UNIT – II

CLASSES, ARRAYS, STRINGS ANDVECTORS

CLASS **[2 Marks]**

Class is a user defined data type. It is a collection of data and methods.

A class is a group of objects that has common properties. It is a template or blueprint from which objects are created.

Syntax to declare a class:

```
class <class_name>
{
   data member; //properties
   method; // behavior
}
```

Method in Java

In java, a method is like function i.e. used to expose behavior of an object.

Advantage of Method

- Code Reusability
- Code Optimization

new keyword

The new keyword is used to allocate memory at runtime.

Fields Declaration

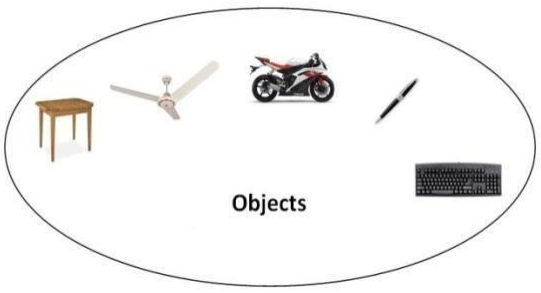
```
Class Student
{
        Int Rno;
        String name;
}
```

Method Declaration

```
Type methodname(parameter-list)
{
        Method Body;
}
```

```
class Student
{
        i
        n
        t
        R
        n
        o
        ;
        S
        tr
        i
        n
        g
        n
        a
        m
        e;

        void insertRecord(int r, String n)
        {
                R
                n
                o
                =
                r
                ;

                n
                a
                m
                e
                =
                n
                ;
        }
}
```

Object in Java



OBJECT

An object is a real time entity that has properties and behavior is known as an object e.g.

chair, bike, marker, pen, table, car etc. It can be physical or logical (tengible and intengibleAnobject has three characteristics: **[2 Marks]**

- **properties:** represents data (value) of an object.
- **behavior:** represents the behavior (functionality) of an object such as deposit, withdraw etc.

Let us now look deep into what are objects. If we consider the real-world we can find manyobjects around us, Cars, Dogs, Humans, etc. All these objects have a state and behavior.

If we consider a dog, then its state is - name, breed, color, and the behavior is - barking,wagging, running

If you compare the software object with a real world object, they have very similar characteristics.

Software objects also have a state and behavior. A software object's state is stored in fieldsand behavior is shown via methods.

So in software development, methods operate on the internal state of an object and the object-to- object communication is done via methods.

Classes in Java:

A class is a blue print from which individual objects are created. A

sample of a class is given below:

```
class Dog
{
  String breed;
  int age;
  String color;

  void barking()
  {
  }

  void hungry()
  {
  }

  void sleeping()
  {
  }
}
```

CREATING AN OBJECT: **[2 Marks]**

As mentioned previously, a class provides the blueprints for objects. So basically an object is created from a class. In Java, the new key word is used to create new objects.

There are three steps when creating an object from a class:

- **Declaration:** A variable declaration with a variable name with an object type.
- **Instantiation:** The 'new' key word is used to create the object.
- **Initialization:** The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Example of creating an object is given below:

```
class Puppy{

  Puppy(String name){
    // This constructor has one parameter, name.
    System.out.println("Passed Name is :" + name );
  }

  Public static void main(String []args){
    // Following statement would create an object myPuppy Puppy
    myPuppy = new Puppy( "tommy" );
  }
}
```

If we compile and run the above program, then it would produce the following result:

```
Passed Name is :tommy
```

Accessing Instance Variables and Methods:

Instance variables and methods are accessed via created objects. To access an instance variable the fully qualified path should be as follows:

```
/* First create an object */
ObjectReference = new Constructor();

/* Now call a variable as follows */
ObjectReference.variableName;

/* Now you can call a class method as follows */
ObjectReference.MethodName();
```

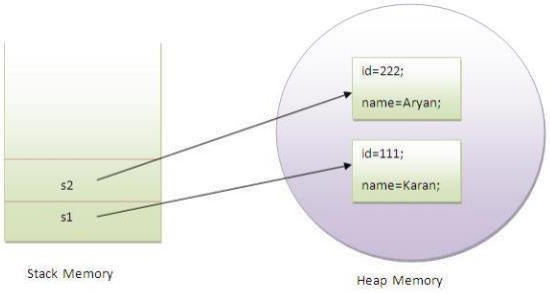
Example of Object and class that maintains the records of students

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method on it. Here, we are displaying the state (data) of the objects by invoking the displayInformation method.

```
   class Student2
{
    int rollno;
    String name;

    void insertRecord(int r, String n)
    {
     rollno=r; name=n;
    }

    void displayInformation()
   {
            System.out.println(rollno+" "+name);
   }
}
class Stud
{
    public static void main(String args[])
  {
```

```
    Student2 s1=new Student2();
    Student2 s2=new Student2();

    s1.insertRecord(111,"Karan");
    s2.insertRecord(222,"Aryan");

    s1.displayInformation();
    s2.displayInformation();
   }
  }
```

```
111 Karan
222 Aryan
```




```
    /* You can access instance variable as follows as well */ System.out.println("Variable
    Value :" + myPuppy.puppyAge );
  }
}
```

If we compile and run the above program, then it would produce the following result:

```
Name chosen is :tommy
Puppy's age is :2
Variable Value :2
```

CONSTRUCTORS IN JAVA **[5 Marks]**

Constructor in java is a special type of method that is used to initialize the object.
Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor.

