JAVA PROGRAMMING

CONTENT

Unit I

INTRODUCTION

Java:

Java was conceived by James Gosling and his team at Sun Micro Systems, Inc in 1991. This language was initially called “Oak” and in 1995 it was renamed as “Java”. Java supports Application programs and Applet programs. An Application program is one that can be run on your computer with its operating system. An Applet is an application designed to be transmitted over the internet and executed by a Java-compatible web browser.

Bytecode:

Java needs a compiler and an interpreter. The java source program is passed to a compiler and the output of the java compiler is a bytecode. The bytecode is a highly optimized set of instructions designed to be executed by the java run-time system, which is the Java Virtual Machine (JVM). The bytecode can be executed by any interpreter. It is machine independent and architecture neutral ie byte can be executed by IBM interpreter, SUN interpreter, SPARC interpreter …etc.

Object Oriented Programming (OOP):

Java is a fully object oriented language. All java programs must have class. The three OOP principles are Encapsulation, Inheritance and Polymorphism.

- Encapsulation: It is the process that binds together code and the data it manipulates. The basis of encapsulation is the class. In a class the user will specify the code and the data. An object is the instance of the class.
- Inheritance: It is the process by which one object acquires the properties of another object. The user can create a new class from the existing class. It supports the reusability of code.
- Polymorphism: It is a feature that allows one interface to be used for a general class of actions.

DATA TYPES, VARIABLES, ARRAYS and OPERATORS

1.1 Java Data Types

Variable declared in java has a data type. Data types specify the type of data and size needed to store the data.

1.1.1 Integers: There are four types of integers namely byte, short, int and long. The size of these four integer types are long – 8 bytes, int – 4 bytes, short – 2 bytes and byte – one byte.

1.1.2 Floating Point types: Java supports two types of floating point data types namely float and double. The float type data uses 4 bytes of storage. Float type data specifies a single precision value. The double type data uses 8 bytes of storage. Double type data specifies double precision value.

1.1.3 Characters: char is the data type used to store characters in java. The size of character type data is 2 bytes.

1.1.4 Boolean: Boolean type data holds either true or false. The keyword boolean is used to declare the Boolean type data.

1.2 Literals:

Literals are constants in java. The value of a literal remains unchanged during program execution. Java supports Integer literals, Floating-Point literals, Boolean literals, Character literals and String literals.

1.2.1 Integer Literals: An integer literal is any whole number (eg: 34, 23, 5…..). There are three types of integer literals namely Decimal literal (base 10), Octal literal (base 8) and Hexadecimal literal (base 16).

1.2.2 Decimal Literal : A decimal literal is the commonly used one , which is a valid decimal integer number. Long integer literals are appended with I or L.

Example: 123 -567 23456 873456***l*** or 873456L(long int literal)

1.2.3 Octal Literal: Octal literals are any integer octal number with leading ***Zero.***

Example: 0356 -0652 054

1.2.4 Hexadecimal Literal: Hexadecimal literals are any hexadecimal number with leading **Zerox (0x or 0X).**

Example: 0x23 0X234 0xA13 0XB6F

1.2.5 Floating-Point Literals: There are two types of floating point literals. They are float literal or double literal: Any floating point number appended with **f** or ***F*** is float literal. A double literal is any floating point number appended with d or D. Double literal is the java‟s default floating point literal.

Example Float literal: 3.14f -8.9F 6.785F -4.53f

Double literal: 23.8976D 345.98765d -56.245D -7.8643d

1.2.6 Boolean Literals: Boolean literals can take any boolean value true or false.

Example: true false

1.2.7 Character Literals: A character literal is a character enclosed within a single quote.

Example: „2‟ „A‟ „\n‟ „c‟

1.2.8 String Literals: A siring literal is a group of characters or a character enclosed within double quotes.

Example: “123” “A” “CVD” “7” “nhg\nbc”

1.3 Variables:

Variable is a quantity whose value can be changed program execution. It is an identifier that denotes a storage location used to store a data value. A variable name may consists of alphabets, digits or underscore.

1.3.1 Variable Declaration: The general form of variable declaration is given below.

type variable-name1, variable-name2 ….nariable-namen;

Example:

```
int x,y,z;
float a,b;
double d1,d2;
char c1,c2;
boolean b1,b2;
```

1.3.2 Initialization:

The initialization can be done in either compile time or at execution(dynamically). Dynamic initialization can be done through any valid expression at the time of variable declaration. The general form is given below.

type variable = value;

type variable = expression;

Example:

```
int max=400;
int min=200;
double pi=3.14;
boolean result = true;
int c=max – min;
```

1.4 Type Conversion and Casting:

In java, when one type of data is assigned to another type of variable, an automatic type conversion takes place if the following two conditions are satisfied.

- The destination memory is larger than the source memory.
- The two types are compatible

1.4.1 Casting:

A cast is an explicit type conversion. The general form is given below.

- Variable = (target-type) value;

Example:

- int x;

```
byte y;
y=(byte)x; // At this point only x is converted to byte.
```

1.5 Automatic Type Promotion in Expressions:

In an expression, if the precision required for a intermediate value exceeds the range of the either operand, Java automatically promotes the operand to next higher type. For example the byte or short type operand can be converted to int when evaluating the expression.

Example:

```
byte x,y;
int ,z;
x=100;
```

y=75;

z=x * y + 25;

In the above example the intermediate term x * y exceeds the range of either of its byte operands. Here Java automatically converts byte to short or int.

1.5.1 Type Promotion Rules:

In java, type promotion rule says that all byte and short values are promoted to int. If one of the operand is a long, the whole expression is promoted to long. If one operand is a float, the entire expression is promoted to float. If any one of the operand is double, the result is in double.

1.6 One Dimensional Arrays:

An array is a group of similar data typed variables that shares a common name. The elements of the array are accessed by the array index. The array index must begin in zero to n-1, where n is the number of elements in the array. The array index should be a positive integer vale. An array may be one-dimensional or multidimensional it may be of any type ie int, float, double, char, String, objects …..

1.6.1 Declaration:

A one-dimensional array is a list of similar typed variables. It can be accessed with one array index. The one-dimensional array can be declared as follows.

type array-name[ ] = new type[size];

Example:

- int a[ ] = new int[10];
- double n[ ] = new double[5];
- String s[ ] = new String[7];

In the above example ***a*** is an integer array of size 10, ***n*** is a double type array of size 5 and ***s*** is a String array of size 7.

1.6.2 Array Input:

The array elements can be assigned directly along with the declaration. This is called initialization of arrays. The values can also be assigned during execution by the user.

Example:

int a[ ] = { 10, 34,22,56,21};

In the above example the integer values are assigned as a[0]=10, a[1]=34, a[2]=22, a[3]=56 and a[4]=21.

String s[ ] = { "Anand", "Bala", "Jhon Samuel", "Hayes"};

In the above example the String objects are assigned as s[0]="Anand"; s[1]="Bala", s[2]="Jhon Samuel", s[3]="Hayes"

1.6.3 Array Output:

The elements of the array can be accessed with the array index and a loop statement.

Example:

```
int b[ ]= { 34,23, 11, 67,23};
System.out.println(" Array Elements");
for(i=0; i<5; i++)
System.out.println(b[i]);
```

1.6.4 Array Processing:

The following example illustrates the array processing.

```
int b[ ]= { 34,23, 11, 67,23};
int s=0;
for(i=0; i<5; i++)
s=s+b[i];
System.out.println("Sum of the elements = " +s);
```

Example: Java program to arrange the given set of numbers in ascending order.

```
import java.io.*;
class Sort {
int a[ ]={3,5,1,-8,2};
int i,j,t;
void sorting( ) {
for(i=0;i<5;i++)
for(j=i+1;j<5;j++)
if(a[i]>a[j]) {
t=a[i];
```

```
a[i]=a[j];
a[j]=t;
}
}
void output( ) {
for(i=0;i<5;i++)
System.out.println(a[i]);
}
}
class Arraysort  {
public static void main(String arg[ ]) {
Sort s1=new Sort( );
System.out.println("Before Sorting");
s1.output( );
s1.sorting( );
System.out.println("After Sorting");
s1.output( );
}
}
```

1.7 Multi-Dimensional Arrays:

1.7.1 Declaration:

A multidimensional array is a list of array of arrays. It may be a two-dimensional array or three-dimensional array. The example for two-dimensional array is a matrix. A two-dimensional array can be accessed by its row index and its column index. The general form and example for two-dimensional array are as follows.

```
type array-name[ ][ ] = new type[ rowsize][columnsize];
```

Example:

- `int m[ ][ ] = new int[3 ] [4];`
- `double d[ ][ ] = new double[2] [3];`

- String name[ ][ ]= new String[3][20];

In the above example ***m*** is an integer type matrix of 3 rows and 4 columns, ***d*** is a double type matrix of 2 rows and 3 columns and ***name*** is a String array of size 3 rows and 20 columns.

1.7.2 Input:

The matrix elements can be assigned directly along with the declaration. This is called initialization of arrays. The values can also be assigned during execution by the user.

Example:

int a[ ][ ]= { {1,2,3},{4,5,6},{7,8,9}};

In the above example the integer values are assigned as a[0][0]=1, a[0][1]=2, a[0][2]=3, a[1][2]=4 ….a[2][2]=9.

1.7.3 Output:

The elements of the matrix can be printed in the matrix format array index and two loop statements.

Example:

```
int a[ ][ ]= { {1,2,3},{4,5,6},{7,8,9}};
System.out.println(“ Matrix);
for(i=0; i<3; i++)
{
for(j=0;j<3;j++)
System.out.print(a[i][j]+”    “]);
System.out.println(“ “);
}
```

1.7.4 Processing:

The following example illustrates the matrix processing.

```
int a[ ][ ]= { {1,2,3},{4,5,6},{7,8,9}};
int b[ ][ ]= { {11,22,33},{44,55,66},{77,88,99}};
int c[ ][ ] = new int[3][3];
```

```
for(i=0; i<3; i++)
for(j=0;j<3;j++)
c[i][j]=a[i][j]+b[i][j];
```

Example: Java program to add two matrices.

```
import java.io.*;
class Retobj {
int r,c,i,j;
int a[ ][ ]=new int[4][4];
Retobj(int r1,int c1)  {
r=r1;
  c=c1;
}
void input( ) throws IOException {
DataInputStream x=new DataInputStream(System.in);
System.out.println("Enter Matrix elements");
for(i=0;i<r;i++)
for(j=0;j<c;j++)
a[i][j]=Integer.parseInt(x.readLine());
}
Retobj sum(Retobj b1) {
Retobj t=new Retobj(3,3);
```

```
for(i=0;i<r;i++)
for(j=0;j<c;j++)
t.a[i][j]=a[i][j]+b1.a[i][j];
return(t);
}
void output( ) {
for(i=0;i<r;i++) {
for(j=0;j<c;j++)
System.out.print(a[i][j]+" ");
System.out.println("  ");
}
}
}
class Mataddretobj {
public static void main(String arg[ ]) throws IOException {
Retobj m1=new Retobj(3,3);
Retobj m2=new Retobj(3,3);
Retobj m3=new Retobj(3,3);
m1.input( ) ;
m2.input(          );
m3=m1.sum(m2);
```

```
m1.output();

m2.output();

m3.output();

}

}
```

1.8 Operators:

1.8.1 Arithmetic Operators:

Arithmetic operators can be used to perform arithmetic operations by writing arithmetic expressions. The modulus operator returns the remainder of a division operation. In java modulo division operator % can be used for both integer and floating point arithmetic operations. Plus (+) operator can be used for addition as well as Concatenation of strings.

