B.Sc (Computer Science)
Programming in Java
Unit-II

1. What is a Constructor?

Java supports a special type of method called **Constructor** that enables an object to initialize itself when it is created.

```
class Rectangle
{
  int length;
  int width;

  Rectangle(int x,int y)
  {
   length=x;
   width=y;
  }

}
```

Special Characters:

- Constructors have the same name as the class itself.
- They don't have any return type, not even void. This is because they return the instance of the class.

2. What is the need for copy Constructor?

Copy Constructor is a constructor that takes the object of same class as argument. If all the properties of an object which has to be assigned to another object, this is advisable.

```
Box b1=new Box(5);
Box b2=new Box(b1);
```

3. Explain about inheritance in Java.

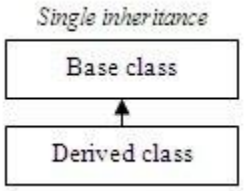
Java classes can be reused in several ways. This is basically done by creating new classes, reusing the properties of existing ones. The mechanism of deriving a new class from old one is called inheritance. The old class is known as **base class** or **super class** or **parent class** and the new one is called the subclass or **derived class** or **child class**.

The inheritance allows subclasses to inherit all the variables and the methods of their parent classes.

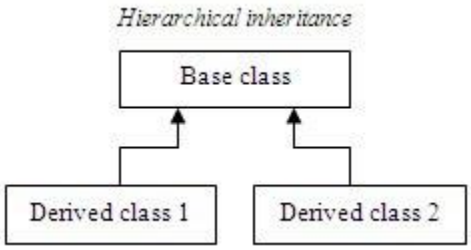
Inheritance may take different forms:

- Single inheritance (only one super class)
- Multiple inheritance (several super class)
- Hierarchical inheritance (one super class, many subclasses)
- Multilevel inheritance (derived from a derived class)

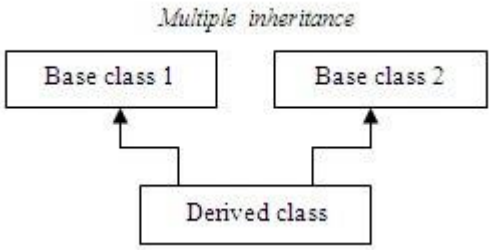
Single inheritance: If a class is derived from another class, it can be called as *single* inheritance.



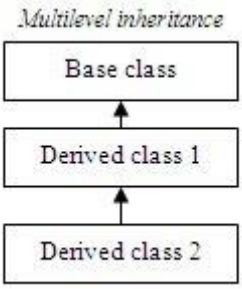
Hierarchical inheritance: If two or more classes are derived from a class, it can be called as *hierarchical* inheritance.



Multiple inheritance: If a class is derived from two or more classes, it can be called as *multiple* inheritance. Java does not support this type of inheritance.



Multilevel inheritance: (Oct 2012) If a class is derived from the class, which is derived from another class; it can be called as *multilevel* inheritance.



Defining a subclass: A subclass is defined as follows:

```
class subclass extends superclass
{
        Variable declarations;
        Methods declarations;
}
```

The keyword extends signifies that the properties of the superclass are extended to the subclass. The subclass will now contain its own variables and methods as well those of the superclass. This kind of situation occurs when we want to add some more properties.

4. Difference between Overloading and Overriding methods. (Mar 2010)

Method overloading:-

Methods are created with the same name but different parameter list and different definitions. This is called **method overloading.**

- Method overloading is used when objects are required to perform similar task but using different. Input parameters.
- When we call a method in an object, java matches up the method name first and then the number of parameters and then type of parameters to decide which one of the definitions to execute. This process is known as polymorphism.
- In the below example, constructor overloading is used.

```
class Room
{
float length;
float breadth;
Room(float x,float y)
{
length=x;
breadth=y;
}
Room(float x)
{
length=breadth=x;
}
Int area()
{
Return length*breadth;
}
```

Method overriding:-

A method defined in a super class can be inherited by its subclass and is used by the objects created by its subclass. Method inheritance enables us to define and use methods repeatedly in subclasses without having to define the methods again in the subclass. However, there may be occasions when an object to respond to the same method but have different behavior when that method is called. That means, the super-class method should be override. This is possible by defining a method in the subclass that has the same name, same arguments and same return type as a

method in the super-class. When that method is called, the method defined in the subclass is invoked and executed instead of the one in the super-class. This is known as overriding.

```
class A
{
int i, j;
A(int a, int b) {
i = a;
j = b;
}
void show() {
System.out.println("i and j: " + i + " " + j);
}
}

class B extends A {
int k;
B(int a, int b, int c) {
super(a, b);
k = c;
}
void show() {
System.out.println("k: " + k);
}
}

class OverrideMethod {
public static void main(String args[]) {
B b = new B(1, 2, 3);
b.show();
A a=new  A(10,20);
a.show();
}
}
```

