BCS-13 INTERNET & JAVA PROGRAMMING

Course Category: Department Core (DC) **Pre-requisite Subject: NIL**

Contact Hours/Week: Lecture: 3, Tutorial: 1, Practical: 2

Number of Credits: 5

Course Assessment Methods:

Continuous assessment through tutorials, attendance, home assignments, quizzes, practical work, record, viva voce and Three Minor tests and One Major Theory & Practical Examination

COURSE OUTCOMES

The students are expected to be able to demonstrate the following knowledge, skills and attitudes after completing this course

- To identify different components of client server architecture on Internet computing.
- Knowledge of how to develop and deploy applications and applets in JAVA.
- Knowledge of how to develop and deploy GUI using JAVA Swing and AWT.
- Design, develop and implement interactive web applications.
- Be able to implement, compile, test and run JAVA programs comprising more than one class and to address a particular software problem.
- To understand the basic concepts of Internet services and related technologies.
- Develop programs using the JAVA Collection API as well as the JAVA standard class library.

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.

INTERNET

- The **Internet is a network of networks that connects computers all over the world**.

- The Internet has its **roots in the U.S. military**, which funded a network in **1969,** called the **ARPANET,** to connect the computers at some of the colleges and universities where military research took place. As more and more computers connected, the ARPANET was replaced by the NSFNET, which was run by the National Science Foundation.

- By the late **1980s**, the Internet had shed its military and research heritage and was **available for use by the general public..**

CONNECTING to INTERNET: TELEPHONE, CABLE, SATELLITE CONNECTION

Your computer is connected to the Internet **if it is connected to another computer or network that is connected to the Internet.**

Several methods of connection are possible, requiring different kinds of hardware

- **Dial-Up Internet Accounts**
- **ISDN, ADSL, and Leased Line Connections**
- **Cable and DSS Internet Accounts**
- **WebTV**
- **Intranets**

CHOOSING an ISP

To connect to the Internet by using a dial-up phone line, high-speed phone line, or leased line, we first need to choose an ISP. (If we connect via cable, our cable company serves as ISP. If we use WebTV, we can use WebTV as our ISP or choose a different ISP.)

- **ISP Features**

To choose an ISP, consider the following factors:

- **Local phone number**
- **Price**
- **Software**
- **Support**
- **Speed**
- **Accessibility**

INTRODUCTION to INTERNET SERVICES

Many services are available over the Internet, and the following are the most popular ones:

- **E-mail**
- **Usenet newsgroups**
- **Online chat**
- **Voice and video conferencing**
- **The World Wide Web**
- **File transfer**

E-MAIL CONCEPTS

E-mail is a means of communication. E-mail has its drawbacks too because e-mail lacks the nuances of face-to-face or phone conversation, an e-mail message can be more easily **misunderstood than verbal communication.**

- **Receiving Incoming Messages**
- **Sending Outgoing Messages**
- **E-mail Security**
- **Reasons to Secure Messages**
- **Public Key Cryptography**

SENDING AND RECEIVING SECURE E-MAIL

contd…

Two **standard systems of public key cryptography** used on the Internet to **send and receive secure e-mail are: Digital certificates and PGP**.

- **Digital certificates**

- **PGP**

VOICE AND VIDEO CONFERENCING

A step beyond written messages is by adding voice and videoconferencing to our kit of communication tools. Three programs for this are **Microsoft NetMeeting, Netscape Conference, and CU-SeeMe.**

- **Voice conferencing** is **talking** to another person via the **microphone and speakers** connected to your computer.

- **Videoconferencing** is **sending our image and voic**e to one or more other people, through the **camera and microphone** attached to computer, and receiving pictures and voices back.

- **Need for Conferencing**

Conferencing is becoming a popular **business application**, to connect a main office with telecommuters, to **meet with customers without incurring travel costs and time**, and to keep branch offices around the world in visual contact with each other.

VOICE AND VIDEO CONFERENCING contd…

- **Limitations of video Conferencing**

- Does the person you want to talk to have a **computer and the hardware and software required** for conferencing?

- A computer isn't as portable as cellular phone; will this **lack of portability** affect our conferencing?

- Since both parties have to be using their computers at the same time, how will we **schedule our conversation?**

VOICE AND VIDEO CONFERENCING contd…

- **Equipments required for Conferencing**

➢ **Conferencing Hardware**

Speakers, microphone, and a camera.

➢ **Conferencing Software**

Microsoft NetMeeting is included with **Windows 98 and Internet Explorer 4**.

Netscape Conference is included with **Netscape Communicator**

CU-SeeMe from **White Pine Software**

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.