The arithmetic operators are given below.

Operator	***Meaning***
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo division
++	Increment
_ _	Decrement
+=	Addition assignment
-=	Subtraction assignment
*=	Multiplication assignment

/=	Division assignment
%=	Modulo division assignment

Example1: Modulo Division

```
double x,y,z;

x=15.3;

y=2.0;

z=x % y;
```

Then z=1.3;// Returns the remainder of 15.3 % 3.0

Example2: Prefix Increment

```
int n,p;

n=5;

p=++n;
```

Now , p=6 and n=6

Example3: Postfix Increment

```
int n,p;

n=5;

p=n++;
```

Now , p=5 and n=6

Example2 and Example3 shows the difference between prefix increment and postfix increment operator.

Example4: Addition assignment

```
int x,y;

x=5;

x+=23;

 ie x=x + 23;// x=28.
```

Similarly other shorthand arithmetic operators can be used.

1.8.2 Bitwise operators:

Java supports the following bitwise operators. They are used to perform bitwise operations.

Operator	Meaning
|	Bitwise OR
&	Bitwise AND
~	Bitwise NOT
^	Bitwise XOR
>>	Right Shift
<<	Left Shift
>>>	Right Shift with zero fill
>>=	Right Shift assignment
<<=	Left Shift assignment
>>>=	Right Shift zero fill assignment
^=	Bitwise XOR assignment
|=	Bitwise OR assignment
&=	Bitwise AND assignment

Example: Bitwise AND

```
a = 1010; // Binary number
b = 1101;// Binary number
 c= a & b;
```

c= (1010) & (1101);

c= 1000;

Bitwise OR

d=a | b;

d=(1010) | (1101);

d = 1111;

Right Shift

x=b >> 2;// Perform two times right shift operation to b and assign to x.

Now, x= 0011;// Binary

Similarly other bitwise operations can be performed.

1.8.3 Relational Operators:

Relational operators are used to write the relational expressions. The result of the operations is a boolean value. They are mainly used in control statements.

The relational operators are given below.

Operator	Meaning
==	Equality
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Example:

```
int a,b;

a=10;

b=20;
```

```
if(a>b)

System.out.println("A is Bigger than B");

else

System.out.println("B is Bigger than A");
```

1.8.4 Boolean Logical Operators:

Boolean logical operators can operate on boolean operands.

Operator	Meaning
|	Logical OR
&	Logical AND
^	Logical XOR
&&	Short circuit AND
||	Short circuit OR
!	Logical NOT
|=	OR assignment
&=	AND assignment
^=	XOR assignment
==	Equality
!=	Not equal to
?:	Conditional (Ternary Operator)

Example1:

```
boolean x=false;
boolean y=true;
boolean z=true;
boolean d;
d=x|y;
  System.out.println("Boolean OR:" + d);// true
```

```
d=x & z;
System.out.println("Boolean AND:" + d);// false
```

Example2: Short circuit AND – Used to combine two relational expressions

```
int m1,m2;
m1=50;
m2=67;
if((m1>=40) &&(m2(>=40)) // short circuit AND
System.out.println("Pass");
else
System.out.println("Fail" );
```

1.8.5 Assignment Operator:

Used to assign a value or to assign an expression to a variable.

Variable = value;

Variable = Expression;

Example:

```
int x,y,s;
x=10;
y=20;
s=x+y;
```

1.8.6 Conditional Operator (?:)

Conditional operator is a ternary operator. It takes three operands. It works similar to if-then-else statement.

Variable=Condition **?** Expression1 **:** Expression2;

Example:

```
int a,b,x;
a=25;
b=34;
```

x=(a>b)?a:b;

In the above example, if the condition is true then the value of **a** is assigned to **x,** otherwise (else) the value of **b** is assigned to **x**. Here the condition is false, so the value of **b** is assigned to **x**.

1.9 Operator Precedence:

Operators in java have its own priority. It is essential to understand the priority to write the correct expressions. Parentheses can change the priority of the operators inside them. Precedence of the java operators are mentioned below.

Highest Priority

( )	[ ]		
++	- -		
*	/	%	
+	-		
>>	>>>	<<	
>	>=	<	<=
= =	!=		
&			
^			
|			
&&			
||			
? :			
=	operator=		

Lowest Priority

INTRODUCING CLASSES

1.10 Class Fundamentals:

Java is a fully object oriented programming language. It means that any program written using java must be encapsulated within a class. Class defines a new data type. Once it is created, that can be used to create objects. The class is a template for an object. The object is an instance of a class. A class is declared using the keyword class.

Generally a class consists of instance variables and methods. The variables declared within a class are called instance variable. The methods contain codes to access the instance variables of the class. The instance variables and methods declared within a class are called members of the class.

The general form of a class definition is given below.

```
class Class-name
{
    type instance-variable1;
     type instance-variable2;
     type instance-variable3;
     type instance-variable -n;
      type method-name1(argument-list)
      {
   // Body of method1
         }

              ----------

    }
```

Example:

```
class Student

    {

       int  rollnum;
```

```
double height, weight;
 String name;
   void input( )   {
     rollnum=2001;
     height=172.5;
     weight=76.8;
     name=”Arun”;
     }
void output( )   {
    System.out.println(“Roll Number = “+rollnum);
     System.out.println(“Name = “+name);
     System.out.println(“Height = “+ht);
     System.out.println(“Weight = “+wt);
       }
  }
```

1.11 Declaring Objects: Objects are the instance of a class or run time entity of a class. The members of a class are accessed by the object and the dot(**.**) operator. Object of a class can be created in two steps. First create a reference variable, then allocate memory to the object using ***new*** operator. The general form is given below.

```
class-name object-name; // creates a reference to object
object-name = new class-name( ); //allocate memory to object
```

(or)

class-name object-name = new class-name( );

Example1

Student s1; // S1 is a reference to object S1=new

Student( ); // allocate memory to object

Example2:

Student S1=new Student( );

S1 is the object of the class type Student. Similarly we can create any number of objects.

1.12 Assigning Object Reference Variables.

Object reference variables can act differently when an object is assigned. In the following example s1 is the object of the class Student and s2 is a reference variable of the class Student. There is no separate memory for it. When an object is assigned to a reference variable, s2 did not allocate any memory or copy of the original object s1.

Student s1 = new Student( );

Student s2=s1;

1.13 Introducing Methods.

In java class consists of instance variables and methods. Methods consist of set of statements to access the instance variables and to perform a particular task. The general form of a method is given below.

type method-name(arguments)

{ // Body of method

return *value*;

}

Type specifies the type of the data returned by the method to the calling method. The type may be int, double, float, char, boolean, String or class-name(for object) and void for return nothing. The return statement can be used to return a value to the calling method.

Example:

```
class Demo {
int x, y, s;
void input(int x1, int y1)    {
  x=x1;
  y=y1;
  }
int  output( ) {
 sx+y;
System.out.println("x= "+x);
System.out.println("y= "+y);
return s;
}
}
```

Explanation:

In the above program the Demo class consists of the instance variable x, y ,s and the methods input( ) and output( ). The input( ) takes two arguments of type int. The output( ) has a return type int. The return statement in the output( ) returns the value s to the calling method.

1.14 Constructor:

A Constructor is similar to a method and it can be used to initialize the members of the class. It initializes the members immediately upon creation. A Constructor is having the same name as class name. It has no return type even void. Constructors are executed automatically immediately after the object is created. Constructors without argument are called default constructor. The default constructor automatically initializes all instance variables to zero. Constructor with argument are called parameterized constructor. Constructors can be overloaded like other methods. The general form of a constructor is given below.

```
class Class-name
 {
   // Instance variables;
    Class-name( )  //Constructor
         {
             ----------
            // inizialization
             ----------------
          }
type methodname( )
     {
            // Body of the method;
      }
  }
```

Example:

```
class Demo {
int x, y, s;
Demo( )  {
  x=25;
  y=35;
  s=0;
  }
void output( )  {
 s=x+y;
System.out.println("x= "+x);
System.out.println("y= "+y);
System.out.println("Sum y= "+s);
}
}
class Mdemo  {
    public static void main(String arg[ ]) {
   Demo d = new Demo( );
  d.output( );
 }
}
```

Explanation:

In the above program Demo( ) is the constructor and it has no arguments. The constructor Demo( ) is used to initialize the instance variables x = 25, y = 35 and s=0. The above constructor is executed immediately after the object d is created.

1.14.1 Parameterized Constructor:

A constructor with argument is called parameterized constructor. The parameterized constructor can be overloaded.

```
class Class-name   {
   // Instance variables;
     Class-name( type arg1, type arg2)  //Constructor
        {
            // inizialization
         }
 type methodname( )   {
      // Body of the method;
  }
}
```

Example:

```
class Demo {
int x, y, s;
Demo(int x1, int y1)   {
  x=x1;
```

```
    y=y1;
    s=0;
    }
  void output( ) {
   s=x+y;
  System.out.println("x= "+x);
  System.out.println("y= "+y);
  System.out.println("Sum y= "+s);
  }
  }
  class Mdemo   {
    public static void main(String arg[ ])          {
    Demo d = new Demo(10,20 );
   d.output( );
        }
   }
```

Explanation:

In the above program Demo( ) is the constructor and it has two arguments. The constructor Demo( ) is used to initialize the instance variables x = x1, y = y1 and s=0. The above constructor is executed immediately after the object d is created and assigns the value to the variables x and y.

Unit II

A CLOSER LOOK AT CLASSES AND METHODS

2.1 Overloading Methods:

Method overloading is a type of polymorphism in java. It is a process in which two or more methods in a class have the same name with different codes. These methods have different number of arguments, different types of arguments, the order of arguments may differ and they may have different return type.

The following example demonstrates method overloading.

//Method Overloading – To calculate Area of a square, Rectangle and Triangle

```
import java.io.*;

class Ovload {

double l,b,s,r;

void area(double s1) {

double a;

a=s1*s1;

System.out.println("Side="+s1);

System.out.println("Area of Square="+a);

}

void area(double l1, double b1) {

double a;

a=l1*b1;

System.out.println("Length="+l1);
```

```
System.out.println("Breadth="+b1);
System.out.println("Area of Rect="+a);
}
void area(double pi, double r1, double r2) {
double a;
a=pi*r1*r1;
System.out.println("Radius="+r1);
System.out.println("Area of Circle="+a);
}
}
class Movload {
public static void main(String arg[ ]) {
Ovload d1,d2,d3;
d1=new Ovload( );
d2=new Ovload( );
d3=new Ovload( );
d1.area(10.0);
d2.area(10.0,20.0);
d3.area(3.14, 10.0,10.0);
}
}
```

Explanation: In the above example the class ovload contains a method area( ) with three different signatures. The method area( ) will be called based on their signature.

2.2 Overloading Constructor:

In java similar to overloading methods, constructors can also be overloaded. If any contains more than one constructor then the constructors are overloaded. The overloaded constructors have different types of arguments, different number of arguments and the order of arguments may also be different.

