bean, it's deployment description is contained inside <session></session> tags.

```
<ejb-jar>
    <description></description>
    <enterprise-beans>
      <session></session>
    </enterprise-beans>
</ejb-jar>
```

Now let's discuss different deployment descriptor tags inside the <session></session> tag. First is the <ejb-name> tag. The value of this tag should be name of EJB *i.e.,* any name you think should point to your Session EJB. In our case it's value is "First". Then come <home>, <remote>

and <ejb-class> tags which contain complete path to Home, Remote and EJB implementation classes. Then comes <session-type> tag whose value is either "Stateless" or "Stateful". In our case it is "Stateless" because our Session bean is stateless. For more info on Stateless and Stateful Session beans, please read "An Introduction to Enterprise JavaBeans" article. Last tag is <transaction-type>, whose value can be either "Container" or "Bean". We will learn more about transactions in another article, for now it is sufficient to say that the transactions for our FirstEJB will be managed by the container.

```
<ejb-jar>
    <description></description>
    <enterprise-beans>
      <session>
          <display-name>FirstEJB</display-name>
          <ejb-name>First</ejb-name>
           <home>com.stardeveloper.ejb.session.FirstHome</home>
           <remote>com.stardeveloper.ejb.session.First</remote>
           <ejb-class>com.stardeveloper.ejb.session.FirstEJB</ejb-class>
          <session-type>Stateless</session-type>
          <transaction-type>Container</transaction-type>
      </session>
    </enterprise-beans>
</ejb-jar>
```

Then there is a <container-transaction> tag which tells that all methods of FirstEJB "support" transactions. We will learn more about these tags and ejb-jar.xml as we continue to learn more about transactions and security in EJB environment in other articles. For now let's move forward.

Compiling the EJB Java source files

Our directory and file structure till now looks something like following:

```
C:\Projects
    TomcatJBoss
    EJB
        FirstEJB
            src
                com
                    stardeveloper
                        ejb
```

session
First.java
FirstHome.java
FirstEJB.java
META-INF
ejb-jar.xml

NOTES

You can compile all the Java source file by using a command like following on the command prompt:

```
C:\Projects\EJB\FirstEJB\src\com\stardeveloper\ejb\session>
javac-verbose-classpath %CLASSPATH%;C:\JBoss\client\jboss-j2ee.jar
    -d C:\Projects\EJB\FirstEJB *.java
```

If you have installed JBoss Server in a separate directory then substitute the path to jboss-j2ee.jar with the one present on your system. The point is to put jboss-j2ee.jar in the CLASSPATH for the javac, so that all EJB source files compile successfully.

On the next page we will learn how to package these .class files and .xml file into an easy to deploy JAR file.

Packaging EJB source files into a JAR file

Till now our directory and file structure should look something like following:

```
C:\Projects
    TomcatJBoss
    EJB
        FirstEJB
            com
                stardeveloper
                    ejb
                        session
                            First.class
                            FirstHome.class
                            FirstEJB.class
            META-INF
                ejb-jar.xml
            src
                com
                    stardeveloper
                        ejb
```

NOTES

session
First.java
FirstHome.java
FirstEJB.java

Now to package the class files and XML descriptor file together, run the following command at the command prompt:

C:\Projects\EJB\FirstEJB>jar cvfM FirstEJB.jar com META-INF

Running this command should produce an EJB JAR file with the name of FirstEJB.jar in the FirstEJB folder. But there is one thing still left to be done.

Adding JBoss specific configuration file

Till now our FirstEJB.jar file contains generic EJB files and deployment description. To run it on JBoss we will have to add one other file into the META-INF folder of this JAR file, called jboss.xml. This file contains the JNDI mapping of FirstEJB.So create a new jboss.xml file in the FirstEJB/META-INF folder where ejb-jar.xml file is present. Copy and paste the following text in it:

```
<?xml version='1.0' encoding='UTF'-8 ?>
<!DOCTYPE jboss PUBLIC "-//JBoss//DTD JBOSS//EN"
"http://www.jboss.org/j2ee/dtd/jboss.dtd">
<jboss>
 <enterprise-beans>
                    <session>
                      <ejb-name>First</ejb-name>
                      <jndi-name>ejb/First</jndi-name>
                    </session>
</enterprise-beans>
</jboss>
```

To add this new jboss.xml file into our existing FirstEJB.jar file, run the following command at the DOS prompt:

C:\Projects\EJB\FirstEJB>jar uvfM FirstEJB.jar META-INF

Now our FirstEJB.jar is ready to be deployed to the JBoss Server.

