UNIT-1

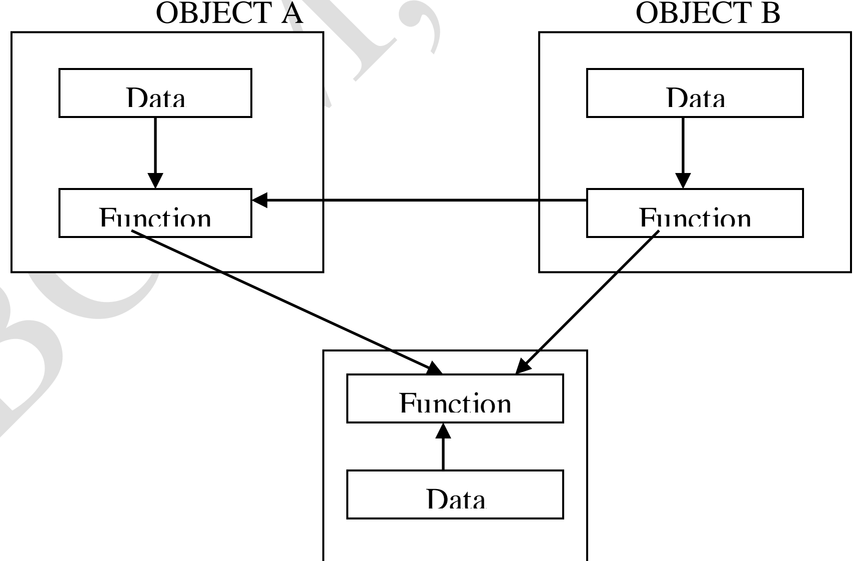
2 Marks:

1. **Define Object – Oriented Program.**

- ✓ **"***Object-oriented programming is an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copied of such modules on demand."*.
- ✓ The data of an object can be accessed only by the functions associated with that object. Any Problem is viewed as a system consisting of several objects.
- ✓ The fundamental idea behind OOP is to combine both data and functions that operate on that data in to a single unit. Such a unit is called an **object.**

2. **Write the Features of OOP.**

- ➢ Emphasis is on data rather than procedure.
- ➢ Programs are divided into as objects.
- ➢ Data structures are designed such that they character the objects.
- ➢ Functions that operate on the data of an object are tied together in the data structure.
- ➢ Data In hidden and cannot be accessed by the external function.
- ➢ Object may communicate with each other through function.
- ➢ New data and functions can be easily added whenever necessary.
- ➢ Follows bottom up approach in program design.



BCA,IT DEPT

3. What are the principle advantage of OOPs?

- Through inheritance, we can eliminate redundant code and extend the use of existing classes.
- We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch .This leads to saving of development time and higher productivity.
- The principle of data hiding helps the programmer to build secure programs that Cannot be invested by code another parts of the programs.
- It is easy to partition the work in a project based on objects.
- The Data centered design approach enables us to capture more details of a model.

4. What are the application of OOPs?

The Areas for application of OOP includes

- Real-time system.
- Simulation and modeling
- Object-oriented data base.
- Hypertext, hypermedia and expertext
- AI and expert system
- Neural networks and parallel programming
- Decision support and office automation system.
- CIM/CAD/CAM. System.

5. Define this Keyword

The **this Keyword** refers to the class in which it is used.
When we used this keyword in the Demo class, it will refer to the Demo class. The this keyword is often used in the Constructor Method.

6..What is meant by Garbage Collection?

- Objects are allocated by using the new Operator.
- The objects are destroyed and their memory released for later reallocation.
- It handles deallocation for you automatically. This is called garbage collection.

7..Define Finalize( ) Method.

- Java Run – Time calls that method, whenever it is an object of that class.
- This method will specify those actions that must be performed before an object is destroyed.
- It is important to understand the method is only called just prior to garbage collection.

BCSM,IT DEPT

8.What is meant by Method OverLoading?

✓ With in the class the method name is same, the parameter is different.

Eg:

```
class over
{
 void disp()
   {
    System.out.println("Method Overloading");
   }
 void disp(int a)
  {
    System.out.println("Method Overloading a = " +a);
   }
 void disp(int a,int b)
   {
    System.out.println("Method Overloading a & b = " +a " " +b);
   }
}
```

---

5 Marks:

1.Explain in detail about types of OOPS.

It will be classified into three types. They are

1. Encapsulation
2. Inheritance
3. Polymorphism.

1. Encapsulation

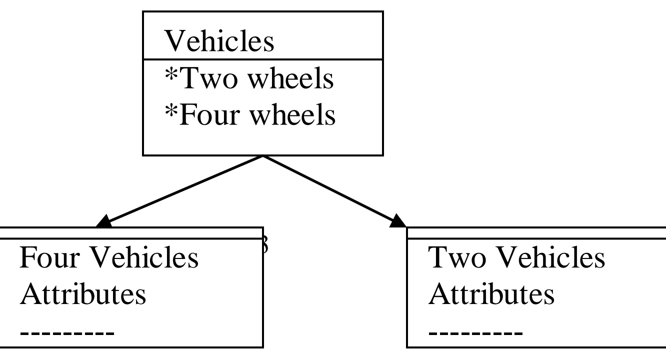
It is the way of combining both data and functions that operate on that data under a single unit is known as encapsulation.

The only way to access the data is provided by the functions, which are wrapped in the class. Is not accessible to the outside world. This function provided the interface between the objects data and program. This insulation of the data from direct access by the program is called data hiding.

2. Inheritance

Inheritance is the process by which object of one class to acquire properties of objects of another class.

The new class will have the combined feature of both the classes. The new class is know as **derived class or sub class.** The existing class is know as **base class**.



BCSM,IT DEPT

For the example, the vehicle is called as base class and the four vehicles and two vehicles is called as derived class.

In oop, the concept of inheritance provides the idea of reusability. This means that we can add additional feature to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined features of both the classes.

3. Polymorphism

Polymorphism is the ability for a message or data to be processed in more than one from. *That is the ability of a function or operator to act indifferent ways on different data types is called polymorphism*.

For example : consider the operation of addition .for two numbers, the operator will generate a sum. if the operands are string. then the operation would produce a third string by concatenation.

---

2. Explain the followings.

Data Types

Data is differentiated into various Types. The type of data element restricts the data element to assume a particular set of values.

Type	Size (bits)	Size(bytes)	Range
Byte	8 bits	1 byte	-127 to 127
Short	16 bits	2 bytes	-32,768 to 32,767
Int	32 bits	4 bytes	-2,147,483,648 to +2,147,483,647
Long	64 bits	8 bytes	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
Float	32 bits	4 bytes	3.4e-038 to 3.4 e+0.38
Double	64 bits	8 bytes	1.7e-308 to 1.7e+308
Char	16 bits	2 bytes	0 to 65,536

Variable:

- Variables are the basic unit of storage in a program. It is treated as an identifier to the memory location.
- The Value will be change during the execution of the Program.

- **Declaration:** type identifier; (Or) type identifier = value;
  Where type specifies the data type and identifier specify the name of the variable.
- **Example:** int a, b; float f ,s;

BCSM,IT DEPT

Constant:

- Referred to the fixed value, that does not change during the execution of a program.
- The value will not be change during the execution of the Program.
- Symbolic Constant: Any value declared as 'const' can't be modified by the program.
- Example: const float Pi = 3.1413;
- Literal Constant:

  Integer Ex: int A = 123;
  Floating point Ex: float F = 12.56;
  Characters Ex: char Ch = 'S';
  String Ex: char Ch [20] = "WELCOME";

Array:

- Array is a group of same type values that are referred by a common name.
- **Declaration:** datatypes varname[size];
- Where type specifies the data type of the array, var-name specify the name of the array and size specify the number of elements stored in the array.
- **Example:** 1. int mark [20];
  2. int mark[ ] ={ 78,29,34,87........};

3.Explain the Jump Statements.

It transfers the control to another part of our program. It will be classified into three types. They are

1. Break
2. Continue
3. Return

Break:

- The break statement has three uses.
- 1. Terminates a statement sequence in a statement.
- 2. It can be used to exist from a loop.
- 3. It can be used as a civilized form of 'goto' statement.

Uses:

1. It can be used to exit a loop.
2. It terminates a statement sequence in a switch statement.

Continue:

In 'while' and 'do- while' loops a 'continue' statement transfer the control directly to the conditional expression. But in a 'for' loop control first go to the iteration part and then go to the conditional expression.

BCSM, IT DEPT

Return:

A return statement is used to transfer the control back to the caller of the method. Used to return a value to the calling method. A return statement immediately terminates the method is which it is executed.

---

10 Marks:

1.Enumerate the concept of Control Statements

Selection Statements:

It allow you to control the flow of your program's execution based upon some conditions.

It will be classified into two types.

1. If Statement

Syntax:

```
if (condition)
      Statement 1;
else
      Statement 2;
```

Value return by the condition is either true or false. If the condition is true, then statement1 is executed. Otherwise statment2 is executed. If the statement is single, then parenthesis ({ }) is not necessary.

Compound statements must be enclosed within the parenthesis.

BCSM, IT DEPT

- **Example:** int A=300 , b=100;

```
if(A>B)
    System.out.println(" value of A is big ");
else
    System.out.println (" value of B is big ");
```

Nested if statement: One if statement contain another if statement.

Syntax:

```
if (conditon1)
      {
         if (condition 2)
                statement1;
         else
                statement2;
      }
else
   Statement 3;
```

If condition1 is true, then condition2 is executed. If condition2 is true then statement1 is executed. Otherwise statement2 is executed. If conditio1 is false, then statement3 is executed.

Example:
```
int A = 30, B = 40, C = 20;
if ( A>B)
  {  if(A>C)
        System.out.println ("Value of A is big");
     else
        System.out.println ("Value of C is big");
  }
else
   System.out.println(”Value of B is big");
```

Switch Statement:

Syntax
```
switch ( expression )
  {
   case value1:   //statements
                  break;
   case value2:   //statements
                  break;
   case value3:   //statements
                  break;
    ....   ........   .....
   default:       //statements
                  break;
  }
              Statement –n;
```

- ❖ Here the expression must be an Integer Expression or Character Expression.
- ❖ Where value1, value 2 ….are integer constant (or) constant expression (or) character constant and each of the value with in the switch statement.
- ❖ The Stat-1,Stat-2 are statement list and it make have more than one statements and there is no need to put open and close brackets.
- ❖ The break statement at the end of each block indicates the end of particular case and causes an exit from the switch statement and transferring the program control to Statement – n following the switch statement.
- ❖ The default is an optional case when it presents it will be executed if the value of the expression does not match with any of the case values.
- ❖ If not present no action lakes place if all matches fail & the control goes to the Statement – n.

Iteration Statement:

BCS DEPT

It create what we commonly called loops. A loop repeatedly executes the same set of instructions until a termination condition is met. It will be classified into three types.

1.For ..Loop
2.While .. Loop
3.do .. while Loop.

While Loop:

Syntax:

```
while (condition )
    {
      .....    ......   ......
      .....    ......   ......    // body of the
                                     loop
      }
```

The set of statements will be executed as long as the value of condition is true. The condition can be any Boolean expression. When the condition becomes false, then the control goes to the next statement followed by the loop. If the condition is initially false, then the body of the loop will not be executed at all.

Do - While Loop:

Syntax:

```
do
   {
        .....    .......
       .......  .......   // body of the loop
   } while (condition);
```

The do-while statement always executes the body of the loop, at least one's. Because it's conditional expression is evaluated at the bottom of the loop. Every time do - while loop first execute the body of the loop and then evaluate the conditional expression. If the condition is true, then the loop will be repeated. Otherwise the loop will be terminated.

for Loop:

Syntax:

```
for (initialization; condition; iteration)
   {
      ......    ........    ........  // body of the loop
      ......    ........    ........
   }
```

- The initialization part executed first and only one time (beginning of the loop) and set the initial value of the control variable.

BCS IT DEPT

- The condition part will be executed second. This must be a Boolean expression. If the condition is true, then the body of the loop will be executed. Otherwise the loop will be terminated.
- The iteration part is an expression, that increment or decrement the loop control variable.

2.Discuss about The Vector Class.

- . Vector class provides the capability to implement a growable array.
- The array grows larger as more elements are added to it.
- The array may also be reduced in size after some of its elements have been deleted.
- This is accomplished using the trim**ToSize()** specify only the initial storage capacity.
- A third, default constructor specifies neither the initial capacity nor the capacityIncrement.
- This constructor lets Java figure out the best parameters to use for Vector objects.
- Finally, a fourth constructor was added with JDK 1.2 to create a Vector out of a Collection object.
- The access methods provided by the Vector class support array-like operations and operations related to the size of Vector objects.
- The array-like operations allow elements to be added, deleted, and inserted into vectors.
- They also allow tests to be performed on the contents of vectors and specific elements to be retrieved.
- The size-related operations allow the byte size and number of elements of the vector to be determined, and the vector size to be increased to a certain capacity or trimmed to the minimum capacity needed.
- Consult the Vector API page for a complete description of these methods.

THE SOURCE CODE OF THE VectorApp PROGRAM.

```
import java.lang.System;
import java.util.Vector;
import java.util.Enumeration;
public class VectorApp {
 public static void main(String args[]){
  Vector v = new Vector();
  v.addElement("one");
  v.addElement("two");
  v.addElement("three");
  v.insertElementAt("zero",0);
  v.insertElementAt("oops",3);
  v.insertElementAt("four",5);
  System.out.println("Size: "+v.size());
  Enumeration enum = v.elements();
```

BCSM IT DEPT

```
  while (enum.hasMoreElements())
   System.out.print(enum.nextElement()+" ");
  System.out.println();
  v.removeElement("oops");
  System.out.println("Size: "+v.size());
  for(int i=0;i<v.size();++i)
   System.out.print(v.elementAt(i)+" ");
  System.out.println();
 }
}
```

- The program creates a Vector object using the default constructor, and uses the **addElement()** method to add the strings "one", "two", and "three" to the vector.
- It then uses the **insertElementAt()** method to insert the strings "zero", "oops", and "four" at locations 0, 3, and 5 within the vector.
- The **size()** method is used to retrieve the vector size for display to the console window.
- The **elements()** method of the Vector class is used to retrieve an enumeration of the elements that were added to the vector.
- A while loop is then used to cycle through and print the elements contained in the enumeration.
- The **hasMoreElements()** method is used to determine whether the enumeration contains more elements. If it does, the **nextElement()** method is used to retrieve the object for printing.
- **The removeElement()** of the Vector class is used to remove the vector element containing the string "oops".
- The new size of the vector is displayed and the elements of the vector are redisplayed.
- The for loop indexes each element in the vector using the **elementAt()** method.