Example: Java program to demonstrate Constructor Overloading.

```
import java.io.*;

class Consovload {

double l,b,s,r;

Consovload(double s1) {

double a;

a=s1*s1;

System.out.println("Side="+s1);

System.out.println("Area of Square="+a);

}

Consovload(double l1, double b1) {

double a;

a=l1*b1;

System.out.println("Length="+l1);

System.out.println("Breadth="+b1);
```

```
System.out.println("Area of Rect="+a);
}
Consovload(double pi, double r1, double r2) {
double a;
a=pi*r1*r1;
System.out.println("Radius="+r1);
System.out.println("Area of Circle="+a);
}
}
class Mconsovload {
public static void main(String arg[]) {
Consovload d1=new Consovload(10);
Consovload d2=new Consovload(10,2);
Consovload d3=new Consovload(3.14,10,10);
}
}
```

Explanation: In the above example the class Consovload contains a three constructors with different number of arguments. The constructors will be executed based on the number of arguments.

2.3 Using Object as Parameter:

The simple types int, char, float, double are used as parameters. It is also possible to pass an object to methods. Similarly we can also use object as parameter in a

constructor or in a method. In the given below example the c1, c2 and c3 are the objects of the class Complex. The object c3 is passed as argument in the method process( ).

Example: Complex number addition - Passing Object as Parameter

```
import java.io.*;
class Complex {
double rp,ip;
Complex( ) {
rp=0.0;
ip=0.0;
}
void input(double rp1, double ip1) {
rp=rp1;
ip=ip1;
 }
void process(Complex c)
 {
Complex t=new Complex( );
t.rp=rp+c.rp;
t.ip=ip+c.ip;
if(t.ip>=0)
System.out.println(t.rp +"+i"+ t.ip);
```

```
else
System.out.println(t.rp +"-i" + t.ip);
}
void output( ) {
if(ip>=0)
System.out.println(rp +"+i"+ ip);
else
System.out.println(rp +"-i" + ip);
}
}
class Pasobj {
 public static void main(String arg[])  {
Complex c1=new Complex( );
Complex c2=new Complex( );
Complex c3=new Complex( );
c1.input(5,7);
c2.input(2,-3);
c1.output( );
c2.output( );
```
c1.process(c2);
```
  }  }
```

2.4 Argument Passing:

In java there are two ways to pass an argument to methods. They are call-by- value and call-by-reference. In call-by reference method the value of an argument is copied into the formal parameters of the method. The changes made to the formal parameter have no effect on the argument. In call-by-reference method a reference to an argument is passed to the parameter. The changes made to the formal parameter will also have the effect in the argument used to call the method.

Example: Call-by-value

```
import java.io.*;

class Pasval {

int x,y;

void input(int x1,int y1) {

x1*=2;

y1/=2;

System.out.println("x="+x1);

System.out.println("y="+y1);

}

}

class Mpasval {

 public static void main(String arg[])  {

    Pasval p1=new Pasval();

          p1.input(10,20);

} }
```

Example: Call-by-reference

```
import java.io.*;
class Pasref {
int x,y;
Pasref(int x1,int y1) {
x=x1;
y=y1;
System.out.println("x="+x);
System.out.println("y="+y);
}
void input(Pasref p) {
p.x*=2;
p.y/=2;
System.out.println("x="+x);
System.out.println("y="+y);
}
}
class Mpasref {
 public static void main(String arg[])
  {
    Pasref p1=new Pasref(50,60);
```

```
        System.out.println("x="+p1.x);
        System.out.println("y="+p1.y);
        p1.input(p1);
        System.out.println("x="+p1.x);
        System.out.println("y="+p1.y);
}
}
```

2.5 Returning Objects:

In java a method can return any type of data ie class type, integer, character, double, float, String or Boolean. If any method returns a class type data then it is called returning object. In this case the receiving type must be the same class type. In the example c1, c2 and c3 are the objects of the class Complex. The object c2 is passed as argument by the method process( ). The method process( ) performs the complex number addition and it returns the object ***t*** of the class Complex to the main( ).

Example: Complex number addition using passing object and returning object.

```
import java.io.*;
class Complex {
double rp,ip;
Complex( ) {
rp=0.0;
ip=0.0;
}
```

```
void input(double rp1, double ip1) {
rp=rp1;
ip=ip1;
}
Complex process(Complex c) {
Complex t=new Complex( );
t.rp=rp+c.rp;
t.ip=ip+c.ip;
return(t);
}
void output( ) {
if(ip>=0)
System.out.println(rp +"+i"+ ip);
else
System.out.println(rp +"-i" + ip);
}
}
class Retobj {
 public static void main(String arg[])  {
Complex c1=new Complex( );
Complex c2=new Complex( );
```

```
Complex c3=new Complex( );

c1.input(23.2,9.7);

c2.input(6.4,8.0);

c1.output( );

c2.output( );

c3=c1.process(c2);

c3.output( );

  }

}
```

2.6 Recursion:

Recursion is the process of calling a method by itself. The method is said to be recursive method. In the example fact( ) is the recursive method. It computes the factorial of 5 by calling itself and it returns the factorial value to the main( ).

Example: Program to find the factorial of a given number using Recursion.

```
import java.io.*;

class Recur {

long f;

long fact(int n) {

if (n==1)

return 1;

else

f=n*fact(n-1);
```

```
return f;

}

}

class Mrecur {

public static void main(String arg[ ]) {

Recur r=new Recur( );

long f1;

f1=r.fact(5);

System.out.println("Factorial of "+ 5 +"is =" +f1);

}

}
```

2.7 Introducing Access Control:

Access control prevents the misuse of data and methods in java. Access specifier determines how a member can be accessed? in a class. Java supports three access specifiers. They are public, private and protected. If no access specifier is used, then by default the members of the class are public within its own package.

public specifier- The public members can be accessed by the members (methods and objects) of the class where it is declared and it can also be accessed by the methods outside the class.

Example:

```
public int n1,n2,n3;
```

private specifier- The private members can be accessed only by the members of the class where it is declared.

Example:

```
private double p,n,r;
```

protected specifier- The protected members are similar to private members except that it can be accessed by the sub-class members (used in inheritance).

Example:

```
protected int x,y,z;
```

Example: The following program demonstrates the use of the various access specifiers.

```
import java.io.*;
import java.lang.*;
class Demoaccs {
int a;
public int b;
private int c;
void output( ) {
System.out.println("a= "+a);
System.out.println("b= "+b);
System.out.println("c= "+c);
}
}
class Mdemoaccs {
public static void main(String ar1[ ]) {
```

```
Demoaccs d=new Demoaccs( );
 d.a=60;
d.b=20;
d.c=30;// Error Private member
  d.output( );
  }
}
```

Explanation: In the above program the class Demoaccs contains the instance variables a is public (default) ,b is public and c is private. The default and public variables can be accessed from outside the class ie the value for a nd b are assigned in main( ). An attempt to assign the value to c creates error, since c is a private member. The private member can be accessed by the constructor or methods of that class.

2.8 Understanding static:

In java the member of the class must be accessed with the object of that class. It is also possible to access the member without its object. When we declare a member as static, it can be accessed directly without reference to any object. The keyword static can be used to declare both the instance variables and methods of the class as static member. The static methods can access only static members but the non-static methods can access both static and non static instance variables.

```
Class Demoddce  {
        public static void main(String arg[ ] )
         {
               System.out.println("Main as staic");
         }    }
```

In java the main( ) declared as static since it must called before any object is created.

Ensure the following while using static members:

(i) Static instance variables of a class are shared by all the objects of that class.
(ii) Static methods can access only static instance variables or other static methods.
(iii) Static members cannot refer to this or super( ).

Example: Java program to demonstrate static members

```
import java.io.*;

import java.lang.*;

class Demostat {

static int a;

static int b;

Demostat(int a1,int b1) {

a=a1;

b=b1;

}

static void output( ) {

System.out.println("a= "+a);

System.out.println("b= "+b);

}

}

class Mdemostat {

public static void main(String ar1[ ]) {
```

```
Demostat d=new Demostat(10,20);

Demostat.output( );

 }

}
```

Explanation: The class Demostat contains the static members a, b and the method output( ). Using the static method output( ) the static instance variables are accessed. An attempt to made by a non static instance variable creates an error.

2.9 Introducing final:

The keyword ***final*** can be used to declare a variable as constant. It will not permit to modify the value of a variable. The final variable must be initialized, when it is declared. In the example the instance variables are declared constants by the keyword final. The method output( ) is also declared final. An attempt to change the value of x in output( ) creates an error. Since x is a constant.

Example: Java program to demonstrate the keyword final

```
Import java.io.*;

class Demofinal

{

public static void main( String arg[ ] )

{

final int x = 10; // final variable

final int y = 20; //final variable

final void output( ) // final method

{
```

x = x+y; //Error since x is a final variable

System.out.println(“x = “ + x); //x=10

System.out.println(“y = “ + y); //y=20

}

}

}

Uses of the ***final*** keyword:

(i) Used to declare a constant variable and method.
(ii) Used to prevent method overriding.
(iii) Used to prevent Inheritance.

2.10 Command Line Arguments:

Command line arguments are useful to pass data to main ( ) when you run the program. The command line argument is the data that is directly next to program‟s name on the command line. All data can be treated as String object. String object can be converted to appropriate data types musing suitable methods.

Example:

C:\>DDCE\Javaprogram>java Democmdln 10 2.3 “Raja”

In the above example 10, 2.3 and “Raja “ are the data items passed to main( ) using the command line arguments.

Example: Java program to find the sum of two numbers using command line argument.

import java.io.*;

class Cmdlnarg {

int x,y,s;

```
void input(int x1, int y1) {
x=x1;
y=y1;
}
void display( ) {
s=x+y;
System.out.println("X="+x);
System.out.println("y="+y);
System.out.println("Sum ="+s);
}
}
class McmdInarg {
public static void main(String arg[ ]) throws IOException {
int xx,yy;
CmdInarg d=new CmdInarg( );
xx=Inteqer.parseInt(arg[0]);
yy=Inteqer.parseInt(arg[1]);
d.input(xx,yy);
d.display( );
}
}
```

Explanation: In the above example the value of xx and yy are passed through command line. Integer.parseInt( ) method can be used to convert the String object into integer type data. The input( ) is used to assign the value of x and y. The display method computes the sum of x and y and it display the values.

INHERITANCE

2.11 Inheritance:

Inheritance is one the basic concepts of object oriented programming. Inheritance

Is the process of creating a new class with the traits of existing class. The existing

class (inherited class) is called super class and the newly created class (inheriting

class) is called sub class. The sub class (inheriting class) inherits all the instance variables and methods of super class (inherited class) except the private members and additional instance variables and methods can also be declared.

The keyword ***extends*** is used to create a sub class.

2.11.1 Single Inheritance:

Single inheritance is the process of creating a sub class from a single super class. It is also called as simple inheritance. Java does not support multiple inheritance. But we can implement multiple inheritance through interface.

General form for single inheritance:

class Superclass

{

protected instance variables;

- - - - -
- - - -
- - -

```
type superclassmethod ( )
{
   // body
    -   - -
}
}
class Subclass extends Superclss
{
Subclass  instance variables;
    -   - - - -
    -   - - -
type subclassmethod ( )
{   body
    -   - -
}
}
```

Example : Java program to demonstrate single inheritance

```
import java.io.*;
class Base {
double l;
Base(doubl e l1) {
```

```
l=l1;
}
void area( ) {
double a=l* l;
System.out.println("Area of FSquare= "+a);
}
}
class Rect extends Base {
double b;
Rect(double l1,double b1) {
super(l1);
b=b1;
}
void area( )
{
double a=l*b;
System.out.println("Area of Rectangle = "+a);
}
}
class Minheritance1 {
public static void main(String args[]) {
```

```
Rect r=new Rect(5.0,6.0);

Base b=new Base(7.0);

b.area( );

r.area( );

}

}
```

2.12 Using super:

In java super keyword is used by the sub class to refer its super class. It is mainly used in Inheritance and method overriding. This super keyword has three uses.