Rules for creating java constructor

There are basically two rules defined for the constructor.

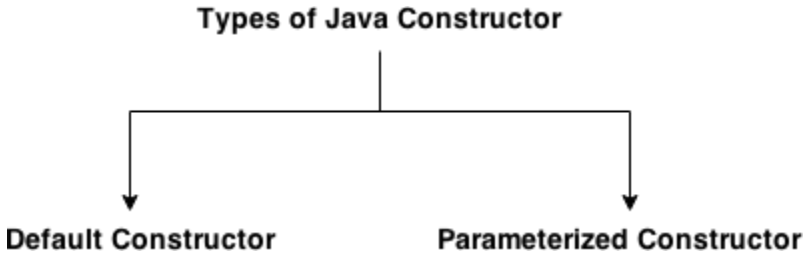
1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

Types of java constructors

There are two types of constructors:

1. Default constructor (no-arg constructor)

2. Parameterized constructor



1 Default Constructor

A constructor that has no parameter is known as default constructor. Syntax of default constructor:

```
<class_name>()
{

}
```

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
class Bike1
{
        Bike1()
        {
                System.out.println("Bike is created");
        }
}
```

```
class DefaultConstructor
{
        public static void main(String args[])
        {
                Bike1 b=new Bike1();
         }
 }
```

Output:

Bike is created

Rule: If there is no constructor in a class, compiler automatically creates a default constructor. default constructor

What is the purpose of default constructor?

Default constructor provides the default values to the object like 0, null etc. depending on the type. Example of default constructor that displays the default values

```
  class Student3{ int
  id;
  String name;

  void display(){System.out.println(id+" "+name);} public

  static void main(String args[]){
  Student3   s1=new   Student3();
  Student3   s2=new   Student3();
  s1.display();
  s2.display();
  }
  }
```

Output:

0 null
0 null

Explanation:In the above class,you are not creating any constructor so compiler provides you a default constructor.Here 0 and null values are provided by default constructor.

2 parameterized constructor
A constructor that has parameters is known as parameterized constructor.

Why use parameterized constructor?
Parameterized constructor is used to provide different values to the distinct objects.

Example of parameterized constructor
In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
class Student4
{
        int id;
        String name;

        Student4(int i,String n)
        {
                id = i;
                name = n;
         }
         void display(){System.out.println(id+" "+name);
}

class ParameterizedConstructor
{
        public static void main(String args[])
        {
                Student4 s1 = new Student4(111,"Karan");
                Student4 s2 = new Student4(222,"Aryan");
                s1.display();
                s2.display();
        }
}
```

Output:
111 Karan
222 Aryan

CONSTRUCTOR OVERLOADING IN JAVA **[5 Marks]**

Constructors having the same name but in parameter list is called constructor overloading

```
class Calculator
{
	int a,b;
	double d;
	Calculator()
	{
		a=0;
		b=0;
		System.out.println("The addition of " +a+ "+" +b+ " = " +(a+b));
	}

	Calculator(int x,int y)
	{
		a=x;
		b=y;
		System.out.println("The addition of " +a+ "+" +b+ " = " +(a+b));
	}

	Calculator(double x,int y,int z)
	{
		d=x;
		a=y;
		b=z;
		System.out.println("The addition of "+d+ "+" +a+ "+" +b+ " = "+(d+a+b));
	}
}

class ConstructorOverload
{
	public static void main(String args[])
	{
		Calculator obj1=new Calculator(); Calculator
		obj2=new Calculator(10,20); Calculator
		obj3=new Calculator(35.6,10,20);

	}
}
```

OUTPUT

```
C:\Program Files\Java\jdk1.7.0\bin>javac ConstructorOverload.java
C:\Program Files\Java\jdk1.7.0\bin>java ConstructorOverload
The addition of 0+0 = 0
The addition of 10+20 = 30
The addition of 35.6+10+20 = 65.6
```

METHOD OVERLOADING **[5 Marks]**

The methods that have the same name but differs in parameter list is called method overloading.