The output of the VectorApp program

```
Size: 6
zero one two oops three four
Size: 5
zero one two three four
```

---

3. Write down the features of Java.

- Sun describes Java as a "simple, object-oriented, interpreted, robust, secure, architecture-neutral, portable, high-performance, multithreaded, and dynamic language."

Simple

- Java is simple to use for three main reasons:
  - First, Java is familiar to you if you know C.
  - Second, Java eliminates components of C that cause bugs and memory leaks and replaces their functionality with more efficient solutions and automated tasks, so you have a lot less debugging.

BCSM,IT DEPT

- ⇨ Third, Java provides a powerful set of pre-tested class libraries that give you the ability to use their advanced features with just a few additional lines of code.

Object-Oriented

- ✓ Java is an object-oriented programming language that uses software objects called *classes* and is based upon reusable, extensible code.
- ✓ This means that you can use Java's classes, which are sets of variables and methods, as templates to create other classes with added functionality without rewriting the code from the parent classes or super classes.

Robust

- ✓ Java is robust because the language removes the use of pointers and the Java runtime system manages memory for you.
- ✓ The problems with pointers in C and C++ was that pointers directly addressed memory space.
- ✓ In a distributed environment like the Internet, when code is downloaded to diverse systems, there is no way of knowing for sure that memory space addressed by pointers is not occupied by the system.
- ✓ Overwriting this memory space could crash a system. Java also gives you automatic bounds checking for arrays, so they cannot index address space not allocated to the array.
- ✓ Automatic memory management is done using the **Garbage Collector.**

Interpreted

- ✓ Java is interpreted, so your development cycle is much faster.
- ✓ You need only to compile for a single, virtual machine and your code can run on any hardware platform that has the Java interpreter ported to it.

Secure

- ✓ Java is secure, so you can download Java programs from anywhere with confidence that they will not damage your system.
- ✓ Java provides extensive compile-time checking, followed by a second, multilayered level of runtime checking.

Architecture Neutral

- ✓ Java is architecture neutral, so your applications are portable across multiple platforms.
- ✓ Java's applications are written and compiled into bytecode for Java's virtual machine, which emulates an actual hardware chip.
- ✓ **Bytecode** is converted to binary machine code by the Java interpreter installed at the client, so applications need not be written for individual platforms and then ported from platform to platform.
- ✓ Java additionally ensures that your applications are the same on every platform by strictly defining the sizes of its basic data types and the behavior of its arithmetic operators.

BCSM, IT DEPT

- ✓ Operator overloading, the process of modifying the behavior of operators, is prohibited by Java.

High Performance

- ✓ Java is "high performance" because its bytecode is efficient and has multithreading built in for applications that need to perform multiple concurrent activities.
- ✓ Although threads still require the use of classes, Java balances the addition of thread synchronization between the language and class levels.
- ✓ Java's bytecode is efficient because it is compiled to an intermediate level that is near enough to native machine code that performance is not significantly sacrificed when the Java bytecode is run by the interpreter.

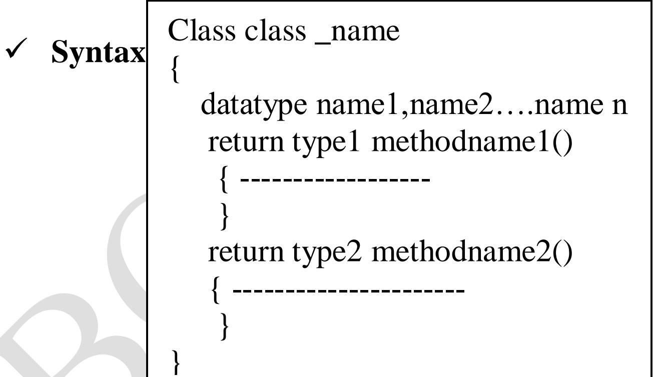
Dynamic

- ✓ Java is dynamic, so your applications are adaptable to changing environments because Java's architecture allows you to dynamically load classes at runtime from anywhere on the network, which means that you can add functionality to existing applications by simply linking in new classes.

4.Explain the Class Fundamentals:

- ✓ Class is the Collection of Object. Object is the collection of data and Member Function. (OR)
- ✓ Template for an object.
- ✓ The basic element of object-oriented programming is a class. A class define the shape and behavior of an object and is a template for multiple object.

- ✓ **Syntax**



```
Class class _name
{
   datatype name1,name2….name n
    return type1 methodname1()
     { ------------------
     }
    return type2 methodname2()
    { ----------------------
     }
}
```

Declaring Objects:

A class shall be viewed as a Template for an object. An object is a single instance of a class. Once we define a class, we can create any number of objects of its type.

Eg:

```
Demo D1 = new Demo();
```

Here the variable "D1" holds a reference of this object. Now we can create two new objects for the class name by using the following two statements.

Eg:

```
Demo D1 = new Demo();
Demo D2 = new Demo();
Demo D3 = D2.
```

Here both the object references D2, D3 will point the same object.

Assigning Object Reference Variable:

```
Demo D1 = new Demo();
Demo D2 =  D1;
```

Here D2 is being assigned a reference a copy of the object referred by D1.

Introducing Methods:

Syntax

```
type name (Parameter – list)
{
// body of the method
}
```

- ✓ Type specifies the type of data returned by the method.
- ✓ The method does not return a value of its return type must be void.
- ✓ The name of the method is specified by **name**
- ✓ Methods that have a return type other than void return a value to the calling routine using the following form of the return statement.
  return Value;

Eg:

```
class k1
{
 int a,b,c;
 k1(int a1,int b1,int c1)
  {
   a=a1;b=b1;c=c1;   }
 void disp()
  {
  System.out.println(a+b+c);
  }
 int disp1()
  {
    return((a+b)*c);
  }
}

class B
```

BCSM, IT DEPT

```
{
  public static void main(String args[])
   {
     k1 k = new k1(2,3,4);
     k.disp();
     k.disp1();
   }
}
```

5.Explain the following.

(A)Access Control:

Encapsulation links data with the code that manipulates it. It provide important attribute access control.

(B)Access Protection:

Accessible From	Public	Private	Protected
Own Class	Yes	Yes	Yes
Derived Class	Yes	No	Yes
Non-Derived Class	Yes	No	No

(C)Derived Type:

Private Public

Base class	Derived class	Base class	Derived class
Private	Not accessible	Private	Not accessible
Public	Private	Public	public

(D)Static Data Members

A static variable shall be declared by preceding its usual declaration with the static keyword.

Syntax

```
Static type variable name1;
```

Eg: Static int a

- We don't initialize the static variable, they will be initialized to default values.
- While Boolean type variables will be set false.
- The numeric type variables will be set to zero.
- Any variable name that acts as a reference to an object of a class will be set to null.
- We can use the keyword static we cannot create object.

BCSM,IT DEPT

Eg:

```
  class Kar
{
 static int a=10;
 public static void main(String args[])
  {
   System.out.println( "  a  = "+a);
  }
}
```

Output
 a = 10

(E)Final Keyword:

The variables declared as a final do not occupy memory. A final variable is essentially a constant.

Eg:

```
  class Kar2
{
 final int a=10;
 public static void main(String args[])
  {
   System.out.println( "  a  = "+a);
  }
}
```

Output
 a = 10

(F)Arrays:

A collection of such related data shall be represented as an array. All the values in an array are collectively represented by a single variable name.

An array is a collection of data storage locations, each of which holds the same type of data. Each storage location is called an element of the array.

You declare an array by writing the type, followed by the array name and the subscript. The subscript is the number of elements in the array, surrounded by square brackets.

BCSM,IT DEPT

Syntax:
Data type array name[size];

Eg: int a[5];
The statement provides the following information.
- The name of the array is **'a'**
- It contains integer constants
- It may contain a maximum of 5 values.

(G)Array of Objects:

- We can also have array of variables that are type class. Such variables are called array of objects.
- **General Format:** className ObjectName [size];
- **Example:** Student S [2];
- The identifier Student is a user defined data type. It can be used to create object. The statement Student S [2]; is an array of object. It contains two objects S [0] and S [1].

Note:

- The subscript of an array always begins with **zero(0)**
- It is an integer
- It cannot be a negative value
- The subscript should be enclosed with **[]**

******END******

BCSM,IT DEPT

UNIT -2

2 Marks:

1. What is inheritance?

- Inheritance is the process by which object of one class acquire the properties of objects of another class. It supports the concept of hierarchical classification.

2. What is recursion?

- Recursion is the process of defining something in terms of itself. In other words a method that calls itself is said to be recursive. The classic example of recursion is computation of the factorial of a number.

3. Define method overloading.

- Method overloading is done by more than one method within the same class, with the same name, but with a different parameter list. The overloaded methods have different return types. Overloading is determined at compile time.

4. Define method overriding.

- Method over riding occurs only when the names and the type signatures of the two methods are identical. If they are not, then the two methods are simply overloaded. Methods declared final cannot be overridden.
- Overriding methods may throw only the exceptions that the method that they override throws.

5. What are the differences between abstract class and an interface?

- Abstract classes may have instance variables, ordinary or concrete methods. It does not need all methods to be abstract.
- Interface can have variables which are static and the final, and methods, which are abstract only.
- An interface cannot implement any methods, whereas abstract class can.
- A class can implement many interfaces, but can only one super class

6.Define the keyword static

- When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object.
- Can only call other static methods
- Must only access static data
- Cannot refer to this or super anyway

7.Define the keywords super and this?

- A subclass can call a constructor method defined by its super class
- The second form of super acts somewhat like this, except that it always refers to the super class of the super class of the subclass in which it is used.

BCSM, IT DEPT

8.Explain Packages in java with example.

- In simple, it is a way of categorizing the classes and interfaces. When developing applications in Java, hundreds of classes and interfaces will be written, therefore categorizing these classes is a must as well as makes life much easier.
- **Import statements:**

  In Java if a fully qualified name, which includes the package and the class name, is given then the compiler can easily locate the source code or classes. Import statement is a way of giving the proper location for the compiler to find that particular class.

For example, the following line would ask compiler to load all the classes available in directory java_installation/java/io :

```
import java.io.*;
```

9. Define throws/throw Keywords:

- If a method does not handle a checked exception, the method must declare it using the **throws** keyword. The throws keyword appears at the end of a method's signature.
- You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the **throw** keyword.
- Try to understand the difference between throws and throw keywords, *throws* is used to postpone the handling of a checked exception and *throw* is used to invoke an exception explicitly.

10. What is meant by FINALLY block?

- The finally block follows a try block or a catch block. A finally block of code always executes, irrespective of occurrence of an Exception.
- Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.
- A finally block appears at the end of the catch blocks.
- **syntax:**

```
try
{
  //Protected code
}catch(ExceptionType1 e1)
{
  //Catch block
}catch(ExceptionType2 e2)
{
```

BCSM,IT DEPT

```
    //Catch block
  }catch(ExceptionType3 e3)
  {
    //Catch block
  }finally
  {
    //The finally block always executes.
  }
```

---

5 Marks:

1.Explain Method Over Loading.

With in the class the method name is same, the parameter is different.

Eg:

```
class over
{
  void disp()
   {
     System.out.println("Method Overloading");
   }
  void disp(int a)
  {
     System.out.println("Method Overloading a = " +a);
   }
  void disp(int a,int b)
   {
     System.out.println("Method Overloading a & b = " +a " " +b);
   }
 }

class Java
 {
   public static void main(String args[])
     {
      over k = new over();
      k.disp();
      k.disp(5);
      k.disp(5,10);
     }
 }
```

BCSM,IT DEPT

Output:

Method Overloading
Method Overloading a = 5
Method Overloading a & b = 5 10

2.Describe the Argument Passing:

Call by value:

This method copies the value of an argument into the formal parameter of the subroutine. Changes made to the parameter of the subroutine have no effect on the argument used to call it.

Eg:

```
class exam
{
 void disp(int i, int j)
  {
   i=i*2;
   j=j/2;
  }
}

class Exa
{
 public static void main(String arg[])
  {
   exam e=new exam();
   int a=10,b=15;
   e.disp(a,b);
   System.out.println("a = " +a + "b = " +b);
  }
}
```

- Here the operations that occur inside the disp() have no effect on the values of a and b.

Call By Reference:

- Reference use to transfer values by arguments
- Use to return more than one value to the calling function.
- Reduce the memory allocation.

3.Explain in detail Returning Objects:

A method can return any type of data, including class types we create.

```
class Exam
{
```

BCSM,IT DEPT

```
  int a;
  Exam(int i)  // Constructor
   {
     a=i;
    }
   Exam KK()
   {
    Exam E = new Exam(a+10);
    return E;
   }
}

class Test
{
 public static void main(String args[])
  {
   Exam e = new Exam(2);
   Exam e1;
   e1=e.kk();
   System.out.println("e.a   =  " +e.a);
   System.out.println("e1.a   =  " +e1.a);
   }
}
```

Output **e.a = 2** **e1.a =12**

4.Discuss Recursion.

Recursion is the process of defining some thing in terms of itself. A method that calls itself is said to be recursive.

Eg:

```
class Facto
{
 int fact(int n)
   {
     int result;
     if(n==1)
      {
        return 1;
      }
     else
      {
        result = fact(n-1)*n;
        return result;
      }
   }
}
```

BCSM, IT DEPT

```
class Demo
{
  public static void main(String args[])
   {
     Facto f = new Facto();
     System.out.println("Factorial is = " +f.fact(4));
   }
}
```

5.Explain the super keyword.

The **super** keyword is similar to **this** keyword following are the scenarios where the super keyword is used.

- It is used to **differentiate the members** of superclass from the members of subclass, if they have same names.
- It is used to **invoke the superclass** constructor from subclass.