5. When do we declare a method or class as final? (Mar 2011)

All methods and variables can be overridden by default in subclasses. A class that cannot be sub classed is called final class. This is achieved in Java using the keyword final as follows:

```
final class A
{

Body of the class

}
```

Making a method final ensures that the functionality defined in the method will never be altered in any way. Similarly, the value of a final variable can never be changed.

6. When do we declare a method or class as abstract?

Java allows the programmer to override a method compulsory. This can be done using the keyword abstract in the method defined.

```
abstract class Shape
{
……..
……..
abstract void draw();
……
……..
}
```

When a class contains atleast one abstract method, it should be declared as abstract.
When using the abstract classes, we must satisfy the following conditions:

- We cannot use abstract classes to instantiate objects directly. For example Shape s=new Shape() is illegal because Shape is an abstract class.
- The abstract methods of an abstract class must be defined in its subclass.
- We cannot declare abstract constructors or abstract static methods.

7. Define Array.

An array represents a group of elements of same data type. It can store a group of elements.
In java arrays are created on dynamic memory i.e. allocated at runtime by JVM.
Arrays are generally categorized into two parts as follows:

- Single dimensional arrays.
- Multi dimensional arrays.

Single dimensional arrays:-

- A list of items can be given one variable name using only one subscript. Such a variable is called single dimensional array.
- Arrays must be declared and created in the computer memory before they are used
- Creation of array involves 3 steps:
  1) Declaring the array
  2) Creating memory locations
  3) Initialization of values into the memory locations.

Declaring the array:-

Arrays in java may be declared in two forms:

```
type arrayname[ ];
        (or)
type[ ] arrayname[ ];
```

Ex:-

```
int counter[ ];
int[ ] counter;
```

Creating memory location to array:-

After declaration of arrays, memory locations are created by new operator.

```
arrayname = new   type[size];

Ex:  counter =new int[5];
```

These lines create necessary memory locations for the array counter. It is also possible to combine the two steps declaration and creation of memory location as follows:

```
int number[ ] = new int[5];
```

Initialization of arrays:-

- Putting values into the array is known a "initialization". This is done as follows:

```
arrayname[subscript]=value;
```

Ex:-

```
counter[0]=35;
```

- Java creates arrays starting with the subscript of 0 and ends with a value one less than the size specified.
- Arrays can also be initialized automatically when they are declared, as shown below:
  Syntax:- type arrayname[ ]={list of values};

  Ex:-

```
int number[ ]={5,4,1,8,9};
```

  In the above example array size is not given, the compiler allocates enough space for all elements specified in the list.

Multi-dimensional arrays:-

Multidimensional arrays can represent the data in the form of rows and columns i.e. by taking two or more indexes.

Multidimensional arrays include double dimensional and 3 dimensional arrays.

2-Dimensional Arrays: 2-Dimensional Arrays can represent the data in the form of rows and columns i.e. by taking two indexes. Generally, these are used to work with matrices.

In java, a double dimensional array can be declared as follows:

```
int a[][] = new int[5][5];
```

Advantages of arrays:

- It is capable of storing many elements at a time
- It allows random accessing of elements i.e. any element of the array can be randomly accessed using indexes.

Disadvantages:

- Predetermining the size of the array is a must.
- There is a chance of memory wastage or shortage.
- To delete one element in the array, we need to traverse throughout the array.

Matrix Multiplication(Two-Dimensional Array):

```
import java.io.*;
class matrixmult
{
public static void main(String args[]) throws IOException
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
int m1[][]=new int[3][3];
int m2[][]=new int[3][3];
int m3[][]=new int[3][3];
int i,j,k;
System.out.println("Enter elements of first matrix ");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
m1[i][j]=Integer.parseInt(br.readLine());
}
}

System.out.println("Enter elements of second matrix ");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
m2[i][j]=Integer.parseInt(br.readLine());
}
}


System.out.println(" elements of first matrix are ....");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
System.out.print(m1[i][j]);
}
System.out.println();
}

System.out.println("Elements of second matrix ");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
System.out.print(m2[i][j]);
}
System.out.println();
}


/*Matrix multiplication*/
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
```

```
{
m3[i][j]=0;
for(k=0;k<3;k++)
{
//m3[i][j]+=m1[i][k]*m2[k][j];
m3[i][j]=m3[i][j]+m1[i][k]*m2[k][j];
}
}
}

System.out.println("matrix multiplication result is ");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
System.out.print(m3[i][j] +" ");
}
System.out.println();
}
}
}

class VectorTest
{
public static void main(String as[])
{
Vector v1=new Vector(10);
v1.add("10");
v1.add("20");
v1.add("30");
System.out.println("Number of Elements:"+v.size());
System.out.println("Vector Elements:"+v1);
}
}
```