Method overloading is used when objects are required to perform similar tasks but using different input parameters.

```
class Geometry
{
        double width,height; int
            radius;

            void area(double x,double y)
            {
                width=x;
                height=y;
                double area=width*height; System.out.println("Area
                of Rectangle is = "+area);
            }

            void area(int x)
            {
                radius=x;
                double area=3.142*radius*radius;
                System.out.println("Area of Circle is = "+area);
            }
}

class Overloading
{
            public static void main(String args[])
            {
                Geometry g=new Geometry();
                g.area(10.2,15.3);
                g.area(5);
            }
}
```

Output

```
C:\Program Files\Java\jdk1.7.0\bin>javac Overloading.java
C:\Program Files\Java\jdk1.7.0\bin>java Overloading
Area of Rectangle is = 156.06
Area of Circle is = 78.55
```

JAVA STATIC KEYWORD **[5 Marks]**

The **static keyword** in java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

The static can be:

1. variable (also known as class variable)
2. method (also known as class method)
3. block

1) Java static variable **[2 Marks]**

If you declare any variable as static, it is known static variable.

- The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees,college name of students etc.
- The static variable gets memory only once in class area at the time of class loading.

Advantage of static variable

It makes your program **memory efficient** (i.e it saves memory).

Understanding problem without static variable

```
class Student
{
   int rollno;
   String name;
   String college="BCA";
}
```

Suppose there are 500 students in my college, now all instance data members will get memory each time when object is created. All students have its unique rollno and name so instance data member is good. Here, college refers to the common property of all objects. If we make it static, this field will get memory only once.

Java static property is shared to all objects.

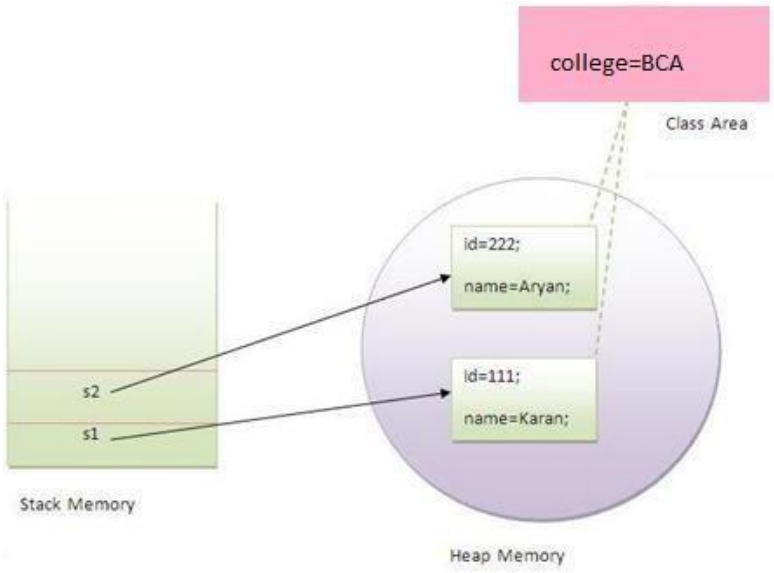
Example of static variable

```
//Program of static variable
class Student
{
      int rollno;
      String name;
      static String college ="BCA";
```

```
    Student(int r,String n)
      {
            rollno = r;
            name = n;
      }
     void display ()
     {
         System.out.println(rollno+" "+name+" "+college);
     }
}

    class StaticVariable
    {
         public static void main(String args[])
         {
                    Student s1 = new Student(111,"Karan");
                    Student s2 = new Student(222,"Aryan");

                    s1.display();
                    s2.display();
       }
    }
```

Output:

111 Karan BCA
222 Aryan BCA



2) Java static method **[2 Marks]**

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- static method can access static data member and can change the value of it.

Example of static method

```
//Program of changing the common property of all objects(static field).

class Student
{
   int rollno;
   String name;
   static String college = "ITS";

   static void change()
    {
            college = "BCA";
   }

   Student(int r, String n)
    {
            rollno = r;
            name = n;
   }

   void display ()
    {
            System.out.println(rollno+" "+name+" "+college);
    }
  }
class StaticMethod
{
   public static void main(String args[])
   {
     Student9.change();

     Student9 s1 = new Student9 (111,"Karan");
     Student9 s2 = new Student9 (222,"Aryan");
     Student9 s3 = new Student9 (333,"Sonoo");

     s1.display();
     s2.display();
     s3.display();
    }
}
```

Output: 111 Karan BCA
222 Aryan BCA

333 Sonoo BCA

Another example of static method that performs normal calculation

```
//Program to get cube of a given number by static method

class Calculate{
 static int cube(int x){
 return x*x*x;
 }

 public static void main(String args[]){
 int result=Calculate.cube(5);
 System.out.println(result);
 }
}
```

Output:125

Restrictions for static method

There are two main restrictions for the static method. They are:

1. The static method can not use non static data member or call non-static method directly.
2. this and super cannot be used in static context.

```
class A{
 int a=40;//non static

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

Output:Compile Time Error

Q) why java main method is static?

Ans) because object is not required to call static method if it were non-static method, jvm create object first then call main() method that will lead the problem of extra memory allocation.

---

3) Java static block **[2 Marks]**

- Is used to initialize the static data member.
- It is executed before main method at the time of classloading.

Example of static block

```
class A2
{
        static
        {
                System.out.println("static block is invoked");
        }
}

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

Output:static block is invoked
Hello main

INHERITANCE **[5 or 10 Marks]**

- **Inheritance** is one of the feature of Object-Oriented Programming (OOPs).
- Inheritance can be defined as the process where one class acquires the properties and methods of another class.
- In other words, the derived class inherits the properties and behaviors from the base class.
- The derived class is also called subclass and the base class is also known as super-class.
- The derived class can add its own additional variables and methods.
- These additional variable and methods differentiates the derived class from the base class.
- The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

Inheritance is a compile-time mechanism. A super-class can have any number of subclasses. But a subclass can have only one superclass. This is because Java does not support multiple inheritance.