INTRODUCTION

Java is a **programming language** and a **platform**. Java is a **high level, robust, object-oriented and secure programming language**. Java was developed by **Sun Microsystems** (which is now the subsidiary of Oracle) in the year **1995**. **James Gosling** is known as the father of Java. Before Java, its name was **Oak**. Since Oak was already a **registered company**, so James Gosling and his team changed the Oak name to Java.

- **Platform**: Any **hardware or software environment** in which a program runs, is known as a **platform**. Since Java has a runtime environment (**JRE) and API, it is called a platform**.

INTRODUCTION

contd…

Java applications

According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some of them are

- **Desktop Applications** such as acrobat reader, media player, antivirus, etc.
- **Web Applications** such as irctc.co.in, javatpoint.com, etc.
- **Enterprise Applications** such as banking applications.
- **Mobile**
- **Embedded System**
- **Smart Card**
- **Robotics**
- **Games, etc.**

INTRODUCTION

contd…

Types of Java Applications

There are mainly four types of applications that can be created using Java programming:

- **Standalone Application**

Standalone applications are also known as **desktop applications or window-based applications.** Examples of standalone application are **Media player, antivirus, etc.**

AWT and Swing are used in Java for creating standalone applications.

- **Web Application**

An application that **runs on the server side and creates a dynamic page** is called a web application.

Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

- **Enterprise Application**

An application that is **distributed in nature**, such as **banking applications**, etc. is called enterprise application.

EJB is used for creating enterprise applications.

- **Mobile Application**

An application which is created for **mobile devices** is called a mobile application.

Android and Java ME are used for creating mobile applications.

INTRODUCTION

contd…

Java Platforms / Editions

There are 4 platforms or editions of Java:

- **Java SE (Java Standard Edition)**

It is a Java programming platform. It **includes Java programming APIs** such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core **topics** like **OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc**.

- **Java EE (Java Enterprise Edition)**

It is an enterprise platform which is mainly used to **develop web and enterprise applications**. It is built on the top of the Java SE platform. It includes **topics** like **Servlet, JSP, Web Services, EJB, JPA, etc.**

- **Java ME (Java Micro Edition)**

It is a micro platform which is mainly used to **develop mobile applications**.

- **JavaFX**

It is used to **develop rich internet applications**. It uses a light-weight user interface API.

INTRODUCTION

contd…

Features of Java

The features of Java are also known as **java buzzwords.** A list of most important features of Java language are

- **Simple**
- **Object-Oriented**
- **Portable**
- **Platform independent**
- **Secured**
- **Robust**
- **Architecture neutral**
- **Interpreted**
- **High Performance**
- **Multithreaded**
- **Distributed**
- **Dynamic**

INTRODUCTION

contd…

```
public class  a
{
      public static void main(String args[])
     {
          System.out.println("Hello Java");
      }
  }
```

To compile and run the above program, go to the current directory first c:\ Write here:

To Compile: **javac a.java**

To execute : **java a**

OPERATOR

Operator in Java is a **symbol which is used to perform operations**. For example: +, -, *, / etc.

There are many types of operators in Java which are given below:

- **Unary Operator,**
- **Arithmetic Operator**
- **Shift Operator**
- **Relational Operator**
- **Bitwise Operator**
- **Logical Operator**
- **Ternary Operator**
- **Assignment Operator**.

Data Types in Java

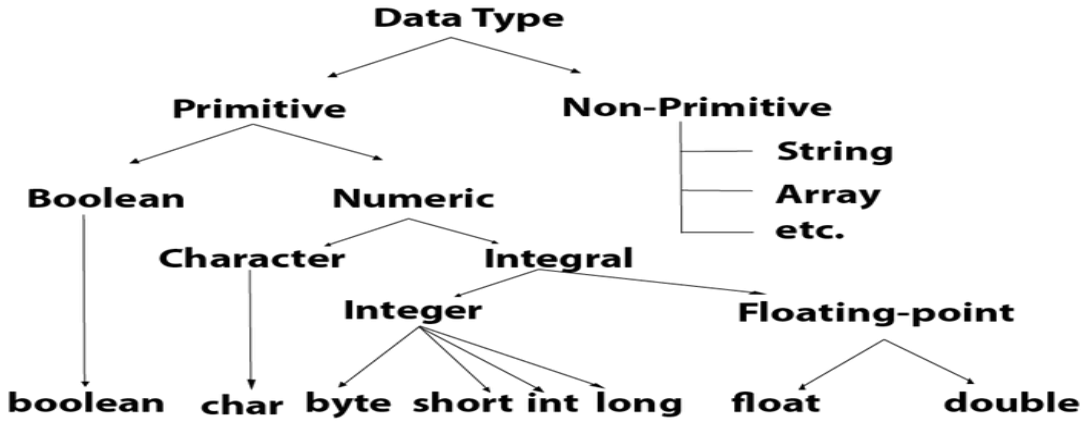
Data types specify the **different sizes and values that can be stored in the variable**.