Deploying FirstEJB.jar on JBoss Server

Deploying EJBs on JBoss is as easy as copying the FirstEJB.jar file and pasting it into the C:\JBoss\deploy folder. If JBoss is running, you should see text messages appearing on the console that this EJB is being deployed and finally deployed and started. As simple as that.

Creating the client JSP page

Create new *WEB-INF folder in C:\Projects\TomcatJBoss folder.* Now create a new web.xml file and copy/paste following text in it:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2.3.dtd">
<web-app>
</web-app>
```

As you can see, this web.xml is almost empty and is not doing anything useful. We still created it because Tomcat will throw an error if you try to access this /jboss context that we had created earlier without creating a /WEB-INF/web.xml file.

firstEJB.jsp JSP Client page:

Create a new JSP page in the C:\Projects\TomcatJBoss folder and save it as "firstEJB.jsp". Now copy and paste the following code in it:

```
<%@ page import="javax.naming.InitialContext,
                 javax.naming.Context,
                 java.util.Properties,
                  com.stardeveloper.ejb.session.First,
                  com.stardeveloper.ejb.session.FirstHome"%>
<%
    long t1 = System.currentTimeMillis();
    Properties props = new Properties();
        props.put(Context.INITIAL_CONTEXT_FACTORY,
            "org.jnp.interfaces.NamingContextFactory");
        props.put(Context.PROVIDER_URL, "localhost:1099");
    Context ctx = new InitialContext(props);
    FirstHome home = (FirstHome)ctx.lookup("ejb/First");
    First bean = home.create();
    String time = bean.getTime();
    bean.remove();
    ctx.close();
    long t2 = System.currentTimeMillis();
%>
<html>
<head>
    <style>p { font-family:Verdana;font-size:12px; }</style>
```

NOTES

```
</head>
<body>
<p>Message received from bean = "<%= time %>".<br>Time taken:
   <%= (t2-t1) %> ms.</p>
</body>
</html>
```

Hit the 'save' button to save the firstEJB.jsp JSP page.

Explanation

As you can see the code to connect to an *external* JNDI/EJB Server is extremely simple. First we create a Properties object and put certain values for Context.INITIAL_CONTEXT_FACTORY and Context. PROVIDER_URL properties. The value for Context.INITIAL_ CONTEXT_FACTORY is the interface provided by JBoss and the value for Context.PROVIDER_URL is the location:port number where JBoss is running. Both of these properties are required to connect to an *external* JNDI Server.

```
Properties props = new Properties();
    props.put(Context.INITIAL_CONTEXT_FACTORY,
        "org.jnp.interfaces.NamingContextFactory");
    props.put(Context.PROVIDER_URL, "localhost:1099");
```

We then get hold of that external JNDI Context object by creating a new InitialContext() object, it's argument being the Properties object we had created earlier.

```
Context ctx=new Initial Context (props):
```

Next we use that external JNDI Context handle to lookup out FistEJB runnig on JBoss. Notice that the argument to Context. lookup ("ejb/ First") is the same value we had put in the jboss.xmi file to bind First EJB to this name in the JNDI context. We use that same value again to look for it. Once our lookup is successful, we cast it to our Fisthome Home interface.

```
FirstHome home = (FirstHome)ctx.lookup("ejb/First");
```

We then use create method of our FirstHome Home interface to get an instance of the Remote interface; First. We will now use the methods of our EJB's remote interface (First).

```
First bean = home.create();
```

We then call the getTime() method we had created in our EJB to get the current time from JBoss Server and save it in a temporary String object.

```
String time = bean.getTime();
```

Once we are done with our Session EJB, we use the remove method to tell the JBoss Server that we no longer need this bean instance. Next we also close the external JNDI Context.