The superclass and subclass have **"is-a"** relationship between them.

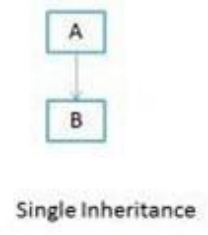
extends Keyword **[2 Marks]**

extends is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.

```
class Parent
{
  .....
  .....
}

class Child extends Parent
{
  .....
  .....

}
```

1) Single Inheritance **[2 Marks]**

When a one class extends another one class only then we call it a single inheritance.
The below flow diagram shows that class B extends only one class which is A. Here A is a **parent class** of B and B would be a **child class** of A.



Single Inheritance

Single Inheritance example program in Java

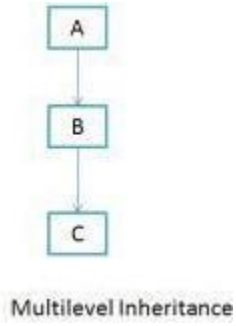
```
class A
{
  public void methodA()
  {
    System.out.println("Base class method");
  }
}

Class B extends A
{
  public void methodB()
  {
    System.out.println("Child class method");
  }
}

class SingleInheritance
{
  public static void main(String args[])
  {
    B obj = new B();
    obj.methodA(); //calling super class method
    obj.methodB(); //calling local method
 }
}
```

Output

```
C:\Program Files\Java\jdk1.7.0\bin>javac SingleInheritance.java
C:\Program Files\Java\jdk1.7.0\bin>java SingleInheritance
Base class method
Child class method
```

2) Multilevel Inheritance **[2 Marks]**

Multilevel inheritance The derived class inherits from one Base class, which intern inherits from some other class is called multilevel inheritance in Java.



Multilevel Inheritance

Multilevel Inheritance example program in Java

```
class A
{
  void methodA()
  {
    System.out.println("Class A method");
  }
}

class B extends A
{
        void methodB()
        {
                System.out.println("Class B method");
        }
}

class C extends B
{
        void methodC()
         {
                System.out.println("Class C method");
        }
}
```

```
class MultilevelInheritance
{
  public static void main(String args[])
  {
    C obj = new C();
   obj.methodA(); //calling grand parent class method
   obj.methodB(); //calling parent class method
   obj.methodC(); //calling child class method
  }
}
```

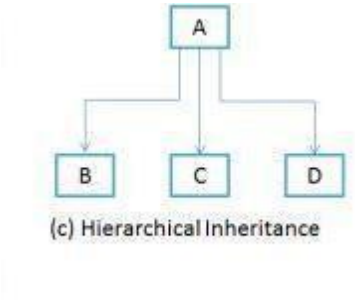
Output

```
C:\Program Files\Java\jdk1.7.0\bin>javac MultilevelInheritance.java
C:\Program Files\Java\jdk1.7.0\bin>java MultilevelInheritance Class A
method
Class B method
Class C method
```

3) Hierarchical Inheritance **[2 Marks]**

In such kind of inheritance one class is inherited by many **sub classes**. In below example class B,C and D **inherits** the same class A. A is **parent class (or base class)** of B,C & D.

Parent class have two or more than two child class then we call it as Hierarchical Inheritance.



Hierarchical Inheritance example program in Java

```
class A
{
  void methodA()
  {
    System.out.println("Class A method");
  }
}
```

```
class B extends A
{
        void methodB()
        {
                System.out.println("Class B method");
        }
}

class C extends A
{
        void methodC()
        {
                System.out.println("Class C method");
         }
}

class D extends A
{
        void methodD()
        {
                System.out.println("Class D method");
        }
}


class HierarchicalInheritance
{
  public static void main(String args[])
  {
          B obj1 = new B();
          C obj2 = new C();
          D obj3 = new D();
          obj1.methodA();          //calling Parent class methodA
          obj1.methodB();           //calling Child class methodB
          obj2.methodA();           //calling Parent class methodA
          obj2.methodC();           //calling Child class methodC
          obj3.methodA();           //calling Parent class methodA
          obj3.methodD();          //calling Child class methodD
  }
}
```