8. Define Interfaces.

Interfaces: Java provides an approach known as interfaces to support the concept of multiple Inheritance. An interface contains methods and variables. All methods of the interface are public and abstract. Interfaces do not specify any code to implement these methods. Therefore it is the responsibility of the class that implements an interface to define the code for implementation of these methods.

The general form of an interface definition is as follows:

```
interface InterfaceName
{
Variables declaration;
Methods declaration;
}
```

An interface variable is public, static and final by default. This means all the variables of the interface are constants. Methods declaration will contain only a list of methods without anybody statements.

Interfaces can be implemented in two ways:

1. **Extending interfaces**

Interfaces can also be extended. The sub interface will inherit all the members of the super interface. This is achieved by using the keyword extends.

```
interface interfacename2 extends interfacename1
{
body of name2
}
```

Sub interfaces can not define the methods declared in the super interfaces. So class is
used to implement the derived interface, to define all the methods.

2. **Implementing interfaces:-**

Class must be defined to implement the code for the method of interface.

```
class classname implements interfacename
{
body of classname;
}
```

Here classname **"implements"** the interface interfacename. A class can implements more than one interface, separated by comma.

Example:

```
interface Area
{
final static double pi=3.14;
double compute (double x, double y);
}

class Rectangle implements Area
{
public double compute(double x, double y)
{
return(x*y);
}
}

class Circle implements Area
{
public double compute(double x, double y)
{
return(pi*x*x);
}

class InterfaceDemo
{
public static void main(String args[])
{
Rectangle r=new Rectangle();
System.out.println("Area of rectangle is"+r.compute(10,20));
```

```
Circle c=new Circle();
System.out.println("Area of circle is"+c.compute(5.1,0);
}
}
```

9. What is the difference between an interface & a class?

Class	Interface
A class must be declared using the keyword **class**	An interface must be declared using the keyword interface
A class contains the methods which are defined.	An interface consists of methods which are declared.
A class can implement many interfaces	An interface can extend another interface
A class can simultaneously extend a class and implement multiple interfaces	An interface is that which can simulate multiple inheritance.
A class which implements an interface must implement all of the methods declared in the interface or be an abstract class.	An interface is implicitly abstract. It cannot be directly instantiated except when instantiated by a class .
Methods of a class can use any access specifier.	In an interface, all methods are implicitly public
Variables of a class can be defined according to the requirement	In an interface, all variables are static and final by default.

10. What is a package?

Package is a Java's way of grouping a variety of classes and/or interfaces together. Upon importing a package in a program, it will be able to access the classes defined in that package.

Creating packages is way of achieving the reusability in Java. The grouping is usually done according to functionality. With the help of packages, we can use classes from other programs without physically copying them into the program under development. Packages concept is similar to "class libraries" in other languages.

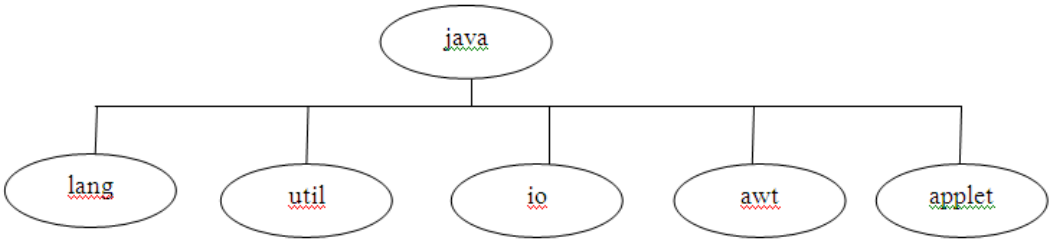
By organizing our classes into packages we achieve the following benefits:

1. The classes contained in the packages of other programs can be easily reused.

2. In packages, classes can be unique compared with classes in other packages. That is, two classes in two different packages can have the same name. They may be referred to by their name, comprising the package name and the class name.
3. Packages provide a way to "hide" classes thus preventing other programs or packages from accessing classes that are meant for internal use only.