There are **two types** of data types in Java:

- **Primitive data types:** These are the most basic data types available in Java language. There are **8 types of primitive data types.**The primitive data types include **boolean, char, byte, short, int, long, float and double.**

- **Non-primitive data types:** The non-primitive data types include **Classes, Interfaces, and Arrays.**

Data Type	Default size
boolean	1 bit
char	2 byte
byte	1 byte
short	2 byte
int	4 byte
long	8 byte
float	4 byte
double	8 byte



VAR

- **Java Variables**

A variable is a **container which holds the value** while the Java program is executed. A variable is assigned with a **data type. Variable** is a **name of memory location**.

There are **three types of variables** in java.

- **local variable**
- **instance variable**
- **static variable**

VAR

contd…

```
class a   {
            int i=5 ;          //  instance variable
            static int j=6 ;  //   static variable
              void n ()
            {
              int k= 7 ;              //  local variable
             }
         }
```

CONTROL STATEMENTS (If-Else)

The Java if statement is used to test the condition. It **checks boolean condition: true or false.**

There are various **types of if statement** in Java.

- **if statement**
- **if-else statement**
- **if-else-if ladder**
- **nested if statement**
- **Java if Statement**

CONTROL STATEMENTS(If-Else)

contd…

- The Java **if statement** tests the condition. It **executes the if block if condition is true**.

Syntax:

```
if(condition)  {
        //code to be executed
 }
```

- The Java **if-else statement** also tests the condition. It **executes the 'if block' if condition is true otherwise else block is executed.**

```
if(condition) {
        //code if condition is true
 }

else          {
        //code if condition is false
 }
```

CONTROL STATEMENTS (If-Else) contd…

if-else statement

```
public class a
    {
        public static void main(String[] args)
          {
                    int n=10;                          //defining a  variable
                    if(n%2==0)                         //Check if the number is divisible by 2 or not
                    {
                        System.out.println(“Even number");
                    }
                   else
                    {
                         System.out.println(“Odd number");
                    }
           }
     }
```

CONTROL STATEMENTS (Switch)

The Java switch statement **executes one statement from multiple conditions.** It is like **if-else-if** ladder statement.

Syntax:

```
switch(expression) {
                    case value1:
                    //code to be executed;
                     break;  //optional

                     case value2:
                      //code to be executed;
                       break;  //optional
                        ......

                       default:
                        code to be executed if all cases are not matched;
                }
```

CONTROL STATEMENTS (Switch) contd…

```
public class S
{
    public static void main(String[] args)
     {
        int n=1;                                    //Declaring a variable for switch
        switch(n)                                    //Switch expression
         {
            case 1: System.out.println("10");              //Case statements
            break;

            case 2: System.out.println("20");
            break;

           default: System.out.println("Not 10 or 20");     //Default case statement
        }
     }
}
```

LOOPS in JAVA

Loops are used to **execute a set of instructions/functions repeatedly when some conditions become true.** There are **three types of loops** in Java.

- **for loop**
- **while loop**
- **do-while loop**

- **for loop** consists of **four parts**:

➢ **Initialization:** It is the **initial condition** which is **executed once when the loop starts**.

➢ **Condition:** It is the **second condition which is executed each time to test the condition of the loop**. It continues execution **until the condition is false**. It is an optional condition.

➢ **Statement:** The statement of the loop is executed each time **until the second condition is false**.

➢ **Increment/Decrement**: It **increments or decrements the variable value**. It is an optional condition.

Syntax:

```
for(initialization;condition;incr/decr)
 {
        //statement or code to be executed
 }
```

ARRAY

An array is a **collection of similar type of elements which has contiguous memory location**.

```
class arr
    {
        public static void main(String args[])
         {
            int a[]=new int[5];                          //declaration and instantiation
            a[0]=2;                                     //initialization
            a[1]=4;
            a[2]=6;
            a[3]=7;
            a[4]=3;

           for(int i=0;i<a.length;i++)                   //traversing an array
           System.out.println(a[i]);                     //length is the property of an array

        }
    }
```

EXCEPTION HANDLING

The **Exception Handling** in Java is the mechanism to **handle the runtime errors so that normal flow of the application can be maintained e.g.** ClassNotFoundException, IOException, SQLException, RemoteException, etc.