(i) Used to call the super class constructor.
(ii) Used to access the super class instance variables.
(iii) Used to access the super class methods.

2.12.1 Keyword super to call super class constructor

The keyword super can be used to call the super class constructor. The general form is given by

super(argument-list);

Example:

(i) super(10,2.4);
(ii) super(a,b);

2.12.2 Keyword super to access super class instance variable.

The keyword super can be used to access the super class instance variable. It can access the super class instance variable only when it is declared as public or protected. Private instance variable of the super class cannot be accessed by super. The general form is given by

super.variable=value;

super.variable;

Example:

Let x and y be the protected integer variables declared in the super class, then it can be accessed as follows using super.

super.x=15;

super.y=56;

System.out.println("X="+**super.x**);

System.out.println("Y="+**super.y**);

2.12.3 Keyword super to access the super class method.

The keyword super can be used to access the super class methods. It can able to access the super class public and protected methods. Private methods of the super class cannot be accessed by super. The general form is given by

super.super_class_method(arguments-list);

Example:

super.output( );

super.process( );

In the above example the methods output( ) and process( ) must be protected or public methods, declared in the super class.

Example: Java program to demonstrate the use of the keyword *super*.

import java.io.*;

class Demo {

```
 int x,y;
Demo(int x1,int y1) {
x=x1;
y=y1;
}
 void display( ) {
  System.out.println("Super x : "+x);
  System.out.println("Super y : "+y);
  }
}
class Demo1 extends Demo {
int z;
Demo1(int x1, int y1, int z1) {
super(x1,y1);//Calling super class constructor
z=z1;
}
  void display( )   {
   super.display();// Calling supercalss method
   System.out.println("sub z: "+z);
      }
 }
```

```
class Moverriding1  {

  public static void main(String args[ ]) throws IOException   {

    Demo1 d1=new Demo1(10,20,30);

    d1.display( );

  }

 }
```

2.13 Multilevel Hierarchy:

In multilevel hierarchy the user can able to create sub classes in different levels. In multilevel hierarchy the super class create a sub class, and then this sub class becomes a super class and creates another sub class. For example, consider three classes namely Demo, Demo1 and Demo2. Here Demo is the super class and Demo1 is the sub class of Demo, again Demo1 become a super class and it creates another sub class Demo2. This process will for many levels. Basically multilevel hierarchy is single inheritance in different levels.

General form for multilevel inheritance:

```
class Superclass

{

protected  instance variables;

    -   - - - -
    -   - - -

type superclassmethod ( )

{   body

    -   - -
```

```
}

}

class Subclass1 extends Superclass

{

Subclass1  instance variables;

    -   - - - -
    -   - - -

type subclass1_method ( )

{   body

    -   - -

}

}

class Subclass2  extends Subclass1

{

Subclass2  instance variables;

    -   - - - -
    -   - - -

type subclass2_method ( )

{   body

    -   - -

}

}
```

Example : Java program to demonstrate multilevel inheritance

```
import java.io.*;
class Demo {
int x;
Demo(int x1) {
x=x1;
}
 void display( ) {
  System.out.println("Demo x : "+x);
  }
}
class Demo1 extends Demo {
int y;
Demo1(int x1, int y1) {
super(x1);//Calling super class constructor
y=y1;
}
  void display( )   {
   super.display( );// Calling supercalss method
   System.out.println("Demo1 y: "+y);
  }  }
```

```
class Demo2 extends Demo1 {
int z,s;
Demo2(int x1, int y1,int z1) {
super(x1,y1);//Calling super class constructor
z=z1;
}
  void display( ) {
   s=x+y+z;
   super.display( );// Calling supercalss method
   System.out.println("Demo2 Z: "+z);
   System.out.println("Sum = "+s);
  }
  }
class Mmullvlinh {
  public static void main(String args[ ]) throws IOException    {
    Demo2 d= new Demo2(10,20,30);
    d.display( );
     }
 }
```

Explanation: In the above program Demo is the super class and Demo1 is the sub class of Demo. Demo1 inherits all of the traits of Demo and also it adds its own member y. In the second level Demo1 is the super class for Demo2. It inherits all the traits of Demo1

and adds its own members z and s. Similarly we can construct hierarchies that contains many levels of inheritance.

2.14 Method Overriding:

In Inheritance, when a method in a sub class has the same name, same number of arguments, same type of arguments, arguments in the same order as a method in its super class, then the sub class method override the super class method. When the sub class method is called, it will always refer to the sub class method and will not refer the super class method. This mechanism in java is called method overriding. The super class method can be called using the keyword ***super*** .

Example: Java program to demonstrate method overriding.

```
import java.io.*;

class Demo {

int x,y;

Demo(int x1,int y1) {

x=x1;

y=y1;

}

 void display( ) {

  System.out.println("Super x : "+x);

  System.out.println("Super y : "+y);

  }

}
```

```
class Demo1 extends Demo {
int z;
Demo1(int x1, int y1, int z1) {
super(x1,y1);//Calling super class constructor
z=z1;
}
void display( )  {
//   super.display();// To access supercalss method
   System.out.println("sub z: "+z);
     }
 }
class Moverriding1  {
  public static void main(String args[ ]) throws IOException   {
    Demo1 d1=new Demo1(15,25,35);
    d1.display( );
 }
 }
```

Explanation: In the above example Demo is the super class and Demo1 is the sub class. The classes Demo and Demo1 have the same method void output( ) with the same signature. Whenever you call the sub class method, it always call the sub class method and it won‟t call the super class output( ). This is because the sub class method overrides the super class method.

2.15 Dynamic Method Dispatch:

Method overriding forms the basis for Dynamic method dispatch. When a overridden method is called through a super class reference, java determines which method has to be executed at the time the call occurs. This is based on which sub class object is assigned to the super class reference. This mechanism is called dynamic method dispatch. In java, run-time polymorphism can be implemented using Dynamic method dispatch.

Example: Java program to demonstrate Dynamic method dispatch (Polymorphism).

```
import java.io.*;

class Demo {

int x;

Demo(int x1 ) {

x=x1;

}

 void display( )  {

  System.out.println("Demo x : "+x);

  }  }

class Demo1 extends Demo {

int y;

Demo1(int x1, int y1) {

super(x1);//Calling super class constructor

y=y1;
```

```
}
  void display( )   {
   System.out.println("Demo1 y: "+y);
  }
 }
class Demo2 extends Demo1 {
int z,s;
Demo2(int x1, int y1,int z1) {
super(x1,y1);//Calling super class constructor
z=z1;
}
  void display( )   {
   s=x+y+z;
   System.out.println("Demo2 Z: "+z);
   System.out.println("Sum = "+s);
  }
 }
class Dynmtddispch  {
public static void main(String args[ ]) throws IOException    {
Demo d=new Demo(7);
Demo1 d1=new Demo1(12,45);
```

```
Demo2 d2= new Demo2(100,200,300);

Demo dd;

dd=d;

dd.display( );

dd=d1;

dd.display( );

dd=d2;

dd.display( );

 }

 }
```

Explanation: In the above example Demo is the super class, Demo1 is the sub class of Demo and Demo2 is the sub class of Demo1. In the main method create the objects d, d1 and d2 for the classes Demo, Demo1 and Demo2 respectively. dd is the reference variable of the super class. Assign the object to the reference variable dd and call the method display( ). The call to the display ( ) will be resolved during run-time.

Unit III

PACKAGES AND INTERFACES

3.1 Packages:

Packages are containers for classes and interfaces. It is a collection of related classes and interfaces. The package restricts the visibility for the classes inside a package and naming mechanism. The programmer can define classes inside a package that are not accessible by the code outside the package or that may be accessible by the code outside the class.

3.1.1 Defining a package:

The keyword package is used to create a package. The package statement must be the first statement in the source file. The classes that are declared within that file will belong to the specified package. The keyword package defines a name space in which classes are stored. In the absence of package keyword, the classes are stored in a default package. The general form of package is given blow.

```
package package-name;
```

Example:

```
package demopack1;
```

In the general form package-name is the name of the package. In the above example demopack1 is the package name. The package demopack1 must be stored in the directory called demopack1.

3.1.2 Adding a class to a package:

Before adding a class to a package, create a package. The class to be added to the package must follow the package definition.

Example:

```
package student;

class Stu

{

String name;

int roll-num;

//Body

}
```

In the above example the package student is stored in the directory and the class Stu is in the package student.

Example: Program to demonstrate package

```
package demopack;

class Student {

String name;

int rn;

Student(String name1, int rn1) {

name=name1;

rn=rn1;

}

void output( ) {

System.out.println("Name :"+name);
```

```
System.out.println("RNum :"+rn);

}

}

class Mstudent {

public static void main(String arg[ ]) {

Student s=new Student("Raja",1234);

s.output( );

}

}
```

Explanation:

In the above example demopack is a package. This package must be stored in the directory called demopack. The program file name and the class name should be same. In this example the file name is Mstudent.java. The package demopack contains two classes namely Student and Mstudent. Compile the program so that the class file must be in the directory demopack. Now go one level back in the directory. Run the program with the package qualifier.

C:Javaprg\Demopack>javac Demopack.java

C:>Javaprg>java Demopack.Mstudent

3.2 Access Protection:

Packages are the containers for classes, interfaces and other sub-packages. Classes are the containers for data and methods. Java supports the following access specifiers namely private, protected, public and default (ie no access specifier). The visibility of packages, sub-packages, classes and sub-classes are given in the following table.

	Private	Protected	Public	Default
Same class	Yes	Yes	Yes	Yes
Same package sub class	No	Yes	Yes	Yes
Same package Non sub class	No	Yes	Yes	Yes
Different package sub class	No	Yes	Yes	No
Different package non sub class	No	No	Yes	No

3.3 Importing Packages:

The import statement is used to bring the classes and interfaces in a package to use it in a program. The import statement occurs immediately following the package statement and before any class definitions. The packages, sub-packages , classes and methods are separated by the dot(**.**) operator. The general form of the import statement is given below.

```
import package1.sub-package2.class.*;
```

Example:

```
import demopack.demosub.Student.*;
```

Example: Step 1:Program to create a package demopack which contains a file Student.java.

```
package demopack;

class Student {

String name;

int rn;

Student(String name1, int rn1) {

name=name1;

rn=rn1;

}

void output( ) {

System.out.println("Name :"+name);

System.out.println("RNum :"+rn);

} }
```

Step2: Program to import a package.

```
import java.io.*;

import demopack.Student;

class Packdemo {

public static void main(String arg[ ]) {

Student s1=new Student("DDDD",1234);

s1.output( );

}

}
```

Explanation:

The above example shows a program that imports the class Student from the package demopack. The source file should be saved as Packdemo.java and then compiled. The source file (Packdemo.java) and its class file (Packdemo.class) must be saved in the directory (javaprg) and demopack should be the sub directory of javaprg. Now run the program Packdemo.java.

3.4 Interfaces:

An interface is similar to classes but it can declare only abstract methods and final variables. This means that interfaces do not specify any code to implement these methods and constants. An interface has two parts namely defining an interface and implementing an interface. In java Multiple Inheritance can be implemented using Interfaces.