4. Packages also provide a way for separating "design" from "coding". First we can design classes and decide their relationships, and then we can implement the Java code needed for the methods. It is possible to change the implementation of any method without affecting the rest of the design.

11. Explain the process of using Java API Packages & System Packages



java.lang	This package has primary classes and interfaces essential for java program. It consists of wrapper classes, strings, threads, etc. The classes defined in this package are imported by default.
java.util	(utility) in this package utility classes like data structures are available. This package contains useful classes and interfaces like stack, linked list, hash table, arrays. etc.
java.io	This package has file handling classes which can be used to perform input output related tasks
java.awt	Abstract window tool kit. This package helps to develop GUI. It consists of classes for windows, buttons, lists and etc.
java.applet	Classes for creating and implementing Applets.

There are two ways of accessing the classes stored in packages:

- Using fully qualified class name
- Import package and use class name directly.

1. **Using fully qualified class Name:**

   This is done by using the package name containing the class and then appending the class name to it using dot operator.

   Ex: java.util.Vector

2. **Import package and use class name directly:**

   a) **Importing a particular class in a package:** import java.util.Vector;

   b) **Importing all the classes in a package:** import java.util.*;

12. Explain the creation and accessing of a Package.

Creating Package:
Creating a package is quite easy. First declare the name of the package using the 'package' key word followed by a package name. This statement defines a name space in which classes are stored. It is followed by package definition. Here is an example.

Package first package // package declaration

Public class Firstclass // class definition

{
Body of the class
}
Here the package name is first package. The class Firstclass now a part of the package. The listing would be saved as a file called Firstclass.java. It is located in a directory named firstpackage. When the source file is compiled, Java will create a class file and store it in the same directory.

Creating the package involves the following steps:

1. Declare the package at the beginning of a file using the form

Package Packagename;

2. Define the class that is to be put in the package and declare it **public**.

3. Create a subdirectory under the directory where the main source files are stored.

4. Store the listing as the classname.java file in subdirectory.

5. Compile the file. This creates **.class** file in subdirectory.

Java also supports the concept of package hierarchy. Specifying multiple names in a package statement separated by periods does this.

The general form of a multileveled package statement is shown here

Package pkg1.pkg2;

Accessing a Package:

A Java system package can be accessed either using a fully qualified class name or through the

'import' statement. Import statement is used when

1. There are too many references to a particular package.

2. When the package name is too long.

The general form of 'import' statement for searching a class is as follows

import package1[.package2] [.package3].class name;

where package1 is the name of the top level package, package2 is the name of package inside package1 and so on. Finally, 'class name' is specified explicitly.

Example:

import firstPackage.SecondPackage.Myclass ;

Another Approach : import.packagename.*;

Packagename can denote a single package or a hierarchy of packages. The star(*) indicates that the compiler should search this entire package hierarchy.

13. How to add a class to a Package?

Consider the following package:

```
package p1;
public class A
{
Body of the class
}
```

The package p1 contains one public class by name A. Suppose we want to add another class B to this package. This can be done as follows:

1. Define the class and make it public
2. Place the package statement

```
package p1;
public class B
{
Body of B
}
```

3. Store this in B.java under the directory p1.
4. Compile B.java file. This will create a B.class and place it in the directory p1.

Now, the package p1 will contain both the classes A and B.
A statement like

```
import p1.* ;
```

will import both of them.

A java source file can have only one class declared as public, we cannot put two or public classes together in a .java file. This is because of the restriction that the file name should be same as the name of the public class with .java extension.

If we want to create a package with multiple public classes in it, then steps are as follows.

1. Decide the name of the class.
2. Create a subdirectory with this name under the directory where main source files are stored.
3. Create classes that are to be placed in the package in **separate source files** and declare the package statement, package packagename at the top of each source file.
4. Switch to subdirectory earlier and compile each source file. When completed, the package would contain .class files of all the source files.

14. Explain String methods in Java.

String is a set of characters. Java provides a class named "String" to handle strings. The "String" class has several built-in functions to support different

operations on strings. Though String is a class, the instantiation of the objects need not be done using the keyword 'new'. The string value can be directly assigned to String object. A java String is not a character array and it is not terminated by null character.

String can be initialized in many ways.

· String s1 = "IIMC";
· String s2 = new String("ABC");
· String s3 = new String(new char[] {'a','b','c'});

1)toLowerCase():-

- This method converts the string into LowerCase and this method returns string type.