- **Exception handling using Java try-catch block**

EXCEPTION HANDLING

contd…

```
class ex1
    {
        public static void main(String args[])
            {       int a, b;
                try
                  {   a = 0;                                          // monitor a block of code.
                      b = 100 / a;
                      System.out.println("This will not be printed.");
                   }
                catch(Exception e)
                  {     System.out.println("Exception is :"+ e);          // catch divide-by-zero error
                   }
                   System.out.println("After catch statement.");
             }
      }
```

EXCEPTION HANDLING

contd…

```
import java.util.Random;                                  // Handle an exception and move on.
class er
{    public static void main(String args[])
       {    int a=0, b=0, c=0;
            Random r = new Random();
            for(int i=0; i<10; i++)
              {         try {    b = r.nextInt();
                                 c = r.nextInt();
                                a = 5 / (b/c);
                               }
                          catch (ArithmeticException e)  {  System.out.println("Division by zero.");
                                                             a = 0; // set a to zero and continue
                                                            }       System.out.println("a: " + a);
               }
         }
 }
```

MULTITHREAD PROGRAMMING

Multithreading in Java is a process of **executing multiple threads simultaneously.** A **thread** is a **lightweight sub-process, the smallest unit of processing**. However, we use multithreading than multiprocessing because **threads use a shared memory area** so saves memory**.**

- **Life cycle of a Thread (Thread States)**

A thread can be in one of the five states.

- **New**
- **Runnable**
- **Running**
- **Non-Runnable (Blocked)**
- **Terminated**

MULTITHREAD PROGRAMMING contd…

- **Creating Thread**

There are two ways to create a thread:

- **By extending Thread class**
- **By implementing Runnable interface.**

- **By extending Thread class**

Thread class provide constructors and methods to **create and perform operations on a thread.**

- **By implementing Runnable interface.**

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. **Runnable interface have only one method named run().**

public void run(): is used to perform action for a thread.

- **Starting a thread:**

start() method of Thread class is used to **start a newly created thread.**

MULTITHREAD PROGRAMMING contd…

- **Priority of a Thread (Thread Priority):**

Each thread has a priority. **Priorities** are represented by a number **between 1 and 10**. In most cases, thread schedular schedules the threads **according to their priority** (**known as preemptive scheduling**).

- **Constants defined in Thread class:**

- **public static int MIN_PRIORITY**
- **public static int NORM_PRIORITY**
- **public static int MAX_PRIORITY**

Default priority of a thread is **5 (NORM_PRIORITY).** The value of **MIN_PRIORITY is 1** and the value of **MAX_PRIORITY is 10.**

MULTITHREAD PROGRAMMING contd…

```
class d extends Thread
  {   public void run()
        {     for(int i=1;i<=5;i++)
                {
                    System.out.println(i);
                     if(i==2)
                     try{
                            sleep(500);
                       }
                      catch(Exception e)
                        {
                         }
                  }
        }
  }
```

MULTITHREAD PROGRAMMING contd…

```
class e extends Thread
    {
        public void run()
            {
                for(int j=1;j<=5;j++)
                  {
                     System.out.println("hello");
                      if(j==3)
                      stop();
                   }
            }
    }
```

MULTITHREAD PROGRAMMING contd…

```
class ct
    {
        public static void main(String args[])
          {
            d objd=new d();
            e obje=new e();

           objd.start();
           obje.start();
         }
    }
```

MULTITHREAD PROGRAMMING contd…

```
class c implements Runnable
  {
        public void run()
         {
             System.out.println("thread is running...");
          }

         public static void main(String args[])
          {
               c objc=new c();
               Thread t1 =new Thread(objc);
                t1.start();
          }
  }
```

MULTITHREAD PROGRAMMING contd…

```
class cu
  {
        public static void main(String args[])
         {
            d objd=new d();
            e obje=new e();

            objd.setPriority(Thread.MIN_PRIORITY);
            obje.setPriority(Thread.MAX_PRIORITY);

            objd.start();
            obje.start();
         }
 }
```

I/O

Java I/O (Input and Output) is used to **process the input and produce the output** . Java uses the concept of a **stream** to make I/O operation fast. The **java.io package** contains all the **classes required for input and output operations . File handling in Java** is done by **Java I/O API.**

- **Java FileInputStream Class**

Java FileInputStream class obtains **input bytes from a file**. It is used for reading byte-oriented data (streams of raw bytes) such **as image data, audio, video** etc

- **Java FileOutputStream Class**

Java **FileOutputStream** class is an output stream used for **writing data to a file.**

- **Java Scanner**

Scanner class in Java is found in the **java.util package**. Java provides various ways to **read input from the keyboard.**

- **Java Command Line Arguments**

The java command-line argument is an argument i.e. passed at the time of running the java program.