3.4.1 Defining an Interface:

An interface is defined like a class but it contains only final variables and abstract methods. The general form of an interface is as follows.

```
interface Interface-name
{
final variable1=value;
_ _ _ _
Abstract type method(arguments);
_ _ _ _
}
```

Example: Program to define Interface

```
interface Areaiface {
```

```
final double pi=3.14;

void carea(double r);

}
```

Explanation:

In the above example Areaiface is an interface. It contains two members namely a final constant pi and an abstract method Carea(double) with double type argument.

3.4.2 Implementing Interfaces:

Once an interface is defined, many classes can implement the interface. The implements clause can be used to implement an interface. Interfaces are used as super classes whose members are used in sub classes. The general form of implementing an interface is given below.

```
class Class-name implements Interface-name1, Interface-name2 ...

{

//Body of the class

}
```

Example: Program to implement an Interface

```
class Acircle implements Areaiface {

double r,ac;

public void carea(double r1) {

r=r1;

ac=pi*r*r;

System.out.println("Radius="+r);
```

```
System.out.println("Areaof Circle="+ac);
}
}
class Macircle {
public static void main(String arg[ ]) {
Acircle a1=new Acircle( );
a1.carea(10.0);
}
}
```

Explanation:

In this program the class Acircle implements the interface Areaiface. The interface consists of one final constant pi and an abstract method carea(double). The abstract method carea(double) is defined inside the class Acircle. The class Macircle contains the main( ) method through which the carea( ) is called to calculate the area of a circle.

EXCEPTION HANDLING

3.5 Exception Handling:

The abnormal condition that arises in a program during run time is called exception. In other words the run time error is called exception. Java uses the five keywords try, catch, throw, throws and finally to handle the exception.

try block: The set of codes that you want to monitor for exceptions are enclosed within try block.

catch clause: Used to handle the exception thrown by the try block.

throw: Used to throw the exception manually.

throws: Used by the methods to throw out the exception.

finally block: The code that must be executed before a method returns is enclosed in the finally block.

The general form of try-catch block is given below.

```
try {
  // set of code to be monitored
}
catch(Exceptio-type1 exobj1 )
 {
   // methods to handle exception1
}
 catch(Exceptio-type2 exobj2 )
{
   // methods to handle exception2
  }
_ _ _
   finally
    {
     // codes to be executed before tru block ends
      }
```

Example: Program to demonstrate the exception handling (Divide by zero).

```
import java.io.*;

class Mexcep1

{

public static void main(String arg[ ]) {

int x,y,z;

try{

x=20;

y=20;

z=x/(x-y); // Divide by zero error

}

catch(ArithmeticException e){

System.out.println("Divide by Zero ERR"+e);

}

Finally {

System.out.println("final Block z=");

}

System.out.println("After Catch");

}

}
```

Explanation: In the above program divide by zero exception occurred. This exception is handled by the catch clause.

3.6 Exception Types:

In java all exception types are subclasses of the built-in-class Throwable. The two subclasses of Throwable are Exception and Error. Some of the built-in exceptions listed below.

Exception	Meaning
1 ArithmeticException	Arithmetic errors similar to divide by zero
2. ArrayIndexOutOfBoundsException	Array index exceeds the range
3. NegativeArraySizeException	Array index is negative
4. ClassCastException	Invalid cast
5. NullPointerException	Invalid use of a Null pointer
6. StringIndexOutOfBounds	Outside the bounds of a String

3.7 try-catch block:

The default exception handling mechanism by the java run-time system is useful for debugging. The run time error can be handled using try – catch block. A try block and a catch clause form a unit. The code that you want to monitor is enclosed within the try block. Immediately following the try block, include a catch clause that specifies the exception type that you wish to catch.

Example: Java Program to demonstrate Array Index Out of Bounds Exception.

```
import java.io.*;

class Mexcepbnd {

public static void main(String arg[ ]) {
```

```
try{
int a[ ]=new int[5];
int i,s;
for(i=0;i<5;i++)
a[i]=i;
a[20]=15; // Array Index out of Range
for(i=0;i<5;i++)
System.out.println(a[i]);
}
catch(ArrayIndexOutOfBoundsException e) {
System.out.println("Array Index Out of Range EXE");
}
System.out.println("After Catch");
}
}
```

Explanation: In the above program the size of the array ***a*** is 5, ranging from 0 to 4. But it tries to assign 15 to a[20]. The array indexes 20 exceeds the array size so an exception is occurred and the catch clause catch the exception and handle it. The message is displayed on the screen.

MULTITHREADED PROGRAMMING

3.8 Java Thread Model:

A thread is similar to a program that has a single flow of control. Java provides built-in support for multithreading. In java a multithreaded programs contains two or more parts that can be executed in parallel. One part of such program is called Thread. In most of the computers, we have only one processor. Therefore in reality the processor is doing only one job at a time. But the processor switches between the processes. The java model helps the programmer to write efficient programs so that the cpu utilization can be increased and the cpu idle time is reduced.

The advantage of multithreading is that the main loop /mechanism is avoided. One thread can stopped or suspended without affecting other threads. For example, one thread may in sleep and the other thread is in execution. The sleeping thread resumes the operation when it got the chance enters into the cpu for execution.

3.8.1 Thread Life Cycle:

In java thread exists in several states. They are new, running, runnable, blocked and dead.

New state – Thread object is created and is not yet scheduled for running.

Runnable state - Thread is ready for execution and and is waiting for cpu time.

Running state – Thread is in cpu and is in execution.

Blocked state – Thread is prevented from entering into the runnable state and the running state.

Dead state - Thread completed its execution and quits from the cpu.

3.8.2 Thread Priority:

Thread priority is an integer number assigned to the threads. It shows the relative priority of one thread over another thread. Thread priority value is between 1 and 10.

MIN_PRIORITY=1 and MAX_PRIORITY=10. Thread‟s priority decides when the running thread quits the cpu. The setPriority( ) method can be used to set the Thread priority. The general form is given below.

Threadname.setPriority(int number);

Example: Demothread.setPriority(4);

3.8.3 The Thread class and the Runnable Interface:

The program must extend the Thread class or implement the runnable interface to create a new thread. The following are the some of the methods defined by the Thread class to manage the thread.

final String getName( ) – Returns the thread name.

final void setName(String) – Used to set the thread name

int getPriority( ) – Returns the thread‟s priority

boolean isAlive( ) – Returns true if the thread is alive else false

void run( ) – Entry point for the thread

void sleep(long) – Used to suspend a thread for a specified time

void start( ) – Start a thread by calling its run( ) method.

3.9 Main Thread:

The main thread is the thread that starts up, when a java program begins running. The main thread is created automatically when your program is started. It can be controlled through a Thread object and the Thread class methods. The purposes of main thread are

- Child threads will be spawned from the main thread.

- Main thread is the last thread to finish execution because it performs various shutdown actions.

3.10 Creating a Thread:

There are two different ways to create a thread in java. They are

- Implement the Runnable interface
- Extend the Thread class

3.10.1 Creating a Thread by implementing Runnable interface:

The simplest method to create a thread is to create a class that ***implements*** the Runnable interface. The class need to use the run( ) method, to implement the Runnable interface. The body of the run( ) contains the code for the new thread. The start( ) method executes a call to run( ). The general form of the methods used to create the Thread is given below.

- public void run( )
- void start( )
- Thread constructor is

  Thread(Runnable threadob, String thread-name)

Example: Java program to create a Thread using Runnable interface.

```
import java.awt.*;

class Demothread implements Runnable {

  Thread t;

  Demothread( ) {

    t=new Thread(this,"DEMOTHREAD");

    System.out.println("CHILD "+t);
```

```
        t.start( );
      }
public void run( )   {
   try  {
    for(int i=10;i<15;i++)
      {
       System.out.println("Child Thread "+i);
       Thread.sleep(500);
       }
      } catch(InterruptedException e)        {
         System.out.println(" CT INTD  ... ");
       }
   }
  }
class Mthreadr1  {
   public static void main(String args[ ]) {
     Demothread tt=new Demothread( );
     try{
       for(int i=25;i>20;i--)
         {
          System.out.println("MT  "+i);
```

```
        Thread.sleep(500);
        }
     }catch(InterruptedException e)   {
       System.out.println("MT iNTD ...");
      }
     System.out.println("MT exited");
   }
}
```

3.10.2 Creating a Thread by Extending Thread class:

The second method to create a thread is to create a class that extends Thread and then it creates the thread object. The extending class must override the run( ) method, which is the entry point for the new thread. It must also call start( ) to begin execution of the new thread.

Example: Java program to create a thread by extending Thread class.

```
import java.awt.*;
import java.io.*;
class Demothread extends Thread {
 Demothread( ) {
  super("DEMO THREAD");
  System.out.println("CT.."+this);
  start( );
  }
```

```
public void run( ) {
  try
   {
    for(int i=20;i<25;i++)
     {
      System.out.println("CT "+i);
      Thread.sleep(250);
      }
      } catch(InterruptedException e)   {
      System.out.println(" child intd");
      }
    System.out.println("ct exited");
  }
 }
class Mthreadt1 {
 public static void main(String args[ ])   {
   Demothread t1= new Demothread( );
try
   {
    for(int i=300;i<305;i++)
     {
```

```
      System.out.println("MT.. "+i);
      Thread.sleep(500);
      }
      }catch(InterruptedException e)   {
      System.out.println(" Main Thread Intd");
      }
    System.out.println("MT exited");
  }
}
```

Unit IV

THE APPLET CLASS

4.1 Applet:

The Applet class is contained in the java.applet package. Applet contains several methods to control the execution of the Applet. There are three interfaces defined by the package java.applet namely AppletContext, AudioClip, and AppletStub.All applets are sub classes of Applet. All applets must import java.applet. Applets must also import java.awt. AWT stands for Abstract Window Toolkit. Since all applets run in a window, include awt classes.

The applet cane be run in a web browser or in an appletviewer. Execution of an applet does not begin at main( ). The applet will be executed by a java-enabled web browser when it encounters the Applet tag within the HTML file. You can also test the compiled applet by starting the appletviewer with your java source code file. Output to the applet is handled by the awt methods such as drawstring( ). The drawstring( ) outputs a string to a specified X, Y location.

Example:

```
/* <Applet  code = "Demoapplet"     width = 300     Height = 100>
 </Applet>
*/
```

The above example shows an Applet with height of 100 pixels and width of 300 pixels. Demoapplet is the file of the applet program.

4.1.1 The Applet class:

The Applet class defines the methods to manage the applet. Some of the Applet class methods are given below.

(i) void destroy( ) - Called by the browser just before an applet is terminated.

(ii) String getAppletInfo( ) -- Returns a string that describes applet.

(iii) void init( ) – Called when an applet begins execution.

(iv) boolean isActive( ) – Returns true if the applet is started otherwise false.

(v) void play(URL urladdress) – To play the audio clip in the specified location.

(vi) void paly(URL urladdress, String filename) – To play the audio clip in the specified address and name.

(vii) void resize(Dimension dim) – To change the size of an applet as per the new dimension.

(viii) void resize(int width, int height) – Used to change the size of an applet.

(ix) void start( ) – Called by the browser when an applet should startexecution.

(x) void stop( ) – To stop the execution of an applet.

4.2 Applet Architecture:

Java supports two types of programs namely Application program and Applet program. In the previous chapters we discussed about application programs. An Applet is a window-based program. Its architecture is different from he application programs. The concepts about applet are given below.

- Applets are event driven. An applet waits until an event occurs.
- The user initiates interaction with an applet. For an application program the input can be fed through readLine( ) method. This will not work in applet programming. Events are generated when the user select a Checkbox, press the Button, pressed a key .....