Differentiating the members

- If a class is inheriting the properties of another class. And if the members of the superclass have the names same as the sub class, to differentiate these variables we use super keyword as shown below.

```
super.variable
super.method();
```

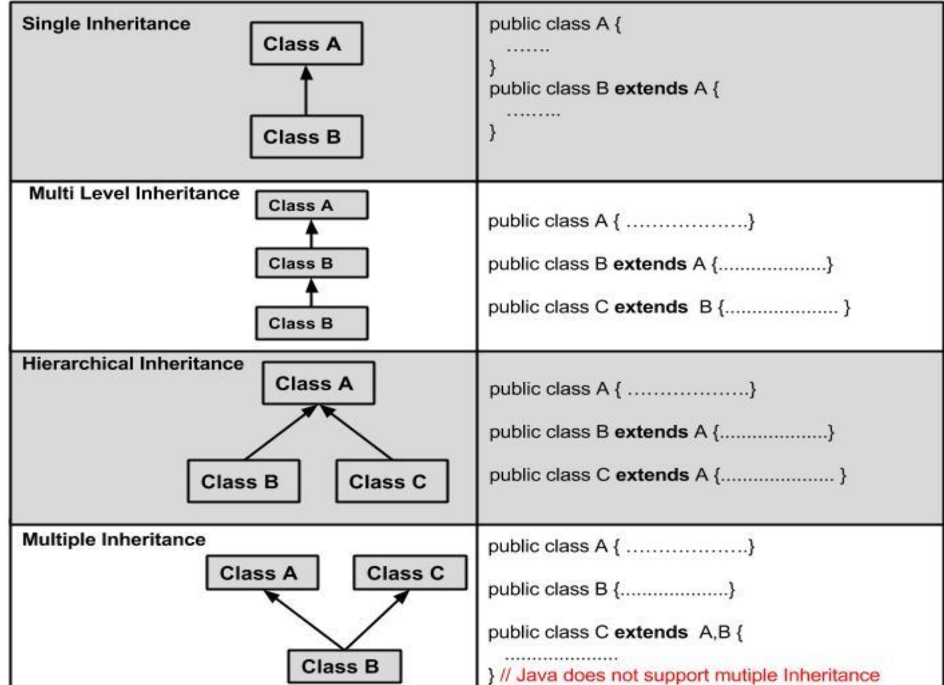
Invoking Super class constructor

If a class is inheriting the properties of another class, the subclass automatically acquires the default constructor of the super class. But if you want to call a parametrized constructor of the super class, you need to use the super keyword as shown below.

super(values);

BCSM,IT DEPT

There are various types of inheritance as demonstrated below.



A very important fact to remember is that Java does not support multiple inheritance. This means that a class cannot extend more than one class.

---

10 Marks:

1.Explain interface concept in java with example

- An interface is a reference type in Java, it is similar to class, it is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.
- Along with abstract methods an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.
- Writing an interface is similar to writing a class. But a class describes the attributes and behaviours of an object. And an interface contains behaviours that a class implements.
- Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

An interface is similar to a class in the following ways:

- An interface can contain any number of methods.

- An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.
- The byte code of an interface appears in a **.class** file.
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including:

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

Declaring Interfaces:

The **interface** keyword is used to declare an interface. Here is a simple example to declare an interface:

Example:

```
/* File name : NameOfInterface.java */
import java.lang.*;
//Any number of import statements

public interface NameOfInterface
{
  //Any number of final, static fields
  //Any number of abstract method declarations\
}
```

Interfaces have the following properties:

- An interface is implicitly abstract. You do not need to use the **abstract** keyword while declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.

Implementing Interfaces:

- When a class implements an interface, you can think of the class as signing a contract, agreeing to perform the specific behaviors of the interface.
- If a class does not perform all the behaviors of the interface, the class must declare itself as abstract.

BCSM, IT DEPT

- A class uses the **implements** keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

```
/* File name : MammalInt.java */
public class MammalInt implements Animal{

  public void eat(){
    System.out.println("Mammal eats");
  }

  public void travel(){
    System.out.println("Mammal travels");
  }

  public int noOfLegs(){
    return 0;
  }

  public static void main(String args[]){
    MammalInt m = new MammalInt();
    m.eat();
    m.travel();
  }
}
```

Result:
ammal eats
ammal travels

When overriding methods defined in interfaces there are several rules to be followed:

- Checked exceptions should not be declared on implementation methods other than the ones declared by the interface method or subclasses of those declared by the interface method.
- The signature of the interface method and the same return type or subtype should be maintained when overriding the methods.
- An implementation class itself can be abstract and if so interface methods need not be implemented.

When implementation interfaces there are several rules:

- A class can implement more than one interface at a time.
- A class can extend only one class, but implement many interfaces.
- An interface can extend another interface, similarly to the way that a class can extend another class.

BCSM,IT DEPT

Extending Interfaces:

- An interface can extend another interface, similarly to the way that a class can extend another class. The **extends** keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

Extending Multiple Interfaces:

- A Java class can only extend one parent class. Multiple inheritance is not allowed. Interfaces are not classes, however, and an interface can extend more than one parent interface.
- The extends keyword is used once, and the parent interfaces are declared in a comma-separated list.

2.Explain in detail about Exception Handling.

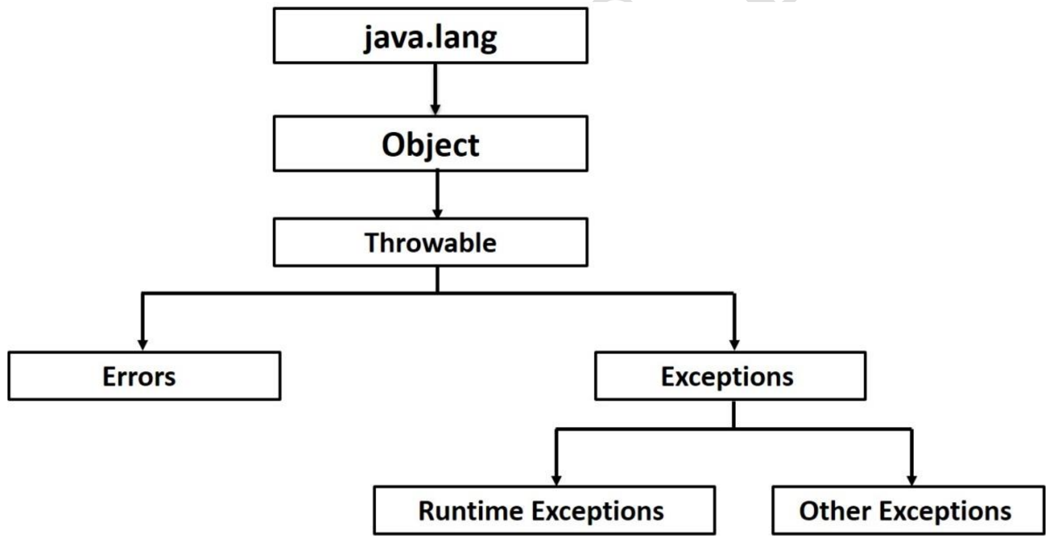
- An exception (or exceptional event) is a problem that arises during the **execution** of a program.
- When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore these exceptions are to be handled.
- An exception can occur for many different reasons, below given are some scenarios where exception occurs.
  - A user has entered invalid data.
  - A file that needs to be opened cannot be found.
  - A network connection has been lost in the middle of communications or the JVM has run out of memory.
- Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.
- Based on these we have three categories of Exceptions you need to understand them to know how exception handling works in Java,
  - **Checked exceptions:** A checked exception is an exception that occurs at the compile time, these are also called as compile time exceptions. These exceptions cannot simply be ignored at the time of compilation, the Programmer should take care of (handle) these exceptions.
  - **Unchecked exceptions:** An Unchecked exception is an exception that occurs at the time of execution, these are also called as Runtime Exceptions, these include programming bugs, such as logic errors or improper use of an API. runtime exceptions are ignored at the time of compilation.

BCSM.IT DEPT

- **Errors:** These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

Exception Hierarchy:

- All exception classes are subtypes of the java.lang.Exception class. The exception class is a subclass of the Throwable class.
- Other than the exception class there is another subclass called Error which is derived from the Throwable class.
- Errors are not normally trapped form the Java programs. These conditions normally happen in case of severe failures, which are not handled by the java programs. Errors are generated to indicate errors generated by the runtime environment. Example : JVM is out of Memory. Normally programs cannot recover from errors.
- The Exception class has two main subclasses: IOException class and RuntimeException Class.



Catching Exceptions:

- A method catches an exception using a combination of the **try** and **catch** keywords. A try/catch block is placed around the code that might generate an exception.
- Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following:

```
try
```

```
{
  //Protected code
}catch(ExceptionName e1)
{
  //Catch block
}
```

- The code which is prone to exceptions is placed in the try block, when an exception occurs, that exception occurred is handled by catch block associated with it.
- Every try block should be immediately followed either by a class block or finally block.
- A catch statement involves declaring the type of exception you are trying to catch.
- If an exception occurs in protected code, the catch block (or blocks) that follows the try is checked.
- If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block much as an argument is passed into a method parameter.

Multiple catch Blocks:

- A try block can be followed by multiple catch blocks. The syntax for multiple catch blocks looks like the following:

```
try
{
  //Protected code
}catch(ExceptionType1 e1)
{
  //Catch block
}catch(ExceptionType2 e2)
{
  //Catch block
}catch(ExceptionType3 e3)
{
  //Catch block
}
```

- The previous statements demonstrate three catch blocks, but you can have any number of them after a single try. If an exception occurs in the protected code, the exception is thrown to the first catch block in the list.
- If the data type of the exception thrown matches ExceptionType1, it gets caught there. If not, the exception passes down to the second catch statement.
- This continues until the exception either is caught or falls through all catches, in which case the current method stops execution and the exception is thrown down to the previous method on the call stack.

BCSM,IT DEPT

Example:

Here is code segment showing how to use multiple try/catch statements.

```
try
{
  file = new FileInputStream(fileName);
  x = (byte) file.read();
}catch(IOException i)
{
  i.printStackTrace();
  return -1;
}catch(FileNotFoundException f) //Not valid!
{
  f.printStackTrace();
  return -1;
}
```

The throws/throw Keywords:

- If a method does not handle a checked exception, the method must declare it using the **throws** keyword. The throws keyword appears at the end of a method's signature.
- You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the **throw** keyword.
- Try to understand the difference between throws and throw keywords, *throws* is used to postpone the handling of a checked exception and *throw* is used to invoke an exception explicitly.

The finally block

- The finally block follows a try block or a catch block. A finally block of code always executes, irrespective of occurrence of an Exception.
- Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.
- A finally block appears at the end of the catch blocks and has the following syntax:

```
try
{
  //Protected code
}catch(ExceptionType1 e1)
{
  //Catch block
}catch(ExceptionType2 e2)
{
```

BCSM,IT DEPT

```
  //Catch block
}catch(ExceptionType3 e3)
{
  //Catch block
}finally
{
  //The finally block always executes.
}
```

User-defined Exceptions:

- You can create your own exceptions in Java. Keep the following points in mind when writing your own exception classes:

- All exceptions must be a child of Throwable.
- If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.
- If you want to write a runtime exception, you need to extend the RuntimeException class.

We can define our own Exception class as below:

```
class MyException extends Exception{

}
```

3.Explain in detail about Constructor.

- The name of the constructor method should be always same as the name of the class.
- It may have optional list of arguments.
- The constructor method does not have a return type.
- Constructor is a special type member function, which execute automatically whenever an object is created.
- The declaration of the class serves only as a template and no memory is allocated at the time of declaration. so the data within the class cannot be initialized at the time of declaration.
- There can be more one constructor in a program. that is the constructor can be overloaded. These constructors are invoked automatically whenever an object is created.

Rules for creating Constructor:

- Name : Constructor name is same as class name.
- Scope : public
- Return Type : Nil
- Argument : Optional

BCSM,IT DEPT

- Overloading: Possible

Use:

- Variable Initialization
- Resource Allocation
- Type Conversion

Eg:

```
class Demo
{
    int a,b,c;
  Demo( int a1,int b1, int c1)
   {
       a=a1; b=b1; c=c1;
    }
  void disp()
    {
      System.out.println(a+b+c);
    }
 }

class Demo1
 {
     public static void main(String args[])
      {
      Demo D = new Demo(1,2,3);
      D.disp();
     }
 }
```

Here the parameter list should be always individually declared.

Constructors that take parameter (Parameterized constructor):

- Constructors that take arguments are called as parameterized constructor.
- We must pass the initial values as arguments to the constructor function when an object is declared.

  This can be done in two ways.

  1) By calling the constructor explicitly.
  2) By calling the constructor implicitly.

this Keyword

The **this Keyword** refers to the class in which it is used. When we used this keyword in the Demo class, it will refer to the Demo class. The this keyword is often used in the Constructor Method.

Eg:

BCSM,IT DEPT

```
class ka
{
   int x,y,z;
   ka( int x,int y)
   {
     this.x=x;
     this.y=y;
   }
  void disp()
   {
     z=x+y;
     System.out.println(z);
   }
}

class Java
{
  public static void main(String args[])
   {
     ka k=new ka(4,4);
     k.disp();
    }
}
```

4.Explain the concept of Inheritance.

Inheritance is the process of creating new class or classes from an already existing class or classes. The existing class is known as base class. The new class is known as sub class or derived class. The derived class inherits all the properties of the base class.

- When a base class is privately inherited then the public members of the base class become private members of the derived class. Therefore they are not accessible by using the objects of the derived class.
- When a base class is publicly inherited then the public members of the base class become public members of the derived class. Therefore they are accessible by using the objects of the derived class.
- In both case the private members are not inherited and therefore the private members of a base class will not become the members of its derived class.

(OR)

The ability to write a class that inherits the member variable or member function of another class.

(OR)

BCS M.IT DEPT

The ability to write a class that extends the definition of a superclass. The new class inherits the member function of this superclass .The new class may override the behavior it is inheriting .