I/O

contd…

```
import java.io.*;
public class io
{   public static void main(String args[]) throws IOException
   {  FileInputStream i = null;
      FileOutputStream o = null;
     try  {  i = new FileInputStream("a.txt");
          o = new FileOutputStream("b.txt");
          int c;
          while ((c = i.read()) != -1)
           {    o.write(c); }
           }
    finally {  if (i != null)
        {  i.close();  }
        if (o != null) { o.close();    }
       }     }
}
```

I/O

contd…

```
import java.util.*;
class uin
{  public static void main(String args[])
  {
     String   a;
     int      b;
     float    c;
     Scanner s = new Scanner(System.in);
     System.out.println("Enter a string: ");
     a = s.nextLine();
     System.out.println("Enter an integer: ");
     b = s.nextInt();
     System.out.println("Enter a float number: ");
     c = s.nextFloat();
  }
}
```

I/O

contd…

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

for(int i=0;i<args.length;i++)
System.out.println(args[i]);

}
}
```

JAVA APPLET

Applet is a special type of **program that is embedded in the webpage to generate the dynamic content**. It runs **inside the browser and works at client side**.

- **Lifecycle of Java Applet**

➢ Applet is **initialized.**

➢ Applet is **started**.

➢ Applet is **painted.**

➢ Applet is **stopped**.

➢ Applet is **destroyed**

JAVA APPLET

contd…

- **Displaying Graphics in Applet**

Commonly used methods of Graphics class:

public abstract void drawString(String str, int x, int y): is used to **draw the specified string**.

public void drawRect(int x, int y, int width, int height): draws a rectangle with the specified width and height.

public abstract void fillRect(int x, int y, int width, int height): is used **to fill rectangle** with the default color and specified width and height.

public abstract void drawOval(int x, int y, int width, int height): is used to **draw oval** with the specified width and height.

public abstract void fillOval(int x, int y, int width, int height): is used **to fill oval** with the default color and specified width and height.

JAVA APPLET

public abstract void drawLine(int x1, int y1, int x2, int y2): is used to **draw line** between the points(x1, y1) and (x2, y2).

public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer): is used **draw the specified image**.

public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle): is used **draw a circular or elliptical arc.**

public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle): is used to **fill a circular or elliptical arc.**

public abstract void setColor(Color c): is used to set the **graphics current color to the specified color.**

public abstract void setFont(Font font): is used to **set the graphics current font to the specified font.**

JAVA APPLET

contd…

```
import java.applet.Applet;
import java.awt.Graphics;
 public class ap extends Applet
{

   public void paint(Graphics g)
  {
    g.drawString("welcome",150,150);
   }
  }
 /* <applet code="ap.class" width="300" height="300"> </applet> */
```

JAVA APPLET

contd…

```
import java.applet.Applet;
import java.awt.*;
public class gr extends Applet{  public void paint(Graphics g){
g.setColor(Color.red);
g.drawString("Welcome",50, 50);
g.drawLine(20,30,20,300);
g.drawRect(70,100,30,30);
g.fillRect(170,100,30,30);
g.drawOval(70,200,30,30);
g.setColor(Color.pink);
g.fillOval(170,200,30,30);
g.drawArc(90,150,30,30,30,270);
g.fillArc(270,150,30,30,0,180);
}  }
/* <applet code="g.class" width="300" height="300"> </applet> */
```

STRING HANDLING

Java String

In Java, string is basically an **object that represents sequence of char values**.

- **Char Array**

An **array of characters** works **same as Java string**.

- **new keyword**

JVM will create a **new string object in normal** (non-pool) heap memory.

- **String Literal**

Java **String literal** is created by using **double quotes**.

- **String compare by == operator**

The = = operator **compares references** not values.

- **String Concatenation by + (string concatenation) operator**

Java string concatenation operator (+) is used to **add strings**

STRING HANDLING contd..

- **Java String class methods**

The **java.lang.String class** provides **methods to work on string**. By the help of these methods, we can perform **operations on string such as trimming, concatenating, converting, comparing, replacing strings etc.**

➢ **Java String toUpperCase() method**

➢ The java string toUpperCase() method converts the string into **uppercase letter .**

➢ **Java toLowerCase() method**

The java stringstring toLowerCase() method converts the string into **lowercase letter.**

➢ **Java String length() method**

The string length() method returns **length of the string.**

STRING HANDLING

contd..

- **Java Substring method**

Substring is a **subset of another string**.

- **Java String indexOf() method**

The **java string indexOf()** method returns **index of given character value or substring**. If it is not found, it returns -1. The index counter **starts from zero**.

- **Java String charAt() method**

The string charAt() method returns a **character at specified index**.

- **Java String replace() method**

The string replace() method **replaces all occurrence of first sequence of character with second sequence of character.**

STRING HANDLING contd..

- **Java String reverse() method**

The **reverse() method** used to replace **the character sequence by the reverse of the sequence.**

- **Java String compare by equals() method**

The String equals() method compares the **original content** of the string.

- **Java String concat method**

The **java string concat()** method **combines specified string at the end of this string**. It returns combined string

STRING HANDLING

contd…