4.3 Applet Skeleton:

Applets override a set of methods that provides the basic mechanism by which the browser or appletviewer interfaces to the applet and controls its execution. The four

basic methods needed for an applet are init( ), start( ), stop( ) and destroy( ). The paint( ) is defined by the AWTComponent class. The following program shows the Applet skeleton.

4.3.1 Applet Life Cycle:

Applet inherits a set of default behaviours from the Applet class. As a result when applet is loaded, AWT calls the following methods in the following sequence.

- init( ) → This method is the first method to be called. It is used to initialize the variables. This method is called only once during the runtime of your applet.
- start( ) → This method is called after init( ). It is also called to restart the applet. The start( ) is called each time an applet‟s HTML document is displayed on the screen.
- paint( ) → This method is called every time to redraw the output. The paint( ) has one parameter of type graphics.

The following methods are called, when an applet is terminated.

- stop( ) → This method is called when a browser leaves the HTML document containing the applet.
- destropy( ) → This method is called when the applet needs to be removed completely from memory.

4.4 Applet Display Methods:

Applets are displayed in a window and they use the AWT to perform input and output. Applet use the drawstring( ) of the Graphics class to output a String to an Applet. This method is called from either paint( ) or update( ). The general form is given below.

```
void drawString(String message, int xcoodnate, int ycoodnate);
```

Example:

```
public void paint(Graphics g)
{
 g.drawString( “DDCE – MSU – WELCOMEs YOU ALL”, 100,100);
}
```

The above segment display the message in the applet at row number 100 and column number 100. The upper left corner is location 0,0.

The following methods are sued to change or set the background and foreground colours in an applet.

```
void    setBackground(Color color1);
void setForeground(Color color2);
```

In the above general forms color1 and color2 shows the new colors.

Example:

```
setBackground(Color.red);
setForeground(Color.cyan);
```

In the above example the background color is red and the foreground color is cyan. Generally these methods are used in the init( ) method.

It is also possible to get the foreground and background colors with the following methods. They returns the Color object.

```
Color getBackground( );
Color getForeground( );
```

Example: Program to demonstrate the Applet display methods.

```
import java.io.*;
import java.awt.*;
import java.applet.*;
/*<applet code="Demoapplet" width=300 height=400>
        </applet> */
public class Demoapplet extends Applet  {
String msg="WELCOME TO OUR COOLLEGE";
int x,y;
public void init( ) {
setBackground(Color.cyan);
setForeground(Color.red);
}
public void start( ) {
x=10;
y=100;
 msg+="inside start";
}
public void paint(Graphics g)   {
 g.drawString(msg,x,y);
  }    }
```

4.5 Requesting Repainting:

An applet uses paint( ) and update( ) methods to write information in a window. These methods are called by AWT. Applet uses the method repaint( ) to update the information displayed on a window. A paint( ) method within a loop will not update the information displayed on a window. This repaint( ) method is defined by the AWT. The repaint9) method is available in four different forms.

- void repaint( ) → This method is used to repaint the entire window.
- void repaint(int left, int top, int width, int height) → This method is used to repaint the specified region/area.
- void repaint(long delay) → This method is used to repaint the window with a delay of mili seconds.
- void repaint(long delay, int x, int y, int width, int height) → This method is used to repaint the specified region/area after delay milli seconds.

4.6 HTML Applet Tag:

The APPLET tag is used to start an applet from both an HTML document and from an applet viewer. The general form of the APPLET tag is given below. In the general form the square bracketed items are optional.

<APPLET

[CODEBASE = codebaseURL]

CODE = applet-class-file

[ALT = Alternate-text]

[NAME = Applet-instance-name]

WIDTH = pixels

HEIGHT = pixels

[ALIGN = alignment]

[VSPACE = pixels]

[HSPACE = pixels]

>

[<PARAM NAME = Parameter-name VALUE = Parameter-value>]

.....

.....

[Text to be displayed in the absence of java.....]

</APPLET>

- CODEBASE → This is an optional attribute that specifies the base URL directory where the applet code resides.
- CODE → CODE is a required attribute that gives the name of the compiled **.class file**
- ALT → The ALT is an optional attribute used to specify a short text message that should be displayed where the applet would go.
- NAME → This is an optional attribute used to specify a name for the applet instance.
- WIDTH → WIDTH attribute is used to specify the width of the applet in pixels.
- HEIGHT → HEIGHT attribute is used to specify the height of the applet in pixels.
- ALIGN → ALIGN is an optional attribute used to specify the alignment of the applet.
- VSPACE → This is an optional attribute used to specify the space in pixels, above and below the applet.

- HSPACE → This is an optional attribute used to specify the space in pixels, on each side of the applet.
- PARAM NAME and VALUE → The PARAM tag allows you to specify applet specific arguments in an HTML page. Applets access their attributes with the getParameter( ) method.

EVENT HANDLING

4.7 Event Handling Mechanisms:

Events are supported by the java.awt.event package. The most commonly handled events are those generated by the mouse, the keyboard and other awt controls Button, Checkbox, Menubar .... .The modern approach is the way that events should be handled by all new programs.

4.8 Delegation Event Model:

Delegation event model defines standard and consistent mechanisms to generate and process events. In this model a source generates an event and sends it to one or more listeners. The listener process the event. In the delegation event model, event notifications are sending to the registered listeners.

An event is an object that describes a state change in a source. For example pressing a key, click a Checkbox, Click a Button or selecting an item using mouse or keyboard are the some of the methods to generate events.

4.8.1 Event Sources:

A source is an object that generates an event. This occurs when the internal state of that object changes in some way. The source may also generate more than one type of event. Each type of event has its own registration method. The general form is given below.

public void addListenerType(ListenerType ob);

Example:

B1.addActionListener(this);

C1.addItemListener(this);

In the above general form the ListenerType may be any one type of listener. In the example the method that registers a Button event is addActionListener( ), the method that registers a Checkbox is addItemListener( ).

The removeListenertype( ) can be used to unregister a listerner. Its general form is given below.

public void removeListenerType(ListenerType ob);

Example:

B1.removeActionListener(this);

C1,removeItemListener(this);

4.8.2 Event Listeners:

An event listener is an object that is notified when an event occurs. It has to do two steps to receive notifications.

- It must register with one or more sources about the type of events.
- It must implement methods to receive and process these notifications.

The methods that receive and process events are defined in a set of interfaces are found in **java.awt.event**.

4.9 Event Classes:

The EventObject is the superclass of all events. The AWTEvent is a superclass of all AWT events that are Handled by the delegation event model. The package **java.awt.event** defines many types of events that are generated by various user interface elements. Some of the event classes are given below.

- ActionEvent → Generated when a button is pressed, menu item is selected...
- AdjustmentEvent → generated when a ScrollBar is manipulated
- ItemEvent → Generated when a Checkbox, list item is clicked
- KeyEvent → Generated when input is received from the keyboard
- MouseEvent → Generated when the mouse is dragged, moved, clicked, pressed or released
- TextEvent → Generated when the value of a TextArea or TextField is changed
- WindowEvent → Generated when a window is closed, opened, activated, deactivated or quit.

4.9.1 ActionEvent Class:

An ActionEvent is generated when a button is pressed, a list item is double clicked, or a menu item is selected. The constructor and the methods defined by the ActionEvent class are explained below.

ActionEvent class constructor:

- ActionEvent(Object ob, int type, String cmd);

Where ob is the reference to the object that generated event, type is the type of event generated and cmd is the command string.

- String getActionCommand( ) → returns the command name equal to the label name on the button, when it is pressed.

4.9.2 ItemEvent Class:

An ItemEvent is generated when a CheckBox is or a list item is clicked or checkable menu item is selected or deselected. ItemEvent class constructor is given below.

- itemEvent(ItemSelectable src, int type, Object ob, int state);

Where src is the component that generated the event, type is the type of event generated, ob is the specific item that generated the ItemEvent and the current state of that item is is represented by state.

- Object getItem( ) → used to obtain a reference to the item that generated an event.
- ItemSelectable getItemSelectable( ) → used to obtain a reference to the ItemSelectable object that generated an event.
- int getStateChanged( ) → returns the state change for the event.

4.9.3 MouseEvent class:

The MouseEvent is generated by the following eight types of mouse events.

- MOUSE_CLICKED → The user clicked the mouse.
- MOUSE_DRAGGED → The user dragged the mouse.
- MOUSE_ENTERED → Mouse entered the component
- MOUSE_PRESSED → User the pressed the mouse
- MOUSE_RELEASED → User released the mouse.
- MOUSE_MOVED → User moved the mouse
- MOUSE_EXITED → Mouse exited from the component
- MOUSE_WHEEL → The mouse wheel was moved.

The constructor of MouseEvent class is given below.

- MouseEvent(Component evsrc, int evtype, long evwhen, int evmodifier, int x int y, int click, boolean popup);

Where evsrc is the event source, evtype is the type of the event generated, evwhen specifies the when that event took pl;ace(system time), evmodifier represents which

modifier is pressed, x and y represents the co-od position of the mouse, click represents the clicks and popup is a flag indicating whether the popup appeared on this platform or not appeared.

The most commonly used methods are given below.

- int getX( )
- int getY( )
- Point getPoint( )
- int getButton( )
- int getClickCount( )
- void translatePoint( )
- boolena isPopupTrigger( )

4.10 Sources of Events:

- Button → Generates ActionEvent when the button is pressed.
- Checkbox → Generates an ItemEvent when the Check box is selected or deselected.
- Window → Generates window events when a window is activated, deactivated, opened, closed or quit
- List → Generates ActionEvents when an item is double clicked
- MenuItem → Generates an ActionEvent when a menu item is selected
- Scrollbar → Generates an AdjustmentEvent when the scrollbar is manipulated.
- Text Components → Generates TextEvent when a character is entered in a TextArea or a TextField.

4.11 Event Listener Interfaces:

Some of the event listener interfaces and its methods are given below.

(i) The ActionListener Interface: This interface defines the actionPerformed( ) method that is called when an action event occurs.

- void actionPerformed(ActionEvent ae)

(ii) The ItemListener Interface: This interface defines the itemStateChanged( ) method that is called when an item event occurs.

- void itemstateChanged(ItemEvent ie)

(iii) The KeyListener Interface: This interface called any one of the following three methods when a key event occurs.

- void keyPressed(KeyEvent ke);
- void keyReleased(KeyEvent ke);
- void keyTyped(KeyEvent ke);

(iv) The MouseListener Interface: This interface called any one of the following five methods when a mouse is pressed, released or quit.

- void mouseClicked(MouseEvenet me);
- void mouseEntered(MouseEvenet me);
- void mousePressed(MouseEvenet me);
- void mouseReleased(MouseEvenet me);
- void mouseExited(MouseEvenet me);

(v) The MouseMotionListener Interface: This interface defines the following two methods. The mouseDragged( ) method is called whenever the mouse id

dragged. The mouseMoved( ) method is called whenever the mouse is moved.

- void mouseDragged(MouseEvenet me);
- void mouseMoved(MouseEvent me);

(vi) The TextListener Interface: This interface defines the textChanged( ) method that is invoked when achange occurs in the TextField or textArea.

- Void textChanged(TextEvent te);

(vii) The adjustmentListener Interface: This interface defines the adjustmentValueChanged( ) method that is invoked when an adjustment event occurs.