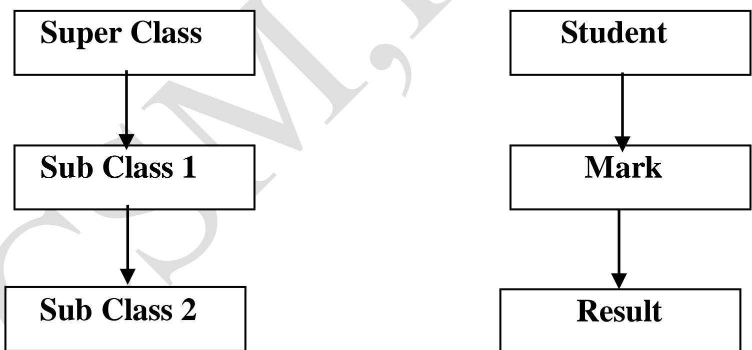
Advantage:

- Code Reusability.
- Reduce the debugging time.
- Reduce the testing time.
- Reduce the maintenance charge.

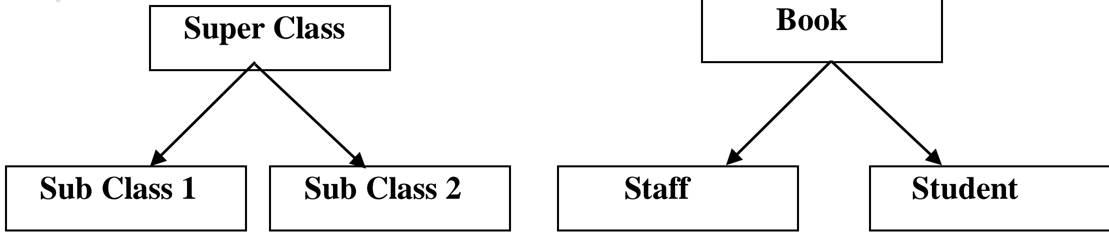
Types of Inheritance:

1. Single Inheritance: A class derived form a single base class is known as single inheritance.

2. Multilevel Inheritance: A class derived from another derived class.



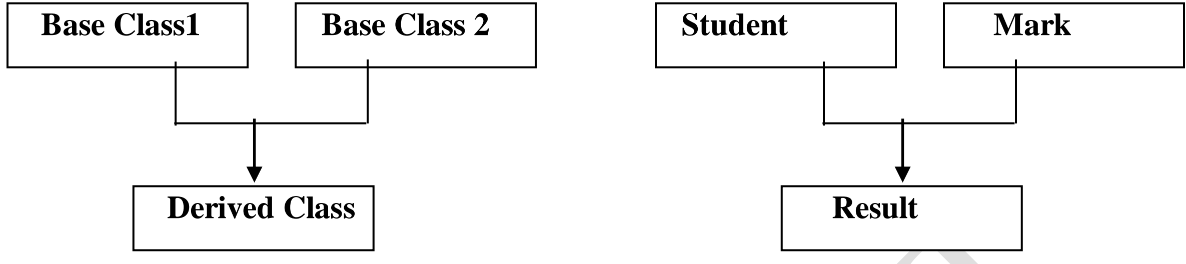
3. Hierarchical Inheritance: More than one Class is derived from a single base class.



BCSM,IT DEPT

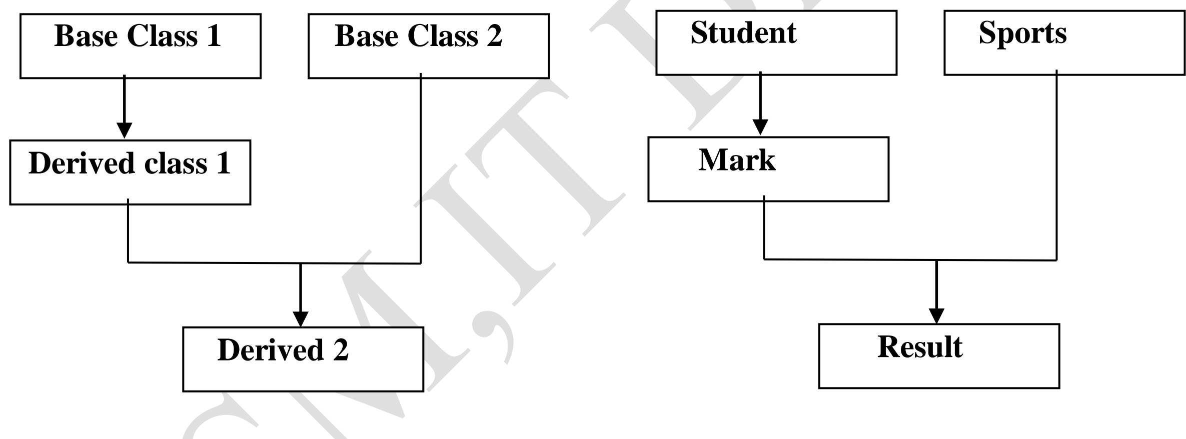
4. Multiple Inheritance: One class derived from more than one base class. (OR)

The ability to write a class that inherits the member variable and member functions of more than one class. see also inheritance.



5. Hybrid Inheritance: (Combination of above two inheritances)

In some situations we need to apply two or more inheritance to design a program. Such type of inheritance is known as hybrid inheritance.



5.Explain the following keyword.

Superclass

- A class from which a given inherits . This can be its immediate parent class or can be more levels away. Superclasses become more than more generic as you travel up the inheritance hierarchy and, for this reason , can often be abstract.

SubClass

- A class that descends or inherits (extends in Java terminology) from a given class. Subclasses are more specialized than the classes they inherit from.

Interface

BCSM,IT DEPT

- A formal set of method and constant declarations that must be defined by the classes that implement it.

Adapter class

- A class that implements all the methods of an interface. You then subclass the adapter class and only override the necessary methods from the original interface. Adapter classes are frequently used when listening for events.

Abstract class

- A class that contains abstract methods. Abstract classes cannot be instantiated.

Abstract method

- A method that declare but not implemented. Abstract method are used to ensure that subclasses implement the method.

Applet

- A Java programs that appears to be embedded in a web document.

AWT

- The collection of Java classes that allows implementation of Platform-independent **GUI.**

Package

- A collection of related class .

Thread

- A single flow of control within a program, similar to a process (or a running program) but easier to create and destroy than a process because less resource management is involved. Each thread must have its own resources, such as the program counter and the execution stack, as the context for execution. However, all threads in a program share many resource, such as memory space and opened files.

Multithreading

- The means to concurrently perform multiple tasks independently each other.

BCSM,IT DEPT

Exception

- It indicates the programmer that the particular method in-capable of handling particular Exception.

(Or)

- When over that particular method invoke through an object it should be given with in **try** and **catch** block.

Polymorphism

- The ability of a single object to operate on many different types.

********END********

BCSM,IT DEPT

UNIT-3

2 Marks:

1.Define stream.

- ✓ Java input and output is based on the use of *streams*, or sequences of bytes that travel from a source to a destination over a communication path.
- ✓ If your program is writing to a stream, it is the stream's *source*. If it is reading from a stream, it is the stream's *destination*.
- ✓ The communication path is dependent on the type of I/O being performed. It can consist of memory-to-memory transfers, a file system, a network, and other forms of I/O.
- ✓ Java defines two major classes of byte streams:InputStream and OutputStream.

2. Define a File class in java.io package.

- The File class does not specify how the information is retrieved from and stored in files it describes the properties of a file itself.
- A File object is used to obtain or manipulate the information associated with a disk file.

---

10 Marks:

1. Explain in detail about I/O PACKAGE.

Streams

- ✓ Java input and output is based on the use of *streams*, or sequences of bytes that travel from a source to a destination over a communication path.
- ✓ If your program is writing to a stream, it is the stream's *source*. If it is reading from a stream, it is the stream's *destination*.
- ✓ The communication path is dependent on the type of I/O being performed. It can consist of memory-to-memory transfers, a file system, a network, and other forms of I/O.
- ✓ Java defines two major classes of byte streams:InputStream and OutputStream.
- ✓ These streams are subclassed to provide a variety of I/O capabilities.
- ✓ The **Reader** and **Writer** classes to provide the foundation for 16-bit Unicode character- oriented I/O.
- ✓ These classes support internationalization of Java I/O. The Reader and Writer classes, such as InputStream and OutputStream, are subclassed to support additional capabilities.

The java.io Class Hierarchy

- ✓ The InputStream, OutputStream, Reader, and Writer classes are the major components of this hierarchy. Other high-level classes include the File,

BCSM,IT DEPT

FileDescriptor, RandomAccessFile, ObjectStreamClass, and StreamTokenizer classes.

The InputStream Class

The InputStream class is an abstract class that lays the foundation for the Java Input class hierarchy. As such, it provides methods that are inherited by all InputStream classes.

THE READ() METHOD

- ✓ The read() method is the most important method of the InputStream class hierarchy.
- ✓ It reads a byte of data from an input stream and blocks if no data is available. When a method *blocks*, it causes the thread in which it is executing to wait until data becomes available.
- ✓ It can read a single byte or an array of bytes, depending upon what form is used. It returns the number of bytes read, or -1 if an end of file is encountered with no bytes read.
- ✓ The read() method is overridden and overloaded by subclasses to provide custom read capabilities.

The available() Method

- ✓ The available()method returns the number of bytes that are available to be read without blocking. It is used to peek into the input stream to see how much data is available.

The close() Method

- ✓ The close() method closes an input stream and releases resources associated with the stream. It is always a good idea to close a stream to ensure that the stream processing is correctly terminated.

The skip() Method

- ✓ The skip() method skips over a specified number of input bytes. It takes a long value as a parameter. You can use the skip() method to move to a specific position within an input stream.

The OutputStream Class

- ✓ The OutputStream class is an abstract class that lays the foundation for the output stream hierarchy. It provides a set of methods that are the output analog to the InputStream methods.

The write() Method

- ✓ The write() method allows bytes to be written to the output stream. It provides three overloaded forms to write a single byte, an array of bytes, or a segment of an array.
- ✓ The blocking causes the thread executing the write() method to wait until the write operation has been completed.

BCSM,IT DEPT

The flush() Method

- ✓ The flush() method causes any buffered data to be immediately written to the output stream.
- ✓ Some subclasses of OutputStream support buffering and override this method to clean out their buffers and write all buffered data to the output stream.
- ✓ They must override the OutputStream flush() method because, by default, it does not perform any operations and is used as a placeholder.

The close() Method

- ✓ It is generally more important to close() output streams than input streams, so that any data written to the stream is stored before the stream is deallocated and lost.
- ✓ The close() method of OutputStream is used in the same manner as that of InputStream.

The Reader and Writer Classes

- ✓ The Reader and Writer classes are abstract classes at the top of a class hierarchy that support the reading and writing of Unicode character streams.

The Reader Class

- ✓ The Reader class supports the standard read(), reset(), skip(), mark(), markSupported(), and close() methods.
- ✓ In addition to these, the ready() method returns a boolean value that indicates whether the next read operation will succeed without blocking.
- ✓ The direct subclasses of the Reader class are BufferedReader, CharArrayReader, FilterReader, InputStreamReader, PipedReader, and StringReader.

The Writer Class

- ✓ The Writer class is the output complement to the Reader class. It declares the write(), flush(), and close() methods.
- ✓ Its direct subclasses are BufferedWriter, CharArrayWriter, FilterWriter, OutputStreamWriter, PipedWriter, StringWriter, and PrintWriter.
- ✓ Each of these subclasses, except PrintWriter, is an output complement to a Reader subclass.

---

2. **Discuss about The language package.**

- ✓ The Java language package is at the heart of the Java language.
- ✓ The language package contains many classes, each with a variety of member variables and methods. .

The Object Class

- ✓ The Object class is probably the most important of all Java classes, simply because it is the superclass of all Java classes.
- ✓ It is important to have a solid understanding of the Object class, as all the classes you develop will inherit the variables and methods of Object.
- ✓ The Object class implements the following important methods:
    - Object clone()
    - boolean equals(Object obj)
    - int hashCode()

BCSM IT DEPT

- final Class getClass()
- String toString()

✓ The Object **clone()** creates a clone of the object it is called on. clone creates and allocates memory for the new object that is being copied to. clone actually creates a new object and then copies the contents of the calling object to the new object. An example of using the clone method follows:

```
Circle circle1 = new Circle(1.0, 3.5, 4.2);

Circle circle2 = circle1.clone();
```

✓ The **equals()** method compares two objects for equality. equals is only applicable when both objects have been stored in a Hashtable.

✓ The **hashCode()** method returns the hashcode value for an object. Hashcodes are integers that uniquely represent objects in the Java system.

✓ The **getClass()** method returns the runtime class information for an object in the form of a Class object.

✓ The **toString()** method returns a string representing the value of an object.

Data Type Wrapper Classes

✓ The data type wrapper classes serve to provide object versions of the fundamental Java data types.

✓ Type wrapping is important because many Java classes and methods operate on classes rather than fundamental types.

✓ Following are the type wrapper classes supported by Java:

- Boolean
- Character
- Double
- Float
- Integer
- Long

✓ Although each wrapper implements methods specific to each data type, there are a handful of methods applicable to all the wrappers. These methods follow:

- ClassType(type)
- type typeValue()
- int hashCode()
- String toString()
- boolean equals(Object obj)
- static boolean valueOf(String s)

✓ The **ClassType** method is actually the creation method for each class. The wrapper creation methods take as their only parameter the type of data they are wrapping. This enables you to create a type wrapper from a fundamental type.

The Boolean Class

✓ The Boolean class wraps the boolean fundamental data type. Boolean implements only one method in addition to the common wrapper methods already mentioned:

BCSM,IT DEPT

```
static boolean getBoolean(String name)
```

- ✓ **getBoolean** returns a type boolean that represents the boolean property value of the String parameter name.
- ✓ The name parameter refers to a property name that represents a boolean property value. Because getBoolean is static, it is typically meant to be used without actually instantiating a Boolean object.
- ✓ The Boolean class also includes two final static (constant) data members: TRUE and FALSE.