```
bean.remove();
ctx.close();
```

We then display this retrieved value from getTime() method on the user screen.

NOTES

Running the JSP page

Before trying to run firstJSP.jsp page, we have to do one other thing. Copy following files from C:\JBoss\client folder and paste them in the C:\Projects\TomcatJBoss\WEB-INF\lib folder. It is a must, without it firstEJB.jsp page will not run.

connector.jar

deploy.jar

jaas.jar

jboss-client.jar

jboss-j2ee.jar

jbossmq-client.jar

jbosssx-client.jar

jndi.jar

jnp-client.jar

You will also copy the FirstEJB.jar file from C:\Projects\EJB\FirstEJB folder to the C:\Projects\TomcatJBoss\WEB-INF\lib folder. Without it the Tomcat will not be able to compile firstEJB.jsp JSP page.You are now ready to run our firstEJB.jsp page. Mover over to the last page of this tutorial, the next page.

Running firstEJB.jsp JSP page

Now start JBoss Server if it is not already running. Also start Tomcat Server. If it is already running, then stop it and restart it again. Open your browser and access following page:

http://localhost:8080/jboss/firstEJB.jsp

SUMMARY

- *EJB stands for "Enterprise JavaBeans" which are distributed network* aware components for developing secure, scalable, transactional and multi-user components in a J2EE environment.
- There are three types of EJBs:
  - Session Beans
  - Entity Beans
  - Message-driven Beans.

NOTES

- Entity beans represent data in an EJB system. That data can reside in any data source including database.
- Message-driven beans and JMS have opened a new paradigm for J2EE developers to create MOM (Message Oriented Middleware) based applications.

REVIEW QUESTIONS

1. How are Java beans helpful in designing business logics?
2. What is JBoss?
3. To design a ecommerce application Java bean are very much helpfull describe how?

FURTHER READINGS

- *'Internet and Java Programming'*, by Harish Kumar Taluja, Firewall media.
- *'Advance Java'*, by Gajendra Gupta, Firewall Media.

UNIT 5 JAVA SERVLETS

NOTES

★ STRUCTURE ★

5.0 Learning Objectives
5.1 Introduction
5.2 Servlets vs CGI
5.3 Basic Servlet Structure
5.4 Compile and Run the Servlet
- *Summary*
- *Review Questions*
- *Further Readings*

5.0 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- define Java servlets.
- state servlets vs CGI.
- discribe basic servlet structure.
- compile and run the servlet.

5.1 INTRODUCTION

Servlets are Java technology's answer to CGI programming. They are programs that run on a Web server and build Web pages. Building Web pages on the fly is useful (and commonly done) for a number of reasons:

- **The Web page is based on data submitted by the user:** For example, the results pages from search engines are generated this way, and programs that process orders for e-commerce sites do this as well.
- **The data changes frequently:** For example, a weather-report or news headlines page might build the page dynamically, perhaps returning a previously built page if it is still up to date.
- **The Web page uses information from corporate databases or other such sources:** For example, you would use this for making a Web page at an on-line store that lists current prices and number of items in stock.

NOTES

5.2 SERVLETS VS CGI

Java servlets are more efficient, easier to use, more powerful, more portable, and cheaper than traditional CGI and than many alternative CGI-like technologies. Like:

- **Efficient:** With traditional CGI, a new process is started for each HTTP request. If the CGI program does a relatively fast operation, the overhead of starting the process can dominate the execution time. With servlets, the Java Virtual Machine stays up, and each request is handled by a lightweight Java thread, not a heavyweight operating system process. Similarly, in traditional CGI, if there are *N* simultaneous request to the same CGI program, then the code for the CGI program is loaded into memory N times. With servlets, however, there are *N* threads but only a single copy of the servlet class. Servlets also have more alternatives than do regular CGI programs for optimizations such as caching previous computations, keeping database connections open, and the like.
- **Convenient:** Hey, you already know Java. Why learn Perl too? Besides the convenience of being able to use a familiar language, servlets have an extensive infrastructure for automatically parsing and decoding HTML form data, reading and setting HTTP headers, handling cookies, tracking sessions, and many other such utilities.
- **Powerful:** Java servlets let you easily do several things that are difficult or impossible with regular CGI. For one thing, servlets can talk directly to the Web server (regular CGI programs can't). This simplifies operations that need to look up images and other data stored in standard places. Servlets can also share data among each other, making useful things like database connection pools easy to implement. They can also maintain information from request to request, simplifying things like session tracking and caching of previous computations.
- **Portable:** Servlets are written in Java and follow a well-standardized API. Consequently, servlets written for, say I-Planet Enterprise Server can run virtually unchanged on Apache, Microsoft IIS, or WebStar. Servlets are supported directly or via a plugin on almost every major Web server.
- **Inexpensive:** There are a number of free or very inexpensive Web servers available that are good for "personal" use or low-volume Web sites. However, with the major exception of Apache, which is free, most commercial-quality Web servers are relatively expensive. Nevertheless, once you have a Web server, no matter the cost of that server, adding servlet support to it (if it doesn't come preconfigured to support servlets) is generally free or cheap.

5.3 BASIC SERVLET STRUCTURE

NOTES

Here's the outline of a basic servlet that handles GET requests. GET requests, for those unfamiliar with HTTP, are requests made by browsers when the user types in a URL on the address line, follows a link from a Web page, or makes an HTML form that does not specify a METHOD. Servlets can also very easily handle POST requests, which are generated when someone creates an HTML form that specifies METHOD="POST". We'll discuss that in later sections.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SomeServlet extends HttpServlet
{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException
  {
      // Use "request" to read incoming HTTP headers (e.g., cookies)
    // and HTML form data (e.g. data the user entered and submitted)
      // Use "response" to specify the HTTP response line and headers
    // (e.g., specifying the content type, setting cookies).
      PrintWriter out = response.getWriter();
    // Use "out" to send content to browser
  }
}
```