- void adjustmentValueChanged(AdjustmentEvent ae);

4.12 Handling Mouse Events:

The mouse events are handled by the interfaces MouseListener and the MouseMotionListener. The MouseListener interface methods and the MouseMotionListener interface methods are used to handle the events. The mouse events are generated when we press or release the mouse, when the mouse is dragged or moved, when the mouse entered or exited and when the mouse is clicked.

Example: Java program to demonstrate the Mouse Events.

```
import java.awt.*;

import java.awt.event.*;

import java.applet.Applet;

/* <applet code="Demomouse" width=300

    height=300 > </applet> */

public class Demomouse extends Applet implements MouseListener  {
```

```
String msg;
int x=0;
int y=0;

public void init( ) {
  addMouseListener(this);
}
 public void mouseClicked(MouseEvent e)   {
  x=200;
  y=50;
   msg="Mouse Clicked";
repaint( );
 }
public void mouseEntered(MouseEvent e)   {
  x=200;
  y=50;
   msg="Mouse Entered";
repaint( );
 }
public void mouseExited(MouseEvent e)   {
  x=200;
```

```
  y=50;
   msg="Mouse Exited";
repaint( );
 }
public void mousePressed(MouseEvent e)   {
  x=e.getX( );
  y=e.getY( );
   msg="Mouse Pressed";
repaint( );
 }
public void mouseReleased(MouseEvent e)   {
  x=e.getX( );
  y=e.getY( );
   msg="Mouse Released";
repaint( );
 }
public void paint(Graphics g) {
g.drawString(msg,x,y);
}
}
```

4.13 Handling Keyboard Events:

The Keyboard events are handled by the interface KeyListener. The KeyListener interface methods are used to handle the events. Key events are generated when the key is pressed, released or typed.

Example: Java program to demonstrate the Keyborad Events.

```
import java.awt.*;

import java.awt.event.*;

import java.applet.Applet;

/* <applet code="Demokey" width=300

    height=300 > </applet> */

public class Demokey extends Applet implements KeyListener {

String msg;

int x=50;

int y=100;

public void init( ) {

  addKeyListener(this);

  requestFocus( );

}

 public void keyPressed(KeyEvent e)   {

  showStatus("Key Pressed");

   repaint( );

 }
```

```
public void keyReleased(KeyEvent e)   {
  showStatus("Key Released ");
  repaint( );
 }
public void keyTyped(KeyEvent e)   {
  msg+=e.getKeyChar( );
  repaint( );
 }
public void paint(Graphics g) {
g.drawString(msg,x,y);
}
}
```

Unit V

GRAPHICS

5.1 Graphics Class:

The graphics class defines several drawing methods. All graphics are drawn relative to window. The origin of each window is top left corner and is 0,0. The co-ordinates are specified in pixels. The co-ordinate position (20, 50) points the column number 20 and row number 50. The graphics context is encapsulated by the following two methods.

- By calling paint( ) or update( ) method.
- Returned by the getGraphics( ) method.

5.1.1 Drawing Lines:

The drawLine( ) method can be used to draw lines. It displays the line in the current drawing color that begins at x1, y1 and ends at x2, y2. The general form of the drawLine( ) and an example is given below.

- void drawLine(int x1, int y1);

Example:

_ _ _ _ _ _ _ _

public void paint(Graphics g) {

g.drawLine(30,20, 200,300);

_ _ _ _ _ _

}

5.1.2 Drawing Rectangles:

The methods used to draw a rectangle and fill the rectangle are given below.

- void drawRect(int top, int left, int length, int breadth);
- void fillRect(int top, int left, int length, int breadth);

Here top and left indicates the top-left corner of the rectangle, the length represents the length of a rectangle and breadth represents the breadth of a rectangle. The fillRect( ) method is used to fill the rectangle.

Example:

_ _ _ _ _ _ _ _

public void paint(Graphics g) {

g.drawRect(100,100, 200,300);

g.fillRect(100,100, 200,300);

_ _ _ _ _ _

}

5.1.3 Drawing Ellipses and Circles:

The drawOval( ) method can be used to draw ellipses and circles. In the given below method if the major and minor are different then an ellipse is drawn, if the both the major and minor are same then a circle is drawn. The fillOval( ) method can be used to fill the ellipse and circles.

- void drawOval(int top, int left, int major, int minor);
- void fillOval(int top, int left, int major, int minor);

Example:

_ _ _ _ _ _ _ _

public void paint(Graphics g) {

g.drawOval(100,100, 200,300); //Ellipse

g.fillOval(100,100, 200,300); //Ellipse

g.drawOval(150,150, **50,50); //Circle**

g.fillOval(150,150, **50,50); //Circle**

_ _ _ _ _ _

}

Example: Java program to demonstrate the Graphics methods.

```
import java.awt.*;
import java.applet.*;
/* <applet code="Demograph" width=300 height=200>
                        </applet> */
public class Demograph extends Applet
 {
 public void paint(Graphics g)
 {
g.setColor(Color.green);
g.drawArc(150,40,20,60,180,360);
g.drawArc(150,40,20,60,90,180);
g.fillArc(150,40,20,60,0,360);
g.setColor(Color.blue);
g.drawArc(100,75,120,25,0,-180);
g.setColor(Color.red);
```

```
g.drawRect(100,100,120,50);

g.fillRect(100,100,120,50);

}

}
```

5.2 Color Class:

In java color is encapsulated by Color class. The AWT color system allows you specify any color you want. Color class defines some of the standard colors as color constant. The user can create any color using the constructors given below.

- Color(int red, int green, int blue);
- Color(float red, float green, float blue);

In the first constructor red, green, blue colors are represented as integer values. The value of the integer value must be between 0 and 255. The second constructor is used if you want to specify the color values in floating point number. In this case the value ranging between 0 and 1.0.

Example:

- Color b = new Color(0, 0, 255);
- Color r = new Color(255, 0, 0);
- Color g = new Color(0.0,1.0, 0,0);
- Color c = new Color(50,75,123);

In the above example „b" is a Blue color, „r" is a Red color, „g" is a Green color and „c" is a new color with the combination of Red, Green and Blue.

Some of the color class methods are given below.

- int getRed( ) → Returns the Red component of the Color.

- int getGreen( ) → Returns the Green component of the Color.
- int getBlue( ) → Returns the Blue component of the Color.
- void setColor(Color new-color) → Used to set or change the Color.
- Color getColor( ) → Returns the current Color.

Example:

- Color b1 = new Color(0, 0, 255);

  int rr = b1.getRed( ); // Returns 0 to rr.

  int gg = b1.getGreen( ); //Returns 0 to gg.

  int bb = b1.getBlue( ); // Returns 255 to bb
- Color c2 = new Color(200,50,125);

  g.setColor(c2);

Example: Java program to demonstrate Color class methods.

```
import java.io.*;
import java.awt.*;
import java.applet.*;
/* <applet code="Colorprg" height=300 width=300> </applet> */
public class Colorprg extends Applet  {
Font f=new Font("Arial",Font.BOLD,14);
Color c=new Color(200,20,167);
String msgn=f.getName();
int r=c.getRed();
```

```
int b=c.getBlue();

int g=c.getGreen();

String msg1=Integer.toString(r);

String msg2=Integer.toString(b);

public void paint(Graphics g)   {

g.setFont(f);

g.setColor(c);

g.drawString("Welcome",50,50);

g.drawString(msg1,100,100);

g.drawString(msg2,100,150);

g.setColor(Color.green);

g.drawOval(100,100,50,30);

  }

 }
```

5.3 Font Class:

The Font object can be constructed using the constructor. The following attributes/variables are defined by the Font class. The general form of the Font constructor is given below.

- String fontname → Specifies the name of the Font.
- int fontsize → Specifies the Font size in integer.
- float fontsize → Specifies the Font size in floating point number.
- int fontstyle → Specifies the Font style.

Constructor:

- Font(String fontname, int fontstyle, int fontsize);

Example:

- Font f1=new Font(“Arial”, Font.BOLD, 14);

Here f1 is the Font object with font name “Arial”, font style BOLD and font size 14 points.

Methods:

- setFont(Font fontobject); → Used to the current font.
- Font getFont( ); //Returns the current Font.

Example: setFont(f1); // f1 is the Font object.

Font f2;

F2=f1.getFont( ); // Returns the variables of f1 to f2.

Example: Java program to demonstrate Font class methods.

```
import java.io.*;
import java.awt.*;
import java.applet.*;
/* <applet code="Demofont" height=300 width=300> </applet> */
public class Demofont extends Applet  {
Font f=new Font("Arial",Font.BOLD,14);
Color c=new Color(200,20,167);
Color c1=new Color(100,220,10);
```

```
String msgn=f.getName();

String msg1=f.toString();

public void paint(Graphics g)   {

  g.setFont(f);

g.setColor(c);

g.drawString("Welcome",50,50);

g.setColor(c1);

g.drawString(msg1,100,100);

g.drawOval(100,100,50,30);

  }

 }
```

AWT CONTROLS

5.4 Control Fundamentals:

The Abstract window toolkit (AWT) controls are the components that allow the user to interact with your application in several ways. Some of the commonly used AWT controls are Labels, Push Buttons, Check Box, TextArea and TextField. The layout manager automatically arrange the components in an Applet or a Window. The add() can be used to add the control to a window. The remove( ) method is used to remove the control from the window.

- Component add(Component object);
- void remove(Component object);

Here the object represents the reference to the control.

Example:

- Button b1 = new Button(“PASS”); //Creates Button object b1

  add(b1); //To add the Button object b1

  remove(b1); // To remove the Button object b1

5.5 Label:

A label is an object of type Label. It contains the String to be displayed. Label is a passive control. The constructors and methods of a Label are given below.

Constructors:

- Label( ); → Default constructor of the Label, it creates label.
- Label(String name); → Creates a label object along with its name.
- Label(String name, int pos) → Creates a label object along with its name and pos shows the alignment of the label. The pos can take any one of these three values ie Label.LEFT, Label.RIGHT, Label.CENTER.

Methods:

- void setText(String name); // Used to change the existing text or to set text for a label.
- String getText( ); // Returns the name of the current lable.
- void setAlignment(int pos); // used to set alignment of the name within label .
- int getAlignment( ); // Returns the position/alignment of the current label.

Example:

Label l1 = new Label(“DDCE”); // Creates a label DDCE

Label l2 = new Label(“M S U”,Label.CENTER);//Creates a Label M S U at the center.

String name = l1.getText( );//Returns the string DDCE to name.

int align=l2.getAlignment( );// returns the alignment value to align.

5.6 Button:

A Button is a component that contains a label. Button is the most commonly used control. It generates an event when it is pressed. The constructors and methods of the Button class are given below.

Constructors:

- Button( ); // Default constructor to create a button.
- Button(String name); // Creates a button with its name.

Methods:

- void setLabel(String name) ; // Used to set/change the button name.
- String getLabel( ); // Returns the name of the button.

Handling Buttons:

When a button is pressed an event is generated. The registered listener receives the event notification. The actionPerformed( ) method of the ActionListener interface is called when an event occurs. ActionEvent object is passed as argument.