The Character Class

- ✓ The Character class wraps the char fundamental type and provides some useful methods for manipulating characters.
- ✓ The methods implemented by Character, beyond the common wrapper methods, follow:
  - static boolean isLowerCase(char ch)
  - static boolean isUpperCase(char ch)
  - static boolean isDigit(char ch)
  - static boolean isSpace(char ch)
  - static char toLowerCase(char ch)
  - static char toUpperCase(char ch)
  - static int digit(char ch, int radix)
  - static char forDigit(int digit, int radix)
- ✓ The **isLowerCase** and **isUpperCase** methods return whether or not a character is an uppercase or lower case character.
- ✓ The **isDigit** method simply returns whether or not a character is a digit (0-9).
- ✓ The **isSpace** method returns whether or not a character is whitespace.
- ✓ The forDigit method performs the reverse of the digit method; it returns the character representation of an integer digit.
- ✓ The Character class provides two final static data members for specifying the radix limits for conversions: **MIN_RADIX and MAX_RADIX.**

Integer Classes

- ✓ The Integer and Long classes wrap the fundamental integer types int and long and provide a variety of methods for working with integer numbers.
- ✓ The methods implemented by Integer follow:
  - static int parseInt(String s, int radix)
  - static int parseInt(String s)
  - long longValue()
  - float floatValue()
  - double doubleValue()
  - static Integer getInteger(String name)
  - static Integer getInteger(String name, int val)
  - static Integer getInteger(String name, Integer val)
- ✓ The **parseInt** methods parse strings for an integer value, and return the value as an int.
- ✓ The **longValue, floatValue, and doubleValue** methods return the values of an integer converted to the appropriate type

BCSM,IT DEPT

- ✓ The **getInteger** methods return an integer property value specified by the String property name parameter name.
- ✓ The Integer class also includes two final static (constant) data members: **MINVALUE and MAXVALUE.**
- ✓ The Long object is very similar to the Integer object except it wraps the fundamental type long.

Floating Point Classes

- ✓ The Float and Double classes wrap the fundamental floating point types float and double. These two classes provide a group of methods for working with floating point numbers.
- ✓ The methods implemented by the Float class follow:
    - boolean isNaN()
    - static boolean isNaN(float v)
    - boolean isInfinite()
    - static boolean isInfinite(float v)
    - int intValue()
    - long longValue()
    - double doubleValue()
    - static int floatToIntBits(float value)
    - static float intBitsToFloat(int bits)
- ✓ The **isNaN** methods return whether or not the Float value is the special not-a-number (NaN) value. The first version of isNaN operates on the value of the calling Float object. The second version is static and takes the float to test as its parameter, v.
- ✓ The i**sInfinite** methods return whether or not the Float value is infinite, which is represented by the special **NEGATIVE_INFINITY and POSITIVE_INFINITY** final static member variables.

The Math Class

- ✓ The Math class contains many invaluable mathematical functions along with a few useful constants.
- ✓ The most useful methods implemented by the Math class follow:
    - static double sin(double a)
    - static double cos(double a)
    - static double tan(double a)
    - static double asin(double a)
    - static double acos(double a)
    - static double atan(double a)
    - static double exp(double a)
    - static double log(double a)
    - static double sqrt(double a)
    - static double pow(double a, double b)
    - static double ceil(double a)
    - static double floor(double a)
    - static int round(float a)

BCSM, IT DEPT

- static long round(double a)
- static double rint(double a)
- static double atan2(double a, double b)
- static synchronized double random()
- static int abs(int a)
- static long abs(long a)
- static float abs(float a)
- static double abs(double a)
- static int min(int a, int b)
- static long min(long a, long b)
- static float min(float a, float b)
- static double min(double a, double b)
- static int max(int a, int b)
- static long max(long a, long b)
- static float max(float a, float b)
- static double max(double a, double b)

✓ The trigonometric methods sin, cos, tan, asin, acos, and atan perform the standard trigonometric functions on double values. All the angles used in the trigonometric functions are specified in radians

✓ The **exp** method returns the exponential number E raised to the power of the double parameter a.

✓ Similarly, the **log** method returns the natural logarithm (base E) of the number passed in the parameter a.

✓ The **sqrt** method returns the square root of the parameter number a.

✓ The **pow** method returns the result of raising a number to a power. pow returns a raised to the power of b.

✓ The **ceil and floor** methods return the "ceiling" and "floor" for the passed parameter a. The ceiling is the smallest whole number greater than or equal to a, where the floor is the largest whole number less than or equal to a.

✓ The **round** methods round float and double numbers to the nearest integer value, which is returned as type int or long. Both round methods work by adding 0.5 to the number and then returning the largest integer that is less than or equal to the number.

✓ The **rint** method returns an integral value, similar to round, that remains a type double.

3.Discuss about String Classes.

✓ Unlike C and C++, text strings in Java are represented with classes rather than character arrays.

✓ The two classes that model strings in Java are **String and StringBuffer**. The reason for having two string classes is that the String class represents constant (immutable) strings and the StringBuffer class represents variable (mutable) strings.

BCSM, IT DEPT

The String Class

- ✓ The String class is used to represent constant strings. The String class has less overhead than StringBuffer, which means you should try to use it if you know that a string is constant. The creation methods for the String class follow:
  - String()
  - String(String value)
  - String(char value[])
  - String(char value[], int offset, int count)
  - String(byte ascii[], int hibyte, int offset, int count)
  - String(byte ascii[], int hibyte)
  - String(StringBuffer buffer)
- ✓ It is readily apparent from the number of creation methods for String that there are many ways to create String objects. The first creation method simply creates a new string that is empty. All of the other creation methods create strings that are initialized in different ways from various types of text data.
- ✓ Following are examples of using some of the String creation methods to create String objects:

String s1 = new String();//an empty String object (s1) is created.

String s2 = s1new String("Hello");// a String object (s2) is created from a literal String value, "Hello".

char cArray[] = {'H', 'o', 'w', 'd', 'y')};
String s3 = new String(cArray); // shows aString object (s3) being created from an array of characters.

String s4 = new String(cArray, 1, 3);// shows a String object (s4) being created from a subarray of characters.

- ✓ The subarray is specified by passing 1 as the offset parameter and 3 as the count parameter. This means that the subarray of characters is to consist of the first three characters starting at one character into the array. The resulting subarray of characters in this case consists of the characters 'o', 'w', and 'd'.
- ✓ Once you have some String objects created, you are ready to work with them using some of the powerful methods implemented in the String class.
- ✓ Some of the most useful methods provided by the String class follow:
  - int length()
  - char charAt(int index)
  - boolean startsWith(String prefix)
  - boolean startsWith(String prefix, int toffset)
  - endsWith(String suffix)
  - int indexOf(int ch)
  - int indexOf(int ch, int fromIndex)
  - int indexOf(String str)

BCSM,IT DEPT

- int indexOf(String str, int fromIndex)
- int lastIndexOf(int ch)
- int lastIndexOf(int ch, int fromIndex)
- int lastIndexOf(String str)
- int lastIndexOf(String str, int fromIndex)
- String substring(int beginIndex)
- String substring(int beginIndex, int endIndex)
- boolean equals(Object anObject)
- boolean equalsIgnoreCase(String anotherString)
- int compareTo(String anotherString)
- String concat(String str)
- String replace(char oldChar, char newChar)
- String trim()
- String toLowerCase()
- String toUpperCase()
- static String valueOf(Object obj)
- static String valueOf(char data[])
- static String valueOf(char data[], int offset, int count)
- static String valueOf(boolean b)
- static String valueOf(char c)
- static String valueOf(int i)
- static String valueOf(long l)
- static String valueOf(float f)
- static String valueOf(double d)

- ✓ The **length** method simply returns the length of a string, which is the number of Unicode characters in the string.
- ✓ The **charAt** method returns the character at a specific index of a string specified by the int parameter index.
- ✓ The **startsWith and endsWith** methods determine whether or not a string starts or ends with a prefix or suffix string, as specified by the prefix and suffix parameters.
- ✓ The second version of **startsWith** enables you to specify an offset to begin looking for the string prefix.
- ✓ Following are some examples of using these methods:

```
String s1 = new String("This is a test string!");

int len = s1.length();

char c = s1.charAt(8);

boolean b1 = s1.startsWith("This");

boolean b2 = s1.startsWith("test", 10);

boolean b3 = s1.endsWith("string.");
```

BCSM, IT DEPT

The indexOf methods

- ✓ It return the location of the first occurrence of a character or string within a String object.
- ✓ The first two versions of indexOf determine the index of a single character within a string, while the second two versions determine the index of a string of characters within a string.
- ✓ Each pair of indexOf methods contain a version for finding a character or string based on the beginning of the String object, as well a version that enables you to specify an offset into the string to begin searching for the first occurrence.
- ✓ If the character or string is not found, indexOf returns -1. The lastIndexOf methods work very much like indexOf, with the exception that lastIndexOf searches backwards through the string.
- ✓ Finally, the **lastIndexOf** method is called with a character parameter of ‘a’. The call to lastIndexOf returns 10, indicating the third ‘a’ in the string. Remember, lastIndexOf searches backward through the string to find the first occurrence of a character.
- ✓ The second version of substring returns a substring beginning at the index specified by **beginIndex** and ending at the index specified by **endIndex.**
- ✓ Similarly, the **compareTo** method compares two strings and returns an integer value that specifies whether the calling String object is less than, greater than, or equal to the anotherString parameter.
- ✓ The **concat** method is used to concatenate two String objects. The string specified in the str parameter is concatenated onto the end of the calling String object.
- ✓ The **replace** method is used to replace characters in a string. All occurrences of oldChar are replaced with newChar.
- ✓ The **toLowerCase** and t**oUpperCase** methods are used to convert all of the characters in a String object to lower and uppercase.
- ✓ Finally, the **valueOf** methods all return String objects that represent the particular type taken as a parameter.

The StringBuffer Class

- ✓ The StringBuffer class is used to represent variable, or non-constant, strings. The StringBuffer class is useful when you know that a string will change in value or in length. The creation methods for the StringBuffer class follow:
    - StringBuffer()
    - StringBuffer(int length)
    - StringBuffer(String str)
- ✓ The first creation method simply creates a new string buffer that is empty. The second creation method creates a string buffer that is length characters long, initialized with spaces. The third creation method creates a string buffer from a String object. This last creation method is useful when you need to modify a constant String object.
- ✓ Following are examples of using the StringBuffer creation methods to create StringBuffer objects:

```
String s1 = new String(“This is a string!”);

String sb1 = new StringBuffer();
```

BCSM, IT DEPT

String sb2 = new StringBuffer(25);

String sb3 = new StringBuffer(s1);

- ✓ Some of the most useful methods implemented by StringBuffer follow:
  - int length()
  - int capacity()
  - synchronized void setLength(int newLength)
  - synchronized char charAt(int index)
  - synchronized void setCharAt(int index, char ch)
  - synchronized StringBuffer append(Object obj)
  - synchronized StringBuffer append(String str)
  - synchronized StringBuffer append(char c)
  - synchronized StringBuffer append(char str[])
  - synchronized StringBuffer append(char str[], int offset, int len)
  - StringBuffer append(boolean b)
  - StringBuffer append(int I)
  - StringBuffer append(long l)
  - StringBuffer append(float f)
  - StringBuffer append(double d)
  - synchronized StringBuffer insert(int offset, Object obj)
  - synchronized StringBuffer insert(int offset, String str)
  - synchronized StringBuffer insert(int offset, char c)
  - synchronized StringBuffer insert(int offset, char str[])
  - StringBuffer insert(int offset, boolean b)
  - StringBuffer insert(int offset, int I)
  - StringBuffer insert(int offset, long l)
  - StringBuffer insert(int offset, float f)
  - StringBuffer insert(int offset, double d)
  - String toString()
- ✓ The **length** method is used to get the length, or number of characters in the string buffer.
- ✓ The **capacity** method is similar to length except it returns how many characters a string buffer has allocated in memory, which is sometimes greater than the length. Characters are allocated for a string buffer as they are needed. Many times more memory is allocated for a string buffer than is actually being used. In these cases, the capacity method will return the amount of memory allocated for the string buffer.
- ✓ You can explicitly change the length of a string buffer using the **setLength** method. An example of using setLength would be to truncate a string by specifying a shorter length.
- ✓ The last method of interest in StringBuffer is the **toString** method. toString returns the String object representation of the calling StringBuffer object. toString is useful when you have a StringBuffer object but need a String object.

BCSM IT DEPT

5. Write brief note on File Navigation and I/O:

- There are several other classes that we would be going through to get to know the basics of File Navigation and I/O.

    - File Class
    - FileReader Class
    - FileWriter Class

Directories in Java:

- A directory is a File which can contains a list of other files and directories. You use **File** object to create directories, to list down files available in a directory.
- For complete detail check a list of all the methods which you can call on File object and what are related to directories.

Creating Directories:

There are two useful **File** utility methods, which can be used to create directories:

- The **mkdir( )** method creates a directory, returning true on success and false on failure. Failure indicates that the path specified in the File object already exists, or that the directory cannot be created because the entire path does not exist yet.
- The **mkdirs()** method creates both a directory and all the parents of the directory.

Following example creates "/tmp/user/java/bin" directory:

```
import java.io.File;

public class CreateDir {
  public static void main(String args[]) {
    String dirname = "/tmp/user/java/bin";
    File d = new File(dirname);
    // Create directory now.
    d.mkdirs();
  }
}
```

Compile and execute above code to create "/tmp/user/java/bin".