```
class st
{   public static void main(String args[])
  {  char a[] = {'j', 'a', 'v','a' };
     String s3 = new String(a);
     System.out.println(s3+"\n");
      String s1="Java";
      String s2="programming";
      System.out.println(s1.toUpperCase());
      System.out.println(s1.toLowerCase());
      System.out.println(s1.length());
      System.out.println(s1.substring(3));
      System.out.println(s1.indexOf('j'));
      System.out.println(s1.charAt(1));
      System.out.println(s1.replace('j','k'));
```

STRING HANDLING

contd…

```
   StringBuffer s4 = new StringBuffer("cobol");
    System.out.println(s4.reverse());



     System.out.println(s1.equals(s2));
      System.out.println(s1==s2);
      System.out.println(s1+" "+s2);
      System.out.println(s1.concat(s2));
}
}
```

NETWORKING

Java Networking is a concept of **connecting two or more computing devices together so that we can share resources. Java** socket programming provides facility to share data between different computing devices.

- **IP Address**

IP address is a **unique number assigned to a node of a network** e.g. 192.168.0.1 . It is composed of **octets that range from 0 to 255**.It is a **logical address** that can be changed.

- **java.net package**

The **java.net package** provides **classes to deal with networking applications** in Java.

e.g. **InetAddress class**

NETWORKING

contd…

- **Java InetAddress class**

Java InetAddress class represents an IP address. An IP address is represented by 32-bit (IPV4)or 128-bit (IPV6) unsigned number.

➢ **Commonly used methods of InetAddress class are**

Method	**Description**
public static InetAddress **getByName(String host)** throws UnknownHostException	It returns the instance of InetAddress containing **LocalHost IP and name.**
public static InetAddress **getLocalHost()** throws UnknownHostException	It returns the instance of InetAdddress containing **local host name and address.**
public String **getHostName()**	it returns the **host name of the IP address**.
public String **getHostAddress()**	it returns the **IP address in string format.**

NETWORKING

contd…

```
import java.net.*;
class nw
{    public static void main(String args[]) throws UnknownHostException
     {  InetAddress ad = InetAddress.getLocalHost();
        System.out.println(ad);
        ad = InetAddress.getByName("www.yahoo.com");
        System.out.println(ad);
        InetAddress a[] = InetAddress.getAllByName("www.yahoo.com");
        for (int i=0; i<a.length; i++)
        System.out.println(a[i]);
      }
}
```

EVENT HANDLING

Changing the state of an object is known as an **event.** For example, **click on button, dragging mouse etc.** The **java.awt.event package** provides many **event classes and Listener interfaces** for event handling.

EVENT CLASSES	LISTENER INTERFACES
ActionEvent	**ActionListener**
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

EVENT HANDLING

contd…