Syntax;- String s1="Bsc";
String s2;
s2=s1.toLowerCase();

2)toUpperCase();

- It converts the string to Uppercase. It also returns String type.

Ex:- String s2,s1="system";
s2 =s1.toUpperCase();

3)length();

- This method returns length of a string that means number of character of a string.

Ex:- String s1="computer":
int l=s1.Length();

4)replace():

- It replaces all appearances of x with y in s1 string.

Syntax:- s1.replace('x','y');

5)trim():-

- This method remove white spaces at the beginning and ending of the string s1.

Ex:- s2=s1.trim();

6)equals():-

- This method returns true if two strings are same otherwise it returns false. This is case sensitive.

Ex:- s1.equals(s2);

7)equalsIgnoreCase():-

- It returns true if both string are true. If ignores the case of characters.

Ex:- s1.equalsIgnoreCase(s2);

8)charAt():-

- This method returns the character at the specified position n.

Syntax:- s1.charAt(n)

9) compareTo():-

- It returns negative if s1<s2, or it returns positive if s1>s2 or it returns zero if s1 is equal to s2.

- It is used as follows:-

S1.compareTo(s2);

10)concat():-

- This method concatenates two strings.

Ex:- s1.concat(s2);

11)substring(n):-

- This method gives substring starting from nth character.

Ex:- s1.substring(5) returns characters starting from 5th character till the end of s1.

12)substring(n.m):-

- It gives substring starting from nth character upto mth character(not including mth character).

Ex:- s1.substring(5,10) returns the characters of s1 starting from 5th to 9th positions.

13)indexOf():-

- It gives the position of the first occurance of 'x' in the string s1.

Ex:- s1.indexOf('x');

Ex:- The following program illustrated the methods of String class.

15. Explain StringBuffer class in Java.

StringBuffer class:-

- String class creates strings of fixed-length, StringBuffer class creates strings of flexible length that can be modified in terms of both length and content.
- We can insert characters and substrings in the middle of a string or append another string to the end, where StringBuffer class is used.
- The following declaration shows to create a string using StringBuffer class.

  StringBuffer sb=new StringBuffer("Hello");
- The following methods are used with String Buffers class.

1)length():-

- This returns the number of characters in the String Buffer object.

Ex:- StringBuffer sb=new StringBuffer("Welcome");
Sb.length();

2)append():-

- It appends the string s2 to s1 at the end.

Ex:- s1.append(s2);

3)insert():-

- It inserts the string s2 at the position n of the string.

Ex:- s1.insert(n.s2);

4)toString():-

- It converts the StringBuffer object into a String object

5)reverse():-

- This method reverses the sequence in the StringBuffer.

  sb.reverse();

6)indexOf():-

- It returns the first occurrence of substring str in the StringBuffer object.

Syntax:- indexOf(str);
Ex:- StringBuffer sb=new StringBuffer("this is a book");
int n=sb.indexOf("is");
it returns 2nd position. That means the first occurrence of "is" in the sb object.

7)setCharAt():-

- This method modifies the nth character to x.

Ex:- s1.setCharAt(n,'x');

16. How does String class differ from StringBuffer class?

Differences between String and StringBuffer are listed below.

1. StringBuffer objects must ne instantiated using 'new'. But, Strings can be instantiated without using 'new'.
2. StringBuffer objects are mutable (self changeable). Where as, String objects are immutable.
3. StringBuffer provides functions like setCharAt(), deleteCharAt() and etc.
4. StringBuffer allocates extra 16 bits of memory
5. StringBuffer can be treated as a dynamic string.

17. Explain Wrapper Classes.

Wrapper classes can be used to convert primitive data types like int, float, long, and double to objects. These classes are contained in the java.lang package. The classes Byte, Double, Float, Integer and Short extend the abstract class Number. All the wrapper classes are public final ie cannot be extended.

All the wrapper classes have two constructor forms.

- A constructor that takes the primitive type and creates an object eg Character(char), Integer(int).
- A constructor that converts a String into an object eg Integer("1"). But, this type of constructor is not available with Character class.

The methods of the wrapper classes are all static so we can use them without creating an instance of the matching wrapper class. Once assigned a value, the value of a wrapper class cannot be changed. The wrapper classes are just that classes, they are not primitives and instances of the wrappers can only be manipulated in the same way as any class.

The following table lists simple data types and their corresponding wrapper classes.

Simple Type	Wrapper class
boolean	Boolean
char	Character
double	Double
float	Float
int	Int
long	Long
byte	Byte
short	Short