Listing Directories:

- You can use **list( )** method provided by **File** object to list down all the files and directories available in a directory as follows:

```
import java.io.File;
```

BCSM, IT DEPT

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

    File file = null;
    String[] paths;

    try{
      // create new file object
      file = new File("/tmp");

      // array of files and directory
      paths = file.list();

      // for each name in the path array
      for(String path:paths)
      {
        // prints filename and directory name
        System.out.println(path);
      }
    }catch(Exception e){
      // if any error occurs
      e.printStackTrace();
    }
  }
}
```

********END********

BCSM,IT DEPT

UNIT- 4

2 Mark:

1. What is meant by Frame?

- It is a subclass of Window. It has a title bar, menu bar, borders, and resizing corners.Using **Frame** we can create a normal window. (I.e., it encapsulates what is commonly available in window).**Frame** run on local machine. (But **Applet** run on web browser also).
- **Working with Frame Window:**

  After the applet, you will mostly create windows by using Frame.
- **Frame constructors:**
  - Frame () //creates a standard window that does not contain a title.
  - Frame (String Name)//creates a window with the title specified by Name.

5Mark:

1.Explain Applet package.

An applet has a well-defined life cycle, as shown in Figure 1. Applets do not need to be explicitly constructed. The runtime environment associated with their applet context-- the Web browser or applet viewer, automatically constructs them.

The **init() method** provides the capability to load applet parameters and perform any necessary initialization processing.

The **start() method** serves as the execution entry point for an applet when it is initially executed and restarted as the result of a user returning to the Web page that contains the applet.

The **stop() method** provides the capability to **stop()** an applet's execution when the Web page containing the applet is no longer active.

The **destroy() method** is used at the end of an applet's life cycle to perform any termination processing.

---

10 Mark:

1.Discuss about Java Networking package.

The Internet Protocol Suite

- The java.net package provides a set of classes that support network programming using the communication protocols employed by the Internet. These protocols are

BCSM,IT DEPT

known as the *Internet protocol suite* and include the *Internet Protocol* (*IP*), the *Transport Control Protocol* (*TCP*), and the *User Datagram Protocol* (*UDP*) as well as other, less prominent supporting protocols

- "The Internet" is defined as "the collection of all computers that can communicate, using the Internet protocol suite, with the computers and networks registered with the *Internet Network Information Center* (*InterNIC*)."
- This definition includes all computers to which you can directly send Internet Protocol packets.

Connection-Oriented Versus Connectionless Communication

- Transport protocols are used to deliver information from one port to another and thereby enable communication between application programs.
- They use either a connection- oriented or connectionless method of communication. TCP is a connection-oriented protocol, and UDP is a connectionless transport protocol.
- The TCP connection-oriented protocol establishes a communication link between a source port/IP address and a destination port/IP address.
- The ports are bound together via this link until the connection is terminated and the link is broken.
- A telephone connection is established, communication takes place, and then the connection is terminated.
- The reliability of the communication between the source and destination programs is ensured through error-detection and error-correction mechanisms that are implemented within TCP.
- TCP implements the connection as a stream of bytes from source to destination. This feature allows the use of the stream I/O classes provided by java.io.
- The UDP connectionless protocol differs from the TCP connection-oriented protocol in that it does not establish a link for the duration of the connection. An example of a connectionless protocol is **postal mail.**
- When using UDP, an application program writes the destination port and IP address on a datagram and then sends the datagram to its destination.
- UDP is less reliable than TCP because there are no delivery-assurance or error-detection-and-correction mechanisms built into the protocol.Application protocols such as FTP, SMTP, and HTTP use TCP to provide reliable, stream-based communication between client and server programs.
- Other protocols, such as the Time Protocol, use UDP because speed of delivery is more important than end-to-end reliability.

Multicast Addressing

- Most TCP/IP communication is ***unicast***--packets are sent from a source host to a destination host in a point-to-point fashion. Unicast communication is used by the majority of Internet services.
- A host to be able to simultaneously send IP packets to multiple destination hosts--for example, to transmit an audio or video stream. This form of communication is known as ***multicast*.**

BCSM,IT DEPT

- Multicast communication enables a host to transmit IP packets to multiple hosts, referred to as a *host group*, using a single destination IP address.
- Host groups may be permanent or temporary. Permanent groups are assigned fixed IP addresses.
- Temporary groups are dynamically assigned IP address. Hosts may join or leave a host group in a dynamic fashion--even permanent groups. The existence of a host group is independent of its members.
- Multicast routers are used to send IP multicast packets to the members of host groups.

The InetAddress Class

- The InetAddress class encapsulates Internet addresses. It supports both numeric IP addresses and host names.
- The InetAddress class has no public variables or constructors. It provides 10 access methods that support common operations on Internet addresses. Three of these methods are static.
- The **getLocalHost()**method is a static method that returns an InetAddress object that represents the Internet address of the local host computer.
- The static **getByName()-**method returns an InetAddress object for a specified host.
- The static **getAllByName()**method returns an array of all Internet addresses associated with a particular host.
- The **getAddress()** method gets the numeric IP address of the host identified by the InetAddress object, and the **getHostName()**method gets its domain name.
- The **getHostAddress()**method returns the numeric IP address of an InetAddress object as a dotted decimal string.
- The **isMulticastAddress()** method returns a boolean value that indicates whether an InetAddress object represents a multicast address.
- The equals(), hashCode(), and toString() methods override those of the Object class.

The Socket Class

- The Socket class implements client connection-based sockets. These sockets are used to develop applications that utilize services provided by connection-oriented server applications.
- The Socket class provides eight constructors that create sockets and optionally connect them to a destination host and port.
- The access methods of the Socket class are used to access the I/O streams and connection parameters associated with a connected socket.
- The **getInetAddress() and getPort()** methods get the IP address of the destination host and the destination host port number to which the socket is connected.
- The **getLocalPort()** method returns the source host local port number associated with the socket.

BCSM, IT DEPT

- The **getLocalAddress()** method returns the local IP address associated with the socket.
- The **getInputStream() and getOutputStream()** methods are used to access the input and output streams associated with a socket.
- The **close()** method is used to close a socket.
- The **getSoLinger() and setSoLinger()** methods are used to get and set a socket's SO_LINGER option, which identifies how long a socket is to remain open after a close() method has been invoked and data remains to be sent over the socket.
- The **getSoTimeout() and setSoTimeout()** methods are used to get and set a socket's SO_TIMEOUT option, which is used to identify how long a read operation on the socket is to be blocked before it times out and the blocking ends.
- The **getTcpNoDelay() and setTcpNoDelay()** methods are used to get and set a socket's TCP_NODELAY option, which is used to specify whether Nagle's algorithm should be used to buffer data that is sent over a socket connection. When TCP_NODELAY is true, Nagle's algorithm is disabled.
- The **setSocketImplFactory()** class method is used to switch from the default Java socket implementation to a custom socket implementation.
- The **toString()** method returns a string representation of the socket

The ServerSocket Class

- The ServerSocket class implements a TCP server socket. It provides three constructors that specify the port to which the server socket is to listen for incoming connection requests, an optional maximum connection request queue length, and an optional Internet address.
- The Internet address argument allows ***multihomed*** hosts (that is, hosts with more than one Internet address) to limit connections to a specific interface.
- The **accept()** method is used to cause the server socket to listen and wait until an incoming connection is established. It returns an object of class Socket once a connection is made. This Socket object is then used to carry out a service for a single client.

- The **getInetAddress()** method returns the address of the host to which the socket is connected.
- The **getLocalPort()** method returns the port on which the server socket listens for an incoming connection.
- The **toString()**method returns the socket's address and port number as a string in preparation for printing.
- The **getSoTimeout() and setSoTimeout()** methods set the socket's SO_TIMEOUT parameter.
- The **close()** method closes the server socket.
- The static **setSocketFactory()** method is used to change the default ServerSocket implementation to a custom implementation.
- The **implAccept()** method is used by subclasses of ServerSocket to override the accept() method.

BCSM, IT DEPT

The DatagramSocket Class

- The DatagramSocket class implements client and server sockets using the UDP protocol. UDP is a connectionless protocol that allows application programs (both clients and servers) to exchange information using chunks of data known as *datagrams*.
- DatagramSocket provides three constructors. The default constructor creates a datagram socket for use by client applications. No port number is specified.
- The second constructor allows a datagram socket to be created using a specified port. This constructor is typically used with server applications.
- The third constructor allows an Internet address to be specified in addition to the port. This is used to restrict service to a specific host interface.
- The **send() and receive()** methods are used to send and receive datagrams using the socket. The datagrams are objects of class DatagramPacket.
- The **getLocalPort() and getLocalAddress()** methods return the local port and Internet address of the socket.
- The **close()** method closes this socket.
- The **getSoTimeout() and setSoTimeout()**methods get and set the socket's SO_TIMEOUT parameter.

The DatagramPacket Class

- The DatagramPacket class encapsulates the actual datagrams that are sent and received using objects of class DatagramSocket
- Two different constructors are provided: one for datagrams that are received from a datagram socket, and one for creating datagrams that are sent over a datagram socket.
- The arguments to the received datagram constructor are a byte array used as a buffer for the received data, and an integer that identifies the number of bytes received and stored in the buffer.
- The sending datagram constructor adds two additional parameters: the IP address and port where the datagram is to be sent.
- **Eight access methods are provided.**
- **The getAddress() and getPort()** methods are used to read the destination IP address and port of the datagram.
- The **getLength() and getData()** methods are used to get the number of bytes of data contained in the datagram and to read the data into a byte array buffer.
- The setAddress(), setPort(), setLength(), and setData() methods allow the datagram's IP address, port, length, and data values to be set.

The MulticastSocket Class

The MulticastSocket class is used for developing clients and servers for IP multicasting. It provides the capability for hosts to join and leave multicast groups. All hosts in a multicast group receive UDP datagrams that are sent to the IP address of the group. Each host in the group listens on a common UDP port for the datagrams of the multicast application.

BCSM.IT DEPT

The MulticastSocket class has two constructors--a default parameterless constructor and a constructor that specifies the port number on which to listen for multicast datagrams. The MulticastSocket class has **seven access methods**:

- joinGroup()
- leaveGroup()
- setInterface()
- getInterface()
- setTTL()
- getTTL()
- send()

The **joinGroup() and leaveGroup()** methods are used to join and leave a multicast group at a specified Internet address.

The **getInterface() and setInterface()** methods are used to get and set the IP address of the host interface that is used for multicasting.

The **getTTL() and setTTL()** methods are used to get and set the *time-to-live* for multicast packets that are sent on the multicast socket. Time-to-live specifies the number of times that a packet is forwarded before it expires.

The **send()** method is used to send a datagram to multicast IP address.

2.Discuss about The URL Class,Components.

- ⇨ The URL class encapsulates Web objects by their URL address. It provides a set of constructors that allow URL objects to be easily constructed, and a set of access methods that allow high-level read and write operations to be performed using URLs.
- ⇨ The four URL constructors allow URL objects to be created using a variety of URL parameters, such as protocol type, host name, port, and file path.
- ⇨ These parameters may be supplied separately or in text form as part of an URL string. The URL class treats a file's path and name as a single entity to provide a more convenient way of working with URL components.
- ⇨ URLs can be constructed using their absolute address or using an address that is relative to another URL. Up until now we have been working with the complete or *absolute* address of an URL.
- ⇨ A *relative address* is a path/filename or file offset that is specified relative to an absolute URL.
- ⇨ The URL access methods provide a full set of URL processing capabilities. The **getProtocol(), getHost(), getPort(), getFile(), and getRef()** methods allow the individual address components of the URL to be determined.
- ⇨ The **getContent() and openStream()** methods allow reading of the Web object pointed to by the URL.
- ⇨ The **toExternalForm() and toString()**methods enable URLs to be converted into strings to support display and printing.

BCSMIT DEPT

- The **equals()**method compares URLs, and the sameFile() method compares the Web objects pointed to by the URLs.
- The **openConnection()** method creates an object of class URLConnection to the Web object pointed to be the URL.

URLConnection and HttpURLConnection

- The URLConnnection class is an abstract class that encapsulates an active HTTP connection to a Web object represented by an URL. It provides a number of methods for getting information about the Web object and about the connection to the Web object, and for interacting with the Web object.
- URLConnection defines several class variables that specify the connection state and associated parameters. It also supplies numerous methods that provide access to the HTTP-specific fields of the connection.
- The HttpURLConnection class is a subclass of URLConnection that provides direct access to the HTTP parameters involved in a client/server HTTP connection.

URLEncoder

- The URLEncoder class is a simple class that provides a single static method, encode(), for converting text strings to a form that is suitable for use as part of an URL. This format is known as x-www-form-urlencoded and is typically used to encode form data that is sent to a CGI script.
- The encode() method converts spaces to plus signs (+) and uses the percent character (%) as an escape code to encode special characters. The two characters that immediately follow a percent sign are interpreted as hexadecimal digits that are combined to produce an eight-bit value.

3.Explain APPLET concept in java.

- An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.
- There are some important differences between an applet and a standalone Java application, including the following:

- An applet is a Java class that extends the java.applet.Applet class.
- A main() method is not invoked on an applet, and an applet class will not define main().
- Applets are designed to be embedded within an HTML page.
- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
- A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
- The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
- Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.

BCSM, IT DEPT

- Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file.

Life Cycle of an Applet:

Four methods in the Applet class give you the framework on which you build any serious applet:

- **init:** This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.
- **start:** This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.
- **stop:** This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.
- **destroy:** This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.
- **paint:** Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

---

2. Write in detail about the Applet CLASS:

Every applet is an extension of the *java.applet.Applet class*. The base Applet class provides methods that a derived Applet class may call to obtain information and services from the browser context.

These include methods that do the following:

- Get applet parameters
- Get the network location of the HTML file that contains the applet
- Get the network location of the applet class directory
- Print a status message in the browser
- Fetch an image
- Fetch an audio clip
- Play an audio clip
- Resize the applet

Additionally, the Applet class provides an interface by which the viewer or browser obtains information about the applet and controls the applet's execution. The viewer may:

- request information about the author, version and copyright of the applet
- request a description of the parameters the applet recognizes
- initialize the applet

BCSM, IT DEPT

- destroy the applet
- start the applet's execution
- stop the applet's execution

The Applet class provides default implementations of each of these methods. Those implementations may be overridden as necessary.

The "Hello, World" applet is complete as it stands. The only method overridden is the paint method.

Invoking an Applet:

An applet may be invoked by embedding directives in an HTML file and viewing the file through an applet viewer or Java-enabled browser.

The <applet> tag is the basis for embedding an applet in an HTML file. Below is an example that invokes the "Hello, World" applet:

```
<html>
<title>The Hello, World Applet</title>
<hr>
<applet code="HelloWorldApplet.class" width="320" height="120">
If your browser was Java-enabled, a "Hello, World"
message would appear here.
</applet>
<hr>
</html>
```

- The code attribute of the <applet> tag is required. It specifies the Applet class to run. Width and height are also required to specify the initial size of the panel in which an applet runs. The applet directive must be closed with a </applet> tag.
- If an applet takes parameters, values may be passed for the parameters by adding <param> tags between <applet> and </applet>. The browser ignores text and other tags between the applet tags.
- Non-Java-enabled browsers do not process <applet> and </applet>. Therefore, anything that appears between the tags, not related to the applet, is visible in non-Java-enabled browsers.
- The viewer or browser looks for the compiled Java code at the location of the document. To specify otherwise, use the codebase attribute of the <applet> tag as shown:

```
<applet codebase="http://amrood.com/applets"
code="HelloWorldApplet.class" width="320" height="120">
```

BCSM,IT DEPT

If an applet resides in a package other than the default, the holding package must be specified in the code attribute using the period character (.) to separate package/class components. For example:

```
<applet code="mypackage.subpackage.TestApplet.class"
        width="320" height="120">
```

3.Describe Getting Applet Parameters:

- The following example demonstrates how to make an applet respond to setup parameters specified in the document. This applet displays a checkerboard pattern of black and a second color.
- The second color and the size of each square may be specified as parameters to the applet within the document.
- CheckerApplet gets its parameters in the init() method. It may also get its parameters in the paint() method. However, getting the values and saving the settings once at the start of the applet, instead of at every refresh, is convenient and efficient.
- The applet viewer or browser calls the init() method of each applet it runs. The viewer calls init() once, immediately after loading the applet.
- (Applet.init() is implemented to do nothing.) Override the default implementation to insert custom initialization code.
- The Applet.getParameter() method fetches a parameter given the parameter's name (the value of a parameter is always a string).
- If the value is numeric or other non-character data, the string must be parsed.

The following is a skeleton of CheckerApplet.java:

```
import java.applet.*;
import java.awt.*;
public class CheckerApplet extends Applet
{
  int squareSize = 50;// initialized to default size
  public void init () {}
  private void parseSquareSize (String param) {}
  private Color parseColor (String param) {}
  public void paint (Graphics g) {}
}
```

Here are CheckerApplet's init() and private parseSquareSize() methods:

```
public void init ()
{
  String squareSizeParam = getParameter ("squareSize");
  parseSquareSize (squareSizeParam);
  String colorParam = getParameter ("color");
```

BCSM,IT DEPT

```
    Color fg = parseColor (colorParam);
    setBackground (Color.black);
    setForeground (fg);
  }
  private void parseSquareSize (String param)
  {
    if (param == null) return;
    try {
      squareSize = Integer.parseInt (param);
    }
    catch (Exception e) {
     // Let default value remain
    }
  }
```

- The applet calls parseSquareSize() to parse the squareSize parameter. parseSquareSize() calls the library method Integer.parseInt(), which parses a string and returns an integer.
- Integer.parseInt() throws an exception whenever its argument is invalid.
- Therefore, parseSquareSize() catches exceptions, rather than allowing the applet to fail on bad input.
- The applet calls parseColor() to parse the color parameter into a Color value. parseColor() does a series of string comparisons to match the parameter value to the name of a predefined color. You need to implement these methods to make this applet works.

Specifying Applet Parameters:

The following is an example of an HTML file with a CheckerApplet embedded in it. The HTML file specifies both parameters to the applet by means of the <param> tag.

```
<html>
<title>Checkerboard Applet</title>
<hr>
<applet code="CheckerApplet.class" width="480" height="320">
<param name="color" value="blue">
<param name="squaresize" value="30">
</applet>
<hr>
</html>
```

Application Conversion to Applets:

- It is easy to convert a graphical Java application (that is, an application that uses the AWT and that you can start with the java program launcher) into an applet that you can embed in a web page.

BCSM, IT DEPT

Here are the specific steps for converting an application to an applet.

- Make an HTML page with the appropriate tag to load the applet code.

- Supply a subclass of the JApplet class. Make this class public. Otherwise, the applet cannot be loaded.
- Eliminate the main method in the application. Do not construct a frame window for the application. Your application will be displayed inside the browser.
- Move any initialization code from the frame window constructor to the init method of the applet. You don't need to explicitly construct the applet object.the browser instantiates it for you and calls the init method.
- Remove the call to setSize; for applets, sizing is done with the width and height parameters in the HTML file.
- Remove the call to setDefaultCloseOperation. An applet cannot be closed; it terminates when the browser exits.
- If the application calls setTitle, eliminate the call to the method. Applets cannot have title bars. (You can, of course, title the web page itself, using the HTML title tag.)
- Don't call setVisible(true). The applet is displayed automatically.

********END********

BCSM,IT DEPT

UNIT- 5

5Mark:

1. Explain AWT Event.
ActionEvent:

Generated by user interface actions, such as clicling on a button or selecting a menu item.

AdjustmentEvent:
Genterated by scrolling actions.

ComponentEvent:
Generated by changes to the position ,focus,or sizing of a window componets,or by a keyboard or other mouse action.

ContainerEvent:
Generated by events associated with adding and removing components from a container.

ItemEvent:
Generated by a component state change,such as selecting an item form a list.

TextEvent:
Generated by text-related events,such as changing tah value of a text field.

FocusEvent:
Generated by a change in the staus of a component's input focus.

KeyEvent:
An event which indicates that a ksystroke occurred in a component.

MouseEvent:
An event which indicates that amouse action occurred in a component .
This event is used both for mouse events(click,enter,exit) and mouse motion events(move and drags).

WindowEvent:
Generated by events such as the opening,closing and minimizing of a window.
The AWTEvent class and its subclass allow window-related events to be Directed to specific objects that listen those events. Thes objects implement EventListener interface. The java.utilEventListener interface is tah top-level Interface of the event listener hierarchy. It is an interface in name only because it Does not define any constants or methods. It is extended by the following interface of java.awt.event:

BCSM,IT DEPT

2.Discuss the adapter classes of java.awt.event.

ComponentAdapter

Implements the ComponentListener interface and handles ComponentEvent events.

ContainerAdapter

Implements the ContainerListener interface and handles ContainerEvent events.

FocusAdapter

Implements the FocusListener interface and handles FocueEvent events.

KeyAdapter

Implements the KeyListener interface and handles KeyEvent events.

MouseAdapter

Implements the MouseListener interface and handles clicking-related.

MouseMotinAdapter

Implements the MouseMotionListener interface and handles movement-related MouseEvent events.

WindowAdapter

Implements the WindowListener interface and handles WindowEvent events.

---

10 Mark:

1..Discuss about the AWT COMPONENT.

Button

A Button component is a simple pushbutton that displays a string label and responds to mouse presses from the user. Pressing a button triggers an action event.

Canvas

A Canvas is a blank area suitable for drawing in. You can use a Graphics object to put polygons or images on the Canvas. Because a Canvas has no predefined responses to events and because its appearance is completely arbitrary, a Canvas is a good place to start when designing custom components that look unlike the standard components.

Checkbox

A Checkbox is a small box with two states: either it's checked or unchecked. Clicking the mouse over an enabled Checkbox toggles its state. A Checkbox can have a string label.

BCSM,IT DEPT

CheckboxGroup

You put Checkboxes in a CheckboxGroup in order to make them exhibit "radio button" behavior: when one is checked, the others become unchecked. Only one Checkbox from the group can be checked at any time. The CheckboxGroup does not act as a container for the Checkbox component; it only tells some of the Checkboxes to uncheck themselves when necessary. Checkboxes can be placed in a group on creation or by calling their setCheckboxGroup() method.

Choice

A Choice component allows the user to specify one of a short list of choices, which appear on a little popup menu next to the current choice. The choices on the list are identified by a string name.

Label

A Label component displays a line of text. The text can be aligned to the left, right, or center of the Label. The user isn't allowed to edit the text in a Label. Use a TextField for that.

List

A List presents a scrollable list of items, identified by string names. Use this instead of a Choice when multiple selections are meaningful or when there may be too many items to conveniently display on a single popup menu.

Scrollbar

Most Scrollbar components are automatically generated when required by List or TextArea components. If you want to create your own scrollbars, you can do so. The orientation is specified by the constants Scrollbar.HORIZONTAL and Scrollbar.VERTICAL. The Scrollbar reports its current position via the getValue() method. To make the values meaningful, set the minimum and maximum values, together with the line and page increment values, using either the full five-argument constructor or the setValues() method.

There are five basic operations on a Scrollbar: line up, line down, page up, page down, and absolute positioning. Corresponding to these, there are five scrollbar event types. You don't have to discriminate between these very often. The event argument is always the integer value reflecting the new scrollbar position. Unless you want real-time response to Scrollbar actions, it is not necessary to handle Scrollbar events at all.

TextField

A TextField component holds a single line of text in a little window. The text is allowed to be longer than the window, in which case only part of it shows. By default, the user is allowed to edit the text. You can also set a TextField to be read only using

BCSM, IT DEPT

setEditable(false). This method is from class TextComponent, the abstract superclass of both TextField and TextArea.

TextArea

A TextArea is a pane containing lines of text. Like a TextField, it can be editable or not.

Menus

The AWT includes support for pulldown menus. The various components that implement the menus are not of class Component, but rather of class java.awt.MenuComponent. This reflects the "popup" nature of menus: they don't occupy space in a parent container, but pop up on top of other windows when required.

A menu is represented by a java.awt.Menu object. It can only be displayed on a menu bar (java.awt.MenuBar). In turn, a menu bar must be associated to a Frame in order to be useful. So a Frame can have a MenuBar, a MenuBar contains Menus, and a Menu contains MenuItems.

A MenuItem is a choice on a menu which is labeled with a string. CheckboxMenuItem extends MenuItem and has a string label with a checkbox gadget beside it. Selecting the item toggles the checkbox. The Menu class itself extends MenuItem, so a menu can be an item on another menu, that is to say, a submenu.

2.Explain Layout Managers.

When you add components into a container, it is the layout manager of the container which determines the actual size and location of each component. The two argument forms of add() allow you to specify a placement argument which is interpreted by the layout manager. The java.awt classes implementing layout manager policies are described here. They all implement the interface java.awt.LayoutManager.

All containers have a default layout manager, but you can designate whichever layout manager has your favorite policy by passing a new instance of the LayoutManager to the container's setLayout() method. You can also provide custom layout managers by implementing the LayoutManager interface yourself. The sample program AWTDemo.java demonstrates many of the layout managers in action.

BorderLayout

The possible placements are "North," "South," "East," "West," and "Center." They are identified by the string names. The components around the edges are laid out first and the center component gets the leftover room. This can make some components larger than necessary since they are stretched out to meet the edges of the container. Put the components inside a Panel to avoid this.

CardLayout

This layout manager lets several components occupy the same space, with only one visible at a time. Think of the components as lying on "cards," which are shown one at a

BCSM.IT.DEPT

time. You need a way for the user to flip through the cards. Typically this is a Choice or a series of Checkboxes in a CheckboxGroup. You can also have next/back Buttons.

FlowLayout

This is one of the simpler layout managers. Components are arranged left to right in a row until no more fit. Then a new row is begun. Each row is centered in the parent component by default. This is the default layout for Panels.

GridLayout

A GridLayout arranges components in a grid of rectangular cells, all the same size. The contents of each cell are resized to fill the cell so you may want to put them on a panel in some of the cells and let them take their natural size.
As you add components to a GridLayout, the cells are populated in reading order: from left to right in each row and then down to the next row.

GridBagLayout

The most flexible layout manager provided by java.awt is the GridBagLayout. Like a GridLayout, it is based on a rectangular array of cells. However, each component may occupy a rectangular area covering several cells. There is no requirement that the child components have the same size. Each child component has an associated GridBagConstraints object to give hints to the layout manager about its minimum size and preferred position in the container.

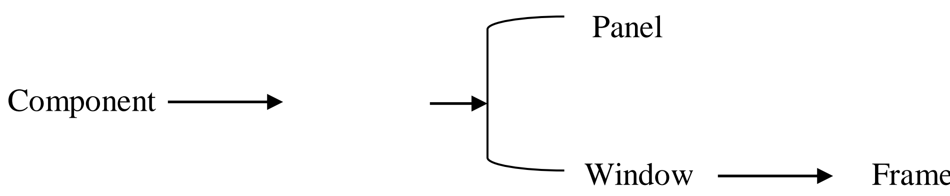
3.Explain in detail about AWT Classes.

- AWT stands for Abstract Window Toolkit.
- It contains several classes and methods that allow you to create and manage windows.
- The AWT classes are contained in the java.awt package.
- It is one of the Java's largest packages.

Window Fundamentals:

- The AWT defines windows according to a class hierarchy.
- The two most common windows are,
    - Panel, which is used by applets.
    - Frame, which is used to create standard window.
- The functionality of these windows is derived from their parent classes.

The class hierarchy for Panel and Frame:



BCSMIT DEPT

Component:

- **Component** is an abstract class that encapsulates all of the attributes of a visual component.
- All user interface elements that are displayed on the screen and interact with the user are subclasses of **Component**.
- It defines hundred of public methods that are responsible for managing events.
- A **Component** object is responsible for remembering the current foreground and background colors and the currently selected text font.

Container:

- The **Container** class is a subclass of Component.
- It has additional methods that allow other Component objects to be nested within it.
- A **Container** is responsible for laying out (positioning) any component that it contains.
- It uses various layout managers to do this work.

Panel:

- The **Panel** class is a concrete subclass of Container.
- It does not add any new methods, but it simply implements Container.
- **Panel** is the super class for Applet. When screen output is directed to an applet, it is drawn on the surface of the Panel object.
- **Panel** is a window that does not contain a title bar, menu bar, or border. Since you don't see these items when an applet is run inside a browser.
- Other components can be added to a Panel object by its add () method. You can position and resizing the components by using setLocation (), setSize (), or subtends () methods defined by the Component.

Window:

- The **Window** class creates a top-level window. It sits directly on the desktop.
- You can't create Window objects directly. But you will use a subclass of Window called **Frame**.

4.Explain the AWT Listeners.

ActionListener:

Implemented by objects that handle ActionEvent events.When implementing ActionListener it must override method called

public void actionPerformed(ActionEvent e)

BCSM IT DEPT

ItemListener:

Implemented by objects that handle ItemEvent event. When implementing ItemListener it must override a menthod called

public void itemStateChange(ItemEvent e)

AdjustmentListener

Implemented by objects that handle A AdjustmentEvent event. Invoked when the value of adjustable has change.

public void adjustmentValueChanged(AdjustmentEvent e)

ComponentListener

Implemented by objects that handle ComponentEvent event. Invoked when the component's size changes.

public void componentResized(ComponentEvent e)

Invoked when the component's position changes.

Public void componentMoved(ComponentEvent e)

Invoked when the component has been made visible.

Public void componentShown(ComponentEvent e)

Invoked when the component has been made invisible.

Public void componentHidden(ComponentEvent e)

ContainerListener

Implemented by objects that handle ContainerEvent event.

Invoked when a component has been added to the container.
Public void componentAdded(ContainerEvent e)
Invoked when a component has been removed from the container.
Public void componentRemoved(ContainerEvent e)

FocusListener

Implemented by objects that handle FocusEvent event.

Invoked when a component gains the keyboard focus.
Public void focusGained(FocusEvent e)

Invoked when a component loses the keyboard focus.

BCSM,IT DEPT

Public void focusLost(FocusEvent e)

KeyListener

Implemented by objects that handle KeyEvent event.
Invoked when a key has been typed.

Public void keyTyped(KeyEvent e)

Invoked when a key has been pressed.

Public void keyPressed(KeyEvent e)

Involed when a key has been released.

Public void keyReleased(KeyEvent e)

MouseListener

Implemented by objects that handle clicking-related MouseEvent events.

Invoked when the mouse has been clicked on a component.

Public void mouseClicked(MouseEvent e)

Invoked when a mouse button has been presswd on a component.

Public void mousePressed(MouseEvent e)

Invoked when a mouse button has been released on a component.

Public void mouseReleased(MouseEvent e)

Invoked when the mouse enter the a component.

Public void mouseEntered(MouesEvent e)

Invoked when tah mouse exits a component.

Public void mouseExited(MouseEvent e)

MouseMotionListener

Implemented by objects that handle movementrelated ItemEvent event.

Invoked when a mouse button is pressed on a component and then dragged.Mouse drage events will continue to be delivered to the component where theFirst originated until the mouse button is released(rdgardless of whether theMouse position is within the bounds of the component).

Public void mouseDragged(MouseEvent e)

Invoked when the mouse button has been moved on a component(with on buttons no down).

Public void mouseMoved(MouseEvent e)

BCSM IT DEPT

TextListener

Implemented by objects that handle TextEvent event.

Invoked when the value of the text has changed.The code written for this method performed the operations that need to occur when text changes.

Public void textValueChanged(TextEvent e)

WindowListener

Implemented by objects that handle WindowEvent event.

Invoked the first time a window is made visible.

Public void windowOpened(WindowEvent e)

Invoked when the user attempts to close window from the window's system menu.If the program does not explicity hide or dispose the window while processing this event ,the window close operation will be cancelled.

Public void windowClosing(WindowEvent e)

Invoked when a window has been closed as the result of calling dispose on the window.

Public void windowClosed(WindowEvent e)

Invoked when a window is changed from a normal to a minimized state. For many platforms, a minimized window is displayed as icon specified in the window's iconImage property.

Public void windowIconified(WindowEvent e)

Invoked when a window is change from a minimized to a normal state.

Public void windowDeiconified(WindowEvent e)

Invoked when tah window is set to be the user's active window,which the window (or one of its subcomponents)will receive keyboard events.

Public void windowActivated(WindowEvent e)

Invoked when a window is no longer tah user's active window,which ,means that Keyboard events will no longer be delivered to the window or its subcomponents.

Public void windowDeactivated(WindowEvent e)

As a convenience, the java.awt.event package provides adapter classes that implement the event listener interface. These classes may be subclassed to override specific eventhandling methods of interest.

BCSM,IT DEPT

5.**Explain in detail about LAYOUT MANAGERS:**

- A layout manager automatically arranges your controls within a window.
- The layout managers are used to organize the controls or components.
- Layout managers are placed inside a Frame or Applet or Panel

To Set Layouts:

- **setLayout** (null); Null layout cancels the container's layout. So users must manually placed the components by using the setBounds () method.
- **setLayout** (new LayoutManager) To set the layout model. If no call to setLayout () method, then the default layout manager is used.

- setBounds (int left, int top, int width, int height); Set the components position and size.
  Ex: - Label L=new Label ();
  L.setBounds (100,100,75,25);

FLOWLAYOUT: Default layout for Applets and Panel.

Constructors:

- **FlowLayout** (); Default constructor, which centers components and leaves 5 pixels of space between each component.
- **FlowLayout** (Align); Specify how each line is aligned.
- **FlowLayout** (Align, int hoz, int ver); Specify the horizontal and vertical space between each component.

Align : 1. FlowLayout.LEFT 2. FlowLayout.RIGHT 3. FlowLayout.CENTER

Container class methods:
1. setLayout (new Layout (…)); 2. add (component) 3. remove (component)

BORDERLAYOUT: Default layout for Frames.

Constructors:

- **BorderLayout** (); Set the default border layout.
- **BorderLayout** (int hor, int ver) Specify horizontal and vertical space between components.

Container class methods: 1. setLayout (new BorderLayout(…)); 2. add (Component,Direction);

BCSM, IT DEPT

Direction: 1. BorderLayout.NORTH 2. BorderLayout.SOUTH 3. BorderLayout.EAST
4. BorderLayout.WEST 5. BorderLayout.CENTER

GRIDLAYOUT: Same width and height for all cells.

Constructors:

- **GridLayout** () Creates a single-column grid layout.
- **GridLayout** (int row, int col) Creates a grid layout with the specified number of rows and columns.
- **GridLayout** (int row, int col, int Hspace, int Vspace)
  Specify the horizontal and vertical space between components.
  If row is zero, then number of columns will be unlimited length.
  If col. is zero, then number of rows will be unlimited length.

Container class methods: 1. setLayout (new GridLayout(…)); 2. add(Component);

CARDLAYOUT: Stores several different layouts.
Deck: - Collection of cards.

Constructors:

- **CardLayout** () Create default card layout.
- **CardLayout** (int hor, int ver) Specify the horizontal and vertical space between components.

Methods: **1.** void first(container) 2. void next(container) 3. void previous(container)
4. void last(container) 5. void show(container, cardName)

Container class methods: 1**.** setLayout (new CardLayout(…)); 2.add (component, Name);

Font class:

Font (String fontName, int fontStyle, int Size) where fontName is name of the font.

Style: 1. Font.PLAIN 2. Font.BOLD 3. Font.ITALIC

void setFont(Font fontObject) where fontObject contain the desired font.
Ex: - setFont (new Font (“Courier”, Font.BOLD | Font.ITALIC, 20));

BCSM, IT DEPT

6.Discuss about Handling Events by Extending AWT Components:

- Java allows you to handle events by extending the AWT components.
- To extend an AWT component, you must call the enableEvents () method of Component.
- G.F: - protected final void enableEvents (long eventMask)

(A)LABEL:

- The **Label** contains a string to be displayed.
- Do not support any interaction with the user.

Constructors:

- **Label** () Create a blank label.
- **Label** (String STR) Creates a label that contains the string specified by STR.
- **Label** (String STR, int align) Alignment may be Label.LEFT or Label.RIGHT or Lable.CENTER.

Methods:

- **void setText** (String STR) Set or change the text in a label. Where STR specify the new label.
- **String getText** () Returns the current label.
- **void setAlignment** (int align) Set the alignment of the string within the label.
- **int getAlignment** () Returns the current alignment of the string within the label.

(B)BUTTON:

- The most widely used control is the **push button**. A push button is a component that contains a label and that generate an event when it is pressed.

Constructors:

Button () Creates an empty button.
Button (String caption) Creates a button that contains caption as a label.

Methods:

void setLabel (String Str) Sets the button label by Str.
String getLabel () Return the label of the button.

Handling Events:

BCSM,IT DEPT

- Each time a button is pressed, an action event is generated. Each listener implements the **ActionListener** interface.
- When events occur the **actionPerformed** () method will be invoked. An **ActionEvent** object is the argument of this method.
- The label is obtained by calling the **getActionCommand** () method on the ActionEvent object.

(C)TEXTFIELD:

- The **TextField** class implements a single line text-entry area. (Also called as edit control).
- Allow the user to enter strings and to edit the test using the arrow keys, cut and paste keys and mouse selections.

Constructors:

- **TextField** ( ) Creates a default text field.
- **TextField** (int numChars) Creates a text field with width specified by numChars.
- **TextField** (String Str) Initialize the text field with the string specified by Str.
- **TextField** (String Str, int numChars) Initialize the text field by Str and set its width by numChars.

Methods:

- **String getText()** Returns the string currently available in the text field.
- **void setText(String Str)** Set the text with the value specified by Str.
- **String getSelectedText ()** Returns the selected text only.
- **void select(int stIndex, int enIndex)** Select the characters beginning at stIndex and ending at enIndex-1.
- **boolean isEditable ()** Returns true if the text may be changed.
- **void setEditable(boolean flag)** If the flag **true**, then the text may be changed. Otherwise the text can't be changed.
- **void setEchoChar(char ch)** Set echo character to the text field. (Such as password)
- **boolean echoCharIsSet ()** Check a text filed whether it is in echoChar mode or not.
- **char getEchoChar ()** Returns the current echo character of a text field.

(D)TEXTAREA:

The **TextArea** class implements a multi-line text-entry area. Allow the user to enter strings and to edit the test using the arrow keys, cut and paste keys and mouse selections.

BCSM,IT DEPT

Constructors:

- **TextArea** () Creates a default text area.
- **TextArea** (int numLines, int numChars) Create a text area with height numLines and width numChars.
- **TextArea** (String Str) Initialize the text area with the string specified by Str.
- **TextArea** (String Str, int numLines, int numChars) Initialize the text area by Str and set its height by numLines and width by numChars.
- **TextArea** (String Str,int numLines, int numChars,int sBar) where sBar specify the scrollbar control.

sBar must be one these values:

1. SCROLLBARS_BOTH 2. SCROLLBARS_NONE
3. SCROLLBARS_HORIZONTAL_ONLY 4. SCROLLBARS_VERTICAL_ONLY

Methods:

- It supports the getText (), setText (), getSelectedText (), select (), isEditable () and setEditable () methods.
- void **append** (String Str) Append the string Str to the end of the text.
- void **insert** (String Str, int Index) Insert the string Str at the location specified Index.
- void **replaceRange** (String Str, int stIndex, int enIndex) Replace the char's from stIndex to enIndex-1 by Str.

7.What is Event Handling? Explain.

- Applets inherit a group of event-handling methods from the Container class.
- The Container class defines several methods, such as processKeyEvent and processMouseEvent, for handling particular types of events, and then one catch-all method called processEvent.

In order to react an event, an applet must override the appropriate event-specific method.