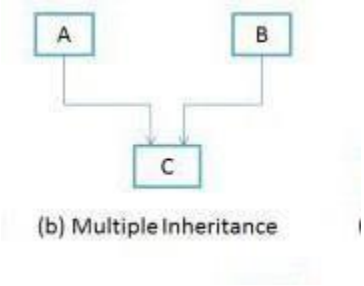
Output

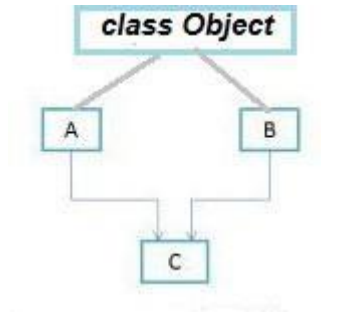
```
C:\Program Files\Java\jdk1.7.0\bin>javac HierarchicalInheritance.java
C:\Program Files\Java\jdk1.7.0\bin>java HierarchicalInheritance Class A
method
Class B method
Class A method
Class C method
Class A method
Class D method
```

4) Multiple Inheritance **[2 Marks]**

Why Java doesn't support multiple inheritance?

C++ and few other languages supports multiple inheritance while java doesn't support it. It is just to **remove ambiguity**, because **multiple inheritance** can cause ambiguity in few scenarios. One of the most common scenario is **Diamond problem.**





class Object is a Parent class of all the classes in java. So when you create a class the class Object is automatically added in the program and diamond shape in multiple inheritance.

5) Hybrid Inheritance

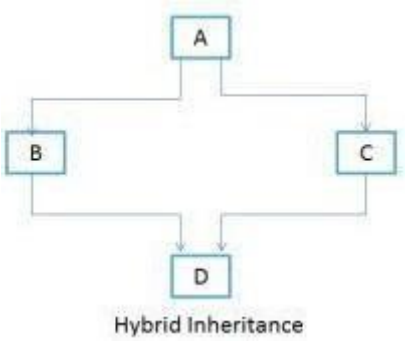
In simple terms you can say that Hybrid inheritance is a combination of **Single** and **Multiple inheritance.** A typical flow diagram would look like below. A hybrid inheritance can be achieved in the java in a same way as multiple inheritance can be!! Using interfaces. yes you heard it right. By using **interfaces** you can have multiple as well as **hybrid inheritance** in Java.

Does java supports hybrid inheritance?

Yes and No. If you are using only classes then this is not allowed in java, however using interfaces it's possible to have hybrid inheritance in java. We will see this in below example programs.

Hierarchical inheritance and Hybrid inheritance are different?

Yes, Hierarchical inheritance is different than hybrid inheritance. Hierarchical inheritance is possible to have in java even using the classes alone itself as in this type of inheritance two or more classes have the same parent class or in other words a single parent class has two or more child classes, which is quite possible to have in java.



Hybrid Inheritance

Hybrid Inheritance example program in Java

```
class A
{
   void methodA()
   {
       System.out.println("Class A methodA");
    }
}

class B extends A
{
    void methodA()
    {
       System.out.println("Child class B is overriding inherited method A");
    }

    void methodB()
    {
       System.out.println("Class B methodB");
     }
}

class C extends A
{
     void methodA()
    {
       System.out.println("Child class C is overriding the methodA");
     }
```

```
    void methodC()
     {

        System.out.println("Class C methodC");
      }
}

class D extends B, C
{
     void methodD()
     {
        System.out.println("Class D methodD");
      }
}

class HybridIneritance
{
      public static void main(String args[])
      {
           D obj1= new D();
           obj1.methodD();
           obj1.methodA();
      }
}
```

Output:

Error!!

Why? Most of the times you will find the following explanation of above error – Multiple inheritance is not allowed in java so class D cannot extend two classes(B and C). **But do you know why it's not allowed?** Let's look at the above code once again, In the above program class B and C both are extending class A and they both have overridden the methodA(), which they can do as they have extended the class A. But since both have different version of methodA(), **compiler is confused** which one to call when there has been a call made to methodA() in child class D (child of both B and C, it's object is allowed to call their methods), this is a ambiguous situation and to avoid it, such kind of scenarios are not allowed in java.

METHOD OVERRIDING **[5 Marks]**

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java**.

In other words, If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

- Method overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

1. method must have same name as in the parent class
2. method must have same parameter as in the parent class.
3. must be IS-A relationship (inheritance).

Advantage of method overriding

The main advantage of method overriding is that the class can give its own specific implementation to a inherited method without even modifying the parent class(base class).

Example program of Method Overriding

```
class Animal
{
	void eat()
	{
		System.out.println("Animal is eating");
	}
}

class Dog extends Animal
{
	void eat()
	{
		System.out.println("Dog is eating");
	}
}
```

```
class Overriding
{
	public static void main(String args[])
	{
		Dog d=new Dog();
		d.eat();
	}
}
```

Output

```
C:\Program Files\Java\jdk1.7.0\bin>javac Overriding.java
C:\Program Files\Java\jdk1.7.0\bin>java Overriding
Dog is eating
```