```
import java.awt.*;
import java.awt.event.*;
public class evh
{   public static void main(String args[])
 {   Frame f=new Frame();
   TextField t=new TextField();
   t.setBounds(50,50, 150,20);
   Button b=new Button("click me");
   b.setBounds(50,100,60,30);
   b.addActionListener ( new ActionListener()
                        { public void actionPerformed(ActionEvent e)
                           { t.setText("Welcome to Java");
                           }
                        }
                    ) ;
   f.add(b);    f.add(t);     f.setSize(300,300);        f.setLayout(null);      f.setVisible(true);
    }
}
```

INTRODUCTION TO AWT

Java AWT(Abstract Window Toolkit)

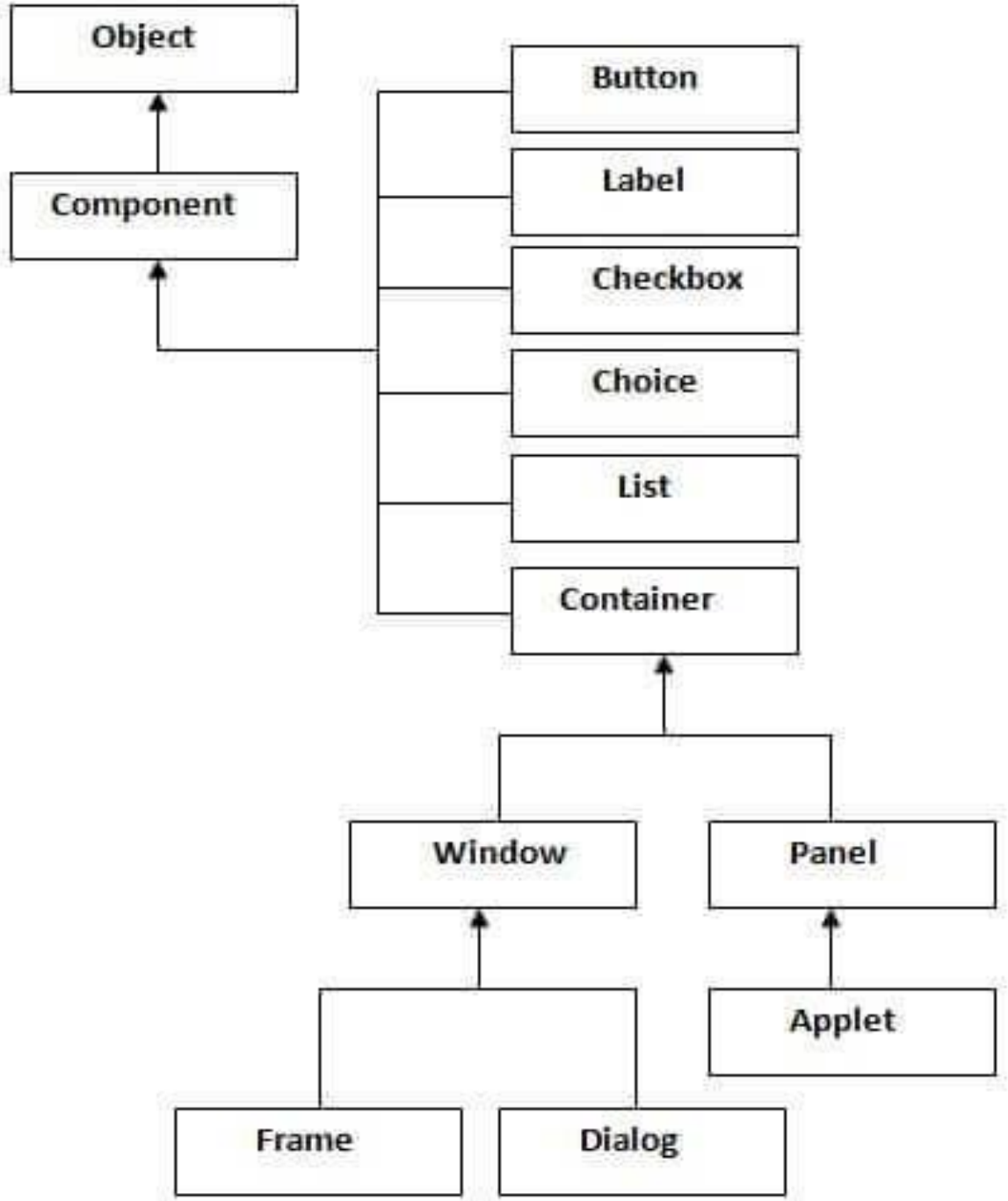
It is an **API to develop GUI or window-based applications in Java**. The **java.awt package** provides **classes for AWT API** such as **TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.**

INTRODUCTION TO AWT

contd…

- **Java AWT Hierarchy**

The hierarchy of Java AWT classes are as



INTRODUCTION TO AWT

contd…

- **Container**

The Container is a component in AWT that can **contain another components like buttons, textfields, labels etc.**

- **Window**

The window is the container that have **no borders and menu bars**. You must use frame, dialog or another window for creating a window.

- **Panel**

The Panel is the container that **doesn't contain title bar and menu bars**. It can have other components like button, textfield etc

- **Frame**

The Frame is the container that contain **title bar** and can have **menu bars**. It can have other components like button, textfield etc.

AWT CONTROLS

- **Frame**
- **Label**
- **Button**
- **TextField**
- **TextArea**
- **Checkbox**
- **CheckboxGroup**

➢ **Frame**

The Frame is the container that contain **title bar and can have menu bars**. It can have other components like button, textfield etc.

Frame f=new Frame ( ) ;

AWT CONTROLS

contd…

- **Label**

The object of Label class is a **component for placing text in a container.** It is used to display a single line of read only text.

```
Label l=new Label("First Label.");
```

- **Button**

The button class is used to **create a labeled button** that h.as platform independent implementation.

```
Button b=new Button("click me");
```

- **TextField**

The object of a TextField class is a text component that allows the **editing of a single line text.**

```
TextField t =new TextField("Enter your name:");
```

AWT CONTROLS

contd…

- **TextArea**

The object of a TextArea class is a multi line region that displays text. It allows the **editing of multiple line text.**

TextArea a =new TextArea("Enter your name:");

- **Checkbox**

The Checkbox class is used to create a checkbox. It is used to **turn an option on (true) or off (false)**

Checkbox c =new Checkbox("Java");

- **CheckboxGroup**

The object of CheckboxGroup class is used to **group together a set of Checkbox**. **At a time only one check box button is allowed to be in "on" state** and remaining check box button in "off" state.

CheckboxGroup c1= new CheckboxGroup();

AWT CONTROLS

contd…