Example: Java program to handle button events.

```
import java.awt.*;

import java.awt.event.*;

import java.applet.Applet;

/* <applet code="Demobutton1" width=300

    height=300 > </applet> */
```

```
public class Demobutton1 extends Applet implements ActionListener   {
Button b1,b2;
String msg;
public void init( ) {
  b1=new Button("ADD");
  b2= new Button("SUB");
  add(b1);
  add(b2);
  b1.addActionListener(this);
  b2.addActionListener(this);
}
public void actionPerformed(ActionEvent ae)   {
  String s;
   s=ae.getActionCommand( );
     if(s.equals("ADD"))
     msg="ADDITION";
  else if(s.equals("SUB"))
     msg="SUBTRACTION";
       else msg="ERROR";
repaint( );
 }
```

```
public void paint(Graphics g) {
 g.drawString(msg,50,100);
 }
}
```

5.7 Checkbox:

The Checkbox is a AWT control that is used to ON or OFF (select or deselect) an option. Checkbox is a small box with check mark if it is in ON state otherwise no check mark on it. We can add a label to the checkbox to show its purpose. The state of the check box is changed by clicking on it. The constructors and methods of the Checkbox control are given below.

Constructors:

- Checkbox( );// Default constructor to create a check box
- Checkbox(String name); // Creates a Check box control with the name
- Checkbox(String name, boolean on);// Creates a check box with the name and is set to ON state
- Checkbox(String name, Boolean on, CheckboxGroup cbgp);// Creates a check box with the name with ON state and is added to the check box group cbgp.

Methods:

- void setState(Boolean on);// Used to set the state of a Checkbox control.
- boolean getState( ); // Returns the current state of the Checkbox
- void setLabel(String label);// Used to set the label for the Checkbox
- String getLabel( ); // Returns the label of the check box control.

Handling Check Boxes:

An item event is generated when a check box is selected or deselected. The registered listener receives the event notification. The itemStateChanged( ) method of the ItemListener interface is called when an event occurs. ItemEvent object is passed as argument.

Example: Java program to handle check box events.

```
import java.awt.*;

import java.awt.event.*;

import java.applet.Applet;

/* <applet code="Democbox" width=300

    height=300 > </applet> */

public class Democbox extends Applet implements ItemListener {

Checkbox cb1,cb2;

String msg;

public void init( ) {

  cb1=new  Checkbox("CBOX1");

  cb2= new Checkbox("CBOX2");

  add(cb1);

  add(cb2);

  cb1.addItemListener(this);

  cb2.addItemListener(this);

}
```

```
public void itemStateChanged(ItemEvent ie)   {
   if(ie.getSource()==cb1)
   msg="CHECKBOX1";
  else if(ie.getSource()==cb2)
      msg="CHECKBOX2";
       else msg="ERROR";
  repaint();
 }
public void paint(Graphics g)   {
 g.drawString(msg,150,100);
 }
}
```

5.8 CheckboxGroup:

CheckboxGroup is an AWT control with more than one check box. In a check box group at a particular time only one check box is selected and all other check boxes are in deselected state. The check boxes in the check box group are also called radio buttons. The constructors and methods of CheckboxGroup are given below.

Constructor:

- CheckboxGroup cbg = new CheckboxGroup( ); //Default constructor to create a CheckboxGroup

Methods:

- void add(Checkbox);// To add a check box to the CheckboxGroup

- Checkbox getSelectedCheckbox( );// Returns the selected Checkbox
- void setSelectedCheckbox(Checkbox cb);//cb is the check box to be set and previously selected check box will be deselected

Example:

```
CheckboxGroup cbg1 = new CheckboxGroup( );

Checkbox cb1 = new Checkbox("+",cbg1,true);

Checkbox cb2 = new Checkbox("-",cbg1,false);

Checkbox cb3;

add(cb1);

add(cb2);

cb3=cbg1.getSelectedCheckbox( );

cbg1.setSelectedCheckbox(cb2);
```

Example: Java program to handle check box group events.

```
// Democbox using drawstring method

import java.awt.*;

import java.awt.event.*;

import java.applet.*;

/* <applet code="Democbgp" width=300

    height=300 > </applet> */

public class Democbgp extends Applet implements ItemListener {

CheckboxGroup cbg;
```

```
Checkbox cb1,cb2;
String msg;
Color c1,c2;
Font f;
public void init( )  {
  cbg= new CheckboxGroup( );
  cb1=new  Checkbox("CBOX1",false,cbg);
  cb2= new Checkbox("CBOX2",false,cbg);
  add(cb1);
  add(cb2);
  cb1.addItemListener(this);
  cb2.addItemListener(this);
  c1=new Color(255,0,0);
  c2=new Color(0,255,0);
  f=new Font("Arial",Font.BOLD,20);
}
public void itemStateChanged(ItemEvent ie) {
  String s;
   if(ie.getItemSelectable()==cb1)
     msg="CHECKBOX1";
  else if(ie.getItemSelectable()==cb2)
```

```
        msg="CHECKBOX2";
      else msg="ERROR";
repaint( );
 }
public void paint(Graphics g)   {
 if(msg=="CHECKBOX1")
  setBackground(c1);
 else
 setBackground(c2);
 g.setFont(f);
 g.drawString(msg,50,100);
 }
}
```

5.9 TextField:

A single line text entry in java is implemented through TextField class. It is also called edit control. TextField is a sub class of TextComponent. The constructors and methods of TextField class are given below.

Constructor:

- TextField( ); //Default constructor to create a TextField
- TextField(int characters);// Creates a TextField with a specified number of character width.
- TextField(String name);// Creates a TextField with a string name.

Methods:

- String getText( );// Returns the current string in the TextField.
- void setText(String name); //Used to set the text in the name.
- void select(int start, int end);//Used to select the characters in a TextField beginning at start and ending at end.
- String getSelectedText( );// Returns the selected text in the TextField.

Example:

TextField tf = newTextField(20);//Creates a TextField with 20 characters

tf.setText("M S University");//The text M S University is displayed in the TextField.

String name =tf.getText( );// Returns the text M S University to the string name.

Example: Java program to handle the TextField.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
/* <applet code="Democbox1" width=300
    height=300 > </applet> */
public class Democbox1 extends Applet implements ItemListener {
Checkbox cb1,cb2;
TextField tf;
String msg;
public void init( )  {
```

```
  cb1=new  Checkbox("CBOX1");
  cb2= new Checkbox("CBOX2");
  add(cb1);
  add(cb2);
  tf=new TextField(20);
add(tf);
cb1.addItemListener(this);
cb2.addItemListener(this);
repaint( );
}
public void itemStateChanged(ItemEvent ie)  {
  String s;
   if(ie.getItemSelectable()==cb1)
     msg="CHECKBOX1";
  else if(ie.getItemSelectable()==cb2)
         msg="CHECKBOX2";
      else msg="ERROR";
tf.setText(msg);
 }
}
```

5.10.TextArea:

In java, TextArea provides a multi-line text editor. TextArea is a sub class of TextComponent. The constructors and methods of TextArea class are given below.

Constructors:

- TextArea( ); //Default constructor to create a TextArea
- TextArea(int lines, int characters);// Creates a TextArea with a specified number of lines and each line is of characters width.
- TextArea(String name);// Creates a TextArea with a initial string name.
- TextArea(String name, int lines, int characters);// Creates a TextArea with a initial text name and specified number of lines and each line is of characters width.

Methods:

- void append(String msg);//Used to append the string specified in the msg.
- void insert(String msg, int location);//Used to insert the string msg in the specified location.
- void replaceRange(String msg, int start, int end);// Used to replace the existing string with the new string msg in the specified starting index start and the ending index end.

Example:

TextArea ta = newTextArea(10,20);//Creates a TextArea with 10 lines and each line with 20 characters

ta.insert("M S University, Tirunelveli-627012, Tamil Nadu",5);//The text M S University, Tirunelveli-627012, Tamil Nadu is inserted in the line number 5.

ta.append("DDCE");// Appends the string DDCE to the existing message.

Example: Java program to handle the TextArea.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
/* <applet code="Demotaea" width=300
    height=300 > </applet> */
public class Demotaea extends Applet implements ItemListener
 {
Checkbox cb1,cb2;
TextArea ta;
String msg1,msg2;
public void init( )
 {
cb1=new  Checkbox("CBOX1");
cb2= new Checkbox("CBOX2");
add(cb1);
add(cb2);
ta=new TextArea(10,20);
Label l1=new Label("DEMOTAREA");
add(ta);
add(l1);
```

```
cb1.addItemListener(this);
cb2.addItemListener(this);
}
public void itemStateChanged(ItemEvent ie)   {
  String s;
   if((ie.getItemSelectable()==cb1) && (cb1.getState()==true))
    {
     msg1="CHECKBOX1            11111111   ";
    ta.insert(msg1,3);
    }
   else if((ie.getItemSelectable()==cb1) && (cb1.getState()==false))
   ta.setText("CB1 Disabled");
  if((ie.getSource()==cb2) && (cb2.getState()==true))
     {
    msg2="CHECKBOX2      222222     ";
     ta.insert(msg2,7);
     }
   else if((ie.getSource()==cb2) && (cb2.getState()==false))
  ta.setText("Cb2 Disabled");
}
}
```

5.11 Layout Managers:

In java a layout manager automatically arranges the controls used in a program. Each Container object has a layout associated with it. A layout manager is an instance of any class that implements the LayoutManager interface. The setLayout() method can be used to set layout. Java supports the following layout managers.

- FlowLayout
- BorderLayout
- GridLayout
- CardLayout

5.11.1 FlowLayout:

FlowLayout is the default layout manager. FlowLayout implements a simple layout style. In FlowLayout the controls are arranged from top-left corner left to right and top to bottom. A small space is left between two components, above and below two components. The constructors and methods of a FlowLayout manager are given below.

Constructors:

- FlowLayout( );//Default constructor to create a layout.
- FlowLayout(int alignment);//Used to create a flow layout with a specific order.
- FlowLayout(int alignment, int horizontal, int vertical);//Used to create a flow layout with a specific order and with horizontal and vertical space between the components. The alignment can take any one of the integer value ie FlowLayout.LEFT, FlowLayout.RIGHT, FlowLayout.CENTER.

Methods:

- void setLayout(LayoutManager obj);// Used to set layout manager.

Example:

setLayout(new FlowLayout(FlowLayout.CENTER));// Creates flow layout with the components are center justified.

MODEL QUESTION

Part – A

Answer ALL Questions. **(10 X 2=20)**

1. What are the basic concepts are of object oriented programming?
2. What is a Constructor"
3. What is the difference between static and non-static member in java?
4. What are the uses of the keyword super?
5. What is the purpose of package in java?
6. What is an Exception?
7. What are the Applet display methods?.
8. What is a Delegation Event model?
9. Write the constructors of Label.
10. Write any Four AWT controls.

Part – B

Answer ALL Questions. **(5 X 5=25)**

11.(a) Explain the Primary Data types in java with example.

(OR)

(b) Explain various types of Literals in java.

12.(a) Explain method overriding with an example.

(OR)

(b) Explain command line argument with an example.

13.(a) How will you create a Thread? Explain with an example.

(OR)

(b) How will you create a package? Explain with an example.

14.(a) Explain the Applet architecture.

(OR)

(b) Explain the various Event classes.

15.(a) Explain any Five Graphics class methods with example.

(OR)

(b) Explain the constructors and methods of Buttons with example.

Part – C

Answer any THREE Questions. (3 X 10=30)

16. Explain the Operators in java with example.
17. Write a java program to calculate the area of a circle, rectangle and triangle by overload the method area( ).
18. Write a java program to create an Interface and implement the Interface.
19. Explain Mouse events with a program.
20. Write a java program to display " Manonmaniam Sundaranar University" in three different fonts with different colors in an Applet.

Dr D.S.Mahendran M.Sc., M.Phil., Ph.D.,PBDCSA
Associate Professor
Department of Computer Science
Aditanar College of Arts & Science
Virapandianpatnam – 628 216
Tiruchendur