Real time example of Java Method Overriding

Consider a scenario, Bank is a class that provides functionality to get rate of interest. But, rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7% and 9% rate of interest.

```
class Bank
{
	int getRateOfInterest()
	{
		return 0;
	}
}

class SBI extends Bank
{
	int getRateOfInterest()
	{
		return 8;
	}
}

class ICICI extends Bank
{
	int getRateOfInterest()
	{
		return 7;
	}
}
```

```
class AXIS extends Bank
{
        int getRateOfInterest()
        {
                return 9;
        }
}
class OverrideDemo
{
        public static void main(String args[])
        {
                SBI s=new SBI();
                ICICI i=new
                ICICI(); AXIS
                a=new AXIS();
                System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
                System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
        }       System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
}
```

Output

```
C:\Program Files\Java\jdk1.7.0\bin>javac OverrideDemo.java
C:\Program Files\Java\jdk1.7.0\bin>java OverrideDemo
SBI Rate of Interest: 8
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9
```

METHOD OVERRIDING IN DYNAMIC METHOD DISPATCH **[5 Marks]**

Dynamic method dispatch is a technique which enables us to assign the base class reference to a child class object. As you can see in the below example that the base class reference is assigned to child class object.

Lab Program : 6

Program to demonstrate method overriding and dynamic dispatch.

```
class Base
{
	void display()
	{
		System.out.println("I am in Base Class Display Method");
	}
}

class Child extends Base
{
	void display()
	{
		System.out.println("I am in Child Class Display Method");
	}
}

class DynamicDispatch
{
	public static void main(String args[])
	{
		Base obj=new Child();
		obj.display();
	}
}
```

Output

```
C:\Program Files\Java\jdk1.7.0\bin>javac DynamicDispatch.java
C:\Program Files\Java\jdk1.7.0\bin>java DynamicDispatch
I am in Child Class Display Method
```

Another Example of Dynamic Dispatch

```
class Parent
{
  void display1()
        {
                System.out.println("I am in Parent Class Display Method");
        }
   void display2()
  {
    System.out.println("I am in Parent Class Display2 Method");
  }
}

class Child extends Parent
{
  void display1()
        {
                System.out.println("I am in Child Class Display1 Method");
        }
   void disp()
  {
    System.out.println("I am in Child Class Disp Method");
  }
}
class DynamicDisp
{
  public static void main( String args[])
  {
    Parent obj = new Child(); //Parent class reference to child class object obj.display1();
    obj.display2();
  }
}
```

Output

```
C:\Program Files\Java\jdk1.7.0\bin>javac DynamicDisp.java
C:\Program Files\Java\jdk1.7.0\bin>java DynamicDisp
I am in Child Class Display1 Method
I am in Child Class Display2 Method
```

Note: In dynamic method dispatch the object can call the overriding methods of child class and all the non-overridden methods of base class but it cannot call the methods which are newly declared in the child class. In the above example the object obj was able to call the dispplay1()(overriding method) and display2()(non-overridden method of base class). However if you try to call the disp() method (which has been newly declared in Test class) [obj.disp()] then it would give compilation error with the following message:

```
Exception in thread "main" java.lang.Error: Unresolved compilation
problem: The method disp() is undefined for the type Parent
```

FINAL VARIABLES AND METHODS **[2 or 5 Marks]**

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

All methods and variables can be overridden by default in subclasses. If we wish to prevent the subclasses from overriding the superclass, we declare them as final using the keyword **final**

Ex- final int a=100;
Final void show();

Making a method final ensures that the functionality defined in this method will never be altered in any way.

Final variable **[2 Marks]**
If you make any variable as final, you cannot change the value of final variable(It will be constant).

Example of final variable

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
class Bike
{
 final int speedlimit=90;//final variable void
 run()
 {
        speedlimit=400;
 }
}
class FinalVariable
{
        public static void main(String args[])
        {
                Bike obj=new Bike();
                obj.run();
        }
}
```

Output

```
C:\Program Files\Java\jdk1.7.0\bin>javac FinalVariable.java
FinalVariable.java:6: error: cannot assign a value to final variable speedlimit
      speedlimit=400;
      ^ 1
error
```

Java final method **[2 Marks]**

If you make any method as final, you cannot override it.

```
class Bike
{
    final void run()
            {
                    System.out.println("running");
            }
}

class Honda extends Bike
{
   void run()
           {
                   System.out.println("running safely with 100kmph");
  }
}
class FinalMethod
{
   public static void main(String args[])
           {
     Honda honda= new Honda();
     honda.run();
   }
}
```

Output

```
C:\Program Files\Java\jdk1.7.0_03\bin>javac Finalmethod.java
Finalmethod.java:11: error: run() in Honda cannot override run() in Bike
   void run()
        ^
 overridden method is final 1
error
```

Java final class **[2 Marks]**

If you make any class as final, you cannot extend it.