```
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
import java.applet.Applet;
import java.awt.Graphics;

public class ExampleEventHandling extends Applet
                                implements MouseListener {

    StringBuffer strBuffer;
```

BCSM, IT DEPT

```
public void init() {
  addMouseListener(this);
  strBuffer = new StringBuffer();
    addItem("initializing the apple ");
}

public void start() {
    addItem("starting the applet ");
}

public void stop() {
    addItem("stopping the applet ");
}

public void destroy() {
    addItem("unloading the applet");
}

void addItem(String word) {
    System.out.println(word);
    strBuffer.append(word);
    repaint();
}

public void paint(Graphics g) {
  //Draw a Rectangle around the applet's display area.
    g.drawRect(0, 0,
              getWidth() - 1,
              getHeight() - 1);

  //display the string inside the rectangle.
    g.drawString(strBuffer.toString(), 10, 20);
}


public void mouseEntered(MouseEvent event) {
}
public void mouseExited(MouseEvent event) {
}
public void mousePressed(MouseEvent event) {
}
public void mouseReleased(MouseEvent event) {
}

public void mouseClicked(MouseEvent event) {
  addItem("mouse clicked! ");
```

BCSM, IT DEPT

```
    }
}
```

Now, let us call this applet as follows:

```
<html>
<title>Event Handling</title>
<hr>
<applet code="ExampleEventHandling.class"
width="300" height="300">
</applet>
<hr>
</html>
```

- Initially, the applet will display "initializing the applet. Starting the applet." Then once you click inside the rectangle "mouse clicked" will be displayed as well.

Displaying Images:

An applet can display images of the format GIF, JPEG, BMP, and others. To display an image within the applet, you use the drawImage() method found in the java.awt.Graphics class.

Following is the example showing all the steps to show images:

```
import java.applet.*;
import java.awt.*;
import java.net.*;
public class ImageDemo extends Applet
{
  private Image image;
  private AppletContext context;
  public void init()
  {
     context = this.getAppletContext();
     String imageURL = this.getParameter("image");
     if(imageURL == null)
     {
       imageURL = "java.jpg";
     }
     try
     {
       URL url = new URL(this.getDocumentBase(), imageURL);
       image = context.getImage(url);
     }catch(MalformedURLException e)
```

BCSM,IT DEPT

```
                {
                  e.printStackTrace();
                  // Display in browser status bar
                  context.showStatus("Could not load image!");
                }
              }
              public void paint(Graphics g)
              {
                context.showStatus("Displaying image");
                g.drawImage(image, 0, 0, 200, 84, null);
                g.drawString("www.javalicense.com", 35, 100);
              }
}
```

Now, let us call this applet as follows:

```
<html>
<title>The ImageDemo applet</title>
<hr>
<applet code="ImageDemo.class" width="300" height="200">
<param name="image" value="java.jpg">
</applet>
<hr>
</html>
```

Playing Audio:

An applet can play an audio file represented by the AudioClip interface in the java.applet package. The AudioClip interface has three methods, including:

- **public void play():** Plays the audio clip one time, from the beginning.
- **public void loop():** Causes the audio clip to replay continually.
- **public void stop():** Stops playing the audio clip.

To obtain an AudioClip object, you must invoke the getAudioClip() method of the Applet class. The getAudioClip() method returns immediately, whether or not the URL resolves to an actual audio file. The audio file is not downloaded until an attempt is made to play the audio clip.

Following is the example showing all the steps to play an audio:

```
import java.applet.*;
import java.awt.*;
import java.net.*;
public class AudioDemo extends Applet
{
  private AudioClip clip;
```

BCSM,IT DEPT

```
        private AppletContext context;
        public void init()
        {
          context = this.getAppletContext();
          String audioURL = this.getParameter("audio");
          if(audioURL == null)
          {
            audioURL = "default.au";
          }
          try
          {
            URL url = new URL(this.getDocumentBase(), audioURL);
            clip = context.getAudioClip(url);
          }catch(MalformedURLException e)
          {
            e.printStackTrace();
            context.showStatus("Could not load audio file!");
          }
        }
        public void start()
        {
          if(clip != null)
          {
            clip.loop();
          }
        }
        public void stop()
        {
          if(clip != null)
          {
            clip.stop();
          }
        }
}
```

*******END********

BCSM,IT DEPT