```
import java.awt.*;
class fr
 {
    fr()
   {  Frame f=new Frame();
    f.setSize(300,300);
    f.setLayout(null);
    f.setVisible(true);
    }
  public static void main(String args[])
   {
    fr f=new fr();
   }
 }
```

AWT CONTROLS

contd…

```
import java.awt.*;
class la
{    la()
  {    Frame f=new Frame();
   Label l=new Label("First label");
   l.setBounds(30,100,80,30);
   f.add(l);
   f.setSize(300,300);
   f.setVisible(true);
   }
 public static void main(String args[])
  {     la f=new la();
  }
}
```

AWT CONTROLS

contd…

```
import java.awt.*;
class bu
{
   bu()
  {  Frame f=new Frame();
   Button b=new Button("click me");
   b.setBounds(30,100,80,30);            // setting button position
   f.add(b);                             //adding button into frame
   f.setSize(300,300);                   //frame size 300 width and 300 height
   f.setLayout(null);                    //no layout manager
   f.setVisible(true);                   //now frame will be visible, by default not visible
   }
 public static void main(String args[])
  {
   bu f =new bu();
  }
}
```

AWT CONTROLS

contd…

```
import java.awt.*;
class tf
{    tf()
  {  Frame f=new Frame();
   TextField t =new TextField("Enter your name:");
   t.setBounds(50,100,200,30);
   f.add(t);
   f.setSize(300,300);
   f.setLayout(null);
   f.setVisible(true);
   }
 public static void main(String args[])
  {     tf f=new tf();
  }}
```

AWT CONTROLS

contd…

```
import java.awt.*;
class ta
{    ta()
  {  Frame f=new Frame();
   TextArea a =new TextArea("Enter your name:");
   a.setBounds(10,30, 300,300);
   f.add(a);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
   }
 public static void main(String args[])
  {     ta f =new ta();
  }}
```

AWT CONTROLS

contd…

```
import java.awt.*;
class cb
{   cb()
  {  Frame f=new Frame();
   Checkbox c =new Checkbox("Java");
  c.setBounds(10,30, 300,300);
   f.add(c);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
   }
 public static void main(String args[])
  {    cb f =new cb();
  }}
```

AWT CONTROLS

contd…

```
import java.awt.*;
class cbgp
{   cbgp()
  {    Frame f=new Frame();
   CheckboxGroup c1= new CheckboxGroup();
    Checkbox c2 =new Checkbox("Java",c1,true);
    c2.setBounds(10,30, 300,300);
   f.add(c2);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
   }
 public static void main(String args[])
  {  cbgp f =new cbgp();
  }}
```

LAYOUT MANAGERS

The **LayoutManagers** are used to **arrange components in a particular manner**.

- There are following **classes that represents the layout managers**:

➢ java.awt.BorderLayout

➢ java.awt.FlowLayout

➢ java.awt.GridLayout

➢ java.awt.CardLayout

➢ java.awt.GridBagLayout

➢ javax.swing.BoxLayout

➢ javax.swing.GroupLayout

➢ javax.swing.ScrollPaneLayout

➢ javax.swing.SpringLayout etc.

LAYOUT MANAGERS

contd…

- **FlowLayout**

The FlowLayout is used to **arrange the components in a line, one after another (in a flow).** It is the **default layout of applet or panel**.

f.setLayout(new FlowLayout(FlowLayout.RIGHT));

- **GridLayout**

The GridLayout is used to **arrange the components in rectangular grid.** One component is displayed in each rectangle.

f.setLayout(new GridLayout(2,2));

- **BorderLayout**

The BorderLayout is used to **arrange the components in five regions: north, south, east, west and center**. Each region (area) may contain one component only. It is the **default layout of frame or window.**

f.add(b1,BorderLayout.NORTH);

LAYOUT MANAGERS

contd…

- **CardLayout**

The CardLayout class manages the components in such a manner that **only one component is visible at a time.**

card=new CardLayout(40,30);

- **GridBagLayout**

The Java GridBagLayout class is used to **align components vertically, horizontally** or along their baseline.

GridBagLayout layout = new GridBagLayout();

- **BoxLayout**

The BoxLayout is used to **arrange the components either vertically or horizontally.**

setLayout (new BoxLayout(this, BoxLayout.X_AXIS));

LAYOUT MANAGERS

contd…

```
import java.awt.*;
class fl
{   fl()
   {  Frame f=new Frame();
    Button b1=new Button("1");
    Button b2=new Button("2");
    Button b3=new Button("3");
    f.add(b1);
    f.add(b2);
    f.add(b3);
    f.setLayout(new FlowLayout(FlowLayout.RIGHT));       //setting flow layout of right alignment
    f.setSize(300,300);
    f.setVisible(true);     }
 public static void main(String[] args)
  {     fl f= new fl();    }  }
```

Thank You