```
final class Bike
{

}

class Honda extends Bike
{
   void run()
        {
                System.out.println("running safely with 100kmph");
        }
}

class FinalClass
{
   public static void main(String args[])
  {
        Honda h= new Honda();
        h.run();
   }
}
```

Output

```
C:\Program Files\Java\jdk1.7.0_03\bin>javac FinalClass.java
FinalClass.java:6: error: cannot inherit from final Bike
class Honda extends Bike
              ^
1 error
```

Is final method inherited?

Yes, final method is inherited but you cannot override it

Example

```
class Bike
{
      void run()
      {
              System.out.println("Running...");
      }
}
```

```
class Honda extends Bike
{

}

class FinalInheritance
{
  public static void main(String args[])
 {
  Honda h= new Honda()
  h.run();
  }
}
```

Output
Running…

Can we declare a constructor final?

No, because constructor is never inherited.

JAVA GARBAGE COLLECTION **[2 Marks]**

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

- It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
- It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

finalize() method **[2 Marks]**

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

```
1. protected void finalize(){}
```

Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

```
1. public static void gc(){}
```

Note: Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.

ABSTRACT METHODS AND CLASSES **[2 or 5 or 10 Marks]**

By making a method **final** we ensure that the method is not get inherited in subclass. But by making method **abstract** we compulsory override it.

A class that is declared using "abstract" keyword is known as abstract class. It may or may not include abstract methods which means in abstract class you can have concrete methods (methods with body) as well along with abstract methods (without an implementation, without braces, and followed by a semicolon). An abstract class can not be **instantiated** (you are not allowed to create **object** of Abstract class).

Abstract class declaration
Specifying **abstract keyword** before the class during declaration, makes it abstract. Have a look at below code:

```
// Declaration using abstract keyword
abstract class AbstractDemo
{
  // Concrete method: body and braces public
  void myMethod()
 {
    //Statements here
  }

  // Abstract method: without body and braces
  abstract public void anotherMethod();
}
```

Remember two rules:
1) If the class is having few abstract methods and few concrete methods: declare it as abstract class.
2) If the class is having only abstract methods: declare it as interface.

Note : Object creation of abstract class is not allowed Example

of Abstract Method

```
abstract class Demo1
{
  public void disp1()
  {
   System.out.println("Concrete method of abstract class");
  }
  abstract void disp2();
}
```

```
class Demo2 extends Demo1
{
  /* I have given the body to abstract method of Demo1 class It is
  must if you don't declare abstract method of super class
  compiler would throw an error*/

  void disp2()
  {
    System.out.println("I'm overriding abstract method");
  }
}

class AbstractMethod
{
  public static void main(String args[])
  {
    Demo2 obj = new Demo2();
    obj.disp2();
  }
}
```

Output

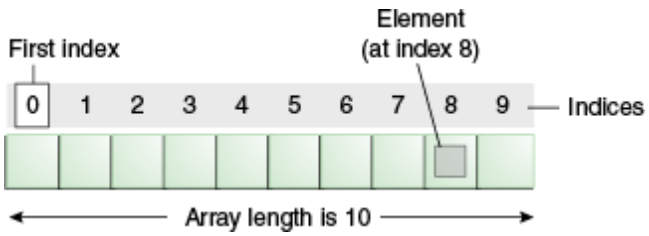
```
C:\Program Files\Java\jdk1.7.0\bin>javac AbstractMethod.java
C:\Program Files\Java\jdk1.7.0\bin>java AbstractMethod
I'm overriding abstract method
```

ARRAYS **[2 or 5 or 10 Marks]**

Array is a collection of similar type of elements that have contiguous memory location.

Java array is an object the contains elements of similar data type. It is a data structure where we store similar elements. We can store only fixed set of elements in a java array.

Array in java is index based, first element of the array is stored at 0 index.



Advantage of Java Array

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data easily.
- **Random access:** We can get any data located at any index position.

Disadvantage of Java Array

- **Size Limit:** We can store only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in java.

Types of Array in java

There are two types of array.

- One Dimensional Array
- Two Dimensional Array
- Multidimensional Array

One Dimensional Array **[10 Marks]**

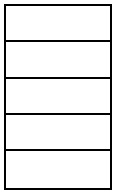
A list of items can be given one variable name using only one subscript and such a variable is called One Dimensional Array.

Syntax :

Datatype ArrayName=new Datatype[size];

Example : int Array=new int[5];

The computer reserves five storage locations as shown below:



Creating an Array

Creation of array involves three steps :
1.Declaring the array.
2.Creating memory location.
3.Putting values into the memory location.

1. Declaration of Array

Array in java may be declared in three ways

1. dataType[] arr; (or)
2. dataType []arr; (or)
3. dataType arr[];

Example :

1. int [] number;
2. int []marks;
3. int students[];

Creation of Arrays

After declaring an array, we need to create it in the memory. Java allows us to create array using **new operator.**

1 dataType[size] array=new dataType;
2 dataType [size]array=new dataType;
3 dataType array[size]=new dataType;

Ex :

```
1   int[5] Array=new int;
2   int [5]Array=new int;
3   int Array[5]=new int;
```

Initialization of Arrays

The final step is to put values into the array created. This process is known as initialization.

Example :