To be a servlet, a class should extend HttpServlet and override doGet or doPost (or both), depending on whether the data is being sent by GET or by POST. These methods take two arguments: an HttpServletRequest and an HttpServletResponse. The HttpServletRequest has methods that let you find out about incoming information such as FORM data, HTTP request headers, and the like. The HttpServletResponse has methods that lets you specify the HTTP response line (200, 404, etc.), response headers (Content-Type, Set-Cookie, etc.), and, most importantly, lets you obtain a PrintWriter used to send output back to the client. For simple servlets, most of the effort is spent in println statements that generate the desired page. Note that doGet and doPost throw two exceptions, so you are required to include them in the declaration. Also note that you have to import classes in java.io (for PrintWriter, etc.), javax.servlet (for HttpServlet, etc.), and javax.servlet.http (for HttpServletRequest and HttpServletResponse). Finally, note that

NOTES

doGet and doPost are called by the service method, and sometimes you may want to override service directly, e.g. for a servlet that handles both GET and POST request.

A Simple Servlet Generating Plain Text

Here is a simple servlet that just generates plain text. The following section will show the more usual case where HTML is generated.

HelloWorld.java servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet
{
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
        throws ServletException, IOException
    {
     PrintWriter out = response.getWriter();
     out.println("Hello World");
  }
}
```

5.4 COMPILE AND RUN THE SERVLET

This example can run on Java Web Server (JWS) or Tomcat or on Blazix webserver , where servlets are expected to be in a directory called servlets in the JWS installation hierarchy. Thus, HelloWorld. java actually goes in a subdirectory named servlets directory. Note that setup on most other servers is similar, and the servlet and JSP examples. With the Java Web Server, servlets are placed in the servlets directory within the main JWS installation directory, and are invoked via http://host/servlet/ ServletName. Note that the directory is servlets, plural, while the URL refers to servlet, singular. Since this example was placed in the hall package, it would be invoked via http://host/servlet/hall.HelloWorld. Other Web servers may have slightly different conventions on where to install servlets and how to invoke them. Most servers also let you define aliases for servlets, so that a servlet can be invoked via http://host/any-path/any-file.html. The process for doing this is completely server-specific; check your server's documentation for details.

Servlet th Generates HTML

Most servlets generate HTML, not plain text as in the previous example. To do that, you need two additional steps: tell the browser that you're

sending back HTML, and modify the println statements to build a legal Web page. The first step is done by setting the Content-Type response header. In general, headers can be set via the setHeader method of HttpServletResponse, but setting the content type is such a common task that there is also a special setContentType method just for this purpose. Note that you need to set response headers *before* actually returning any of the content via the PrintWriter. Here's an example:

HelloWWW.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
  public class HelloWWW extends HttpServlet
    {
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
     throws ServletException, IOException
    {
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
  out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML
4.0 " +
                                  "Transitional//EN\">\n" +
            "<HTML>\n" +
             "<HEAD><TITLE>Hello WWW</TITLE></HEAD>\n" +
            "<BODY>\n" +
            "<H1>Hello WWW</H1>\n" +
            "</BODY></HTML>");
  }
}
```

SUMMARY

- Servlets are Java technology's answer to CGI programming. They are programs that run on a Web server and build Web pages.
- Java servlets are more efficient, easier to use, more powerful, more portable, and cheaper than traditional CGI and than many alternative CGI-like technologies.
- Here's the outline of a basic servlet that handles GET requests. GET requests, for those unfamiliar with HTTP, are requests made by browsers

NOTES

when the user types in a URL on the address line, follows a link from a Web page, or makes an HTML form that does not specify a METHOD. Servlets can also very easily handle POST requests, which are generated when someone creates an HTML form that specifies METHOD="POST".

REVIEW QUESTION

1. How will you call a servlet program using JSP or HTML file, explain with an example.
2. How is servlet helpful in internet programming?

FURTHER READINGS

- *'Internet its applications with HTML and VB script'*, by Shashi Bansal, University Science Press.
- *'Design your web world'*, by Sunil Saloja, Firewall Media.