```
Array[0]=10;
Array[1]=20;
Array[2]=30;
Array[3]=40;
Array[4]=50;
```

Declaration, Instantiation and Initialization of Java Array

We can declare, instantiate and initialize the java array together by:

1. int a[]={33,3,4,5}; //declaration, instantiation and initialization

Example of One Dimensional Array

```
class OneDimentional
{
   public static void main(String args[])
  {
        int a[]=new int[5];//declaration and instantiation
          a[0]=10;//initialization
          a[1]=20;
          a[2]=30;
          a[3]=40;
          a[4]=50;

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

```
C:\Program Files\Java\jdk1.7.0\bin>javac OneDimentional.java
C:\Program Files\Java\jdk1.7.0\bin>java OneDimentional
10
20
30
40
50
```

Program for Sorting a List of Numbers

```
import java.util.Scanner;
class SortingNumber
{
	public static void main(String args[])
	{
		int A[]=new int[50];
		Scanner scan=new Scanner(System.in);
		System.out.println("Enter  Array  Size");
		int n=scan.nextInt();
		System.out.println("Enter Array Elements");
		for(int i=0;i<n;i++)
		{
			A[i]=scan.nextInt();
		}
		for(int i=0;i<n;i++)
		{
			for(int j=i+1;j<n;j++)
			{
				if(A[i]>A[j])
				{
					int temp=A[i];
					A[i]=A[j];
					A[j]=temp;
				}
			}
		}
		System.out.println("The sorted array is ");
		for(int i=0;i<n;i++)
		{
			System.out.println(A[i]);
		}
		System.out.println();
	}
}
```

Output

```
C:\Program Files\Java\jdk1.7.0\bin>javac SortingNumber.java
C:\Program Files\Java\jdk1.7.0\bin>java SortingNumber Enter
Array Size
5
```

Enter Array Elements
13 24 11 65 18
The sorted array is 11
13
18
24
65

Two Dimensional Arrays **[10 Marks]**

```
import java.util.Scanner;
class TwoDArray
{
	public static void main(String args[])
	{
		int A[][]=new int[10][20];
		Scanner scan=new Scanner(System.in);
		System.out.println("Enter row and column");
		int row=scan.nextInt();
		int column=scan.nextInt();

		System.out.println("Enter Array Elements"); for(int
		i=0;i<row;i++)
		{
			for(int j=0;j<column;j++)
			{
				A[i][j]=scan.nextInt();
			}
		}

		System.out.println("The Array Elements are ");
		for(int i=0;i<row;i++)
		{
			for(int j=0;j<column;j++)
			{
				System.out.print(" "+A[i][j]);
			}
			System.out.println(" ");
		}
	}
}
```

Output

```
C:\Program Files\Java\jdk1.7.0\bin>javac TwoDArray.java
C:\Program Files\Java\jdk1.7.0\bin>java TwoDArray Enter
row and column
2
2
Enter Array Elements
11
23
14
21
The Array Elements are
 11 23
 14 21
```

STRING **[2 Marks]**

What is String in java

String is a sequence of characters. But in java, string is an object that represents a sequence of characters. String class is used to create string object.

How to create String object? **[2 Marks]**

There are two ways to create String object:

1. By string literal
2. By new keyword

---

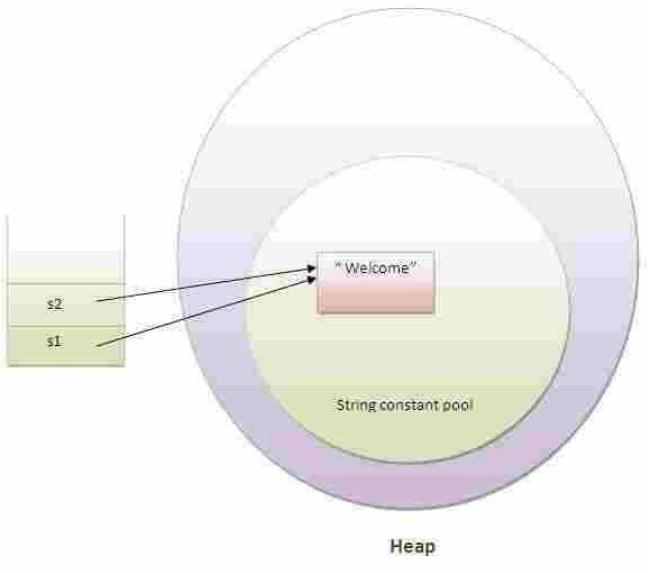
1) String Literal

Java String literal is created by using double quotes. For Example:

```
1. String s="welcome";
```

Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance is returned. If string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

```
1. String s1="Welcome";
2. String s2="Welcome";//will not create new instance
```



In the above example only one object will be created. Firstly JVM will not find any string object with the value "Welcome" in string constant pool, so it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create new object but will return the reference to the same instance.

Note: String objects are stored in a special memory area known as string constant pool. Why

java uses concept of string literal?

To make Java more memory efficient (because no new objects are created if it exists already in string constant pool).

2) By new keyword

1. String s=new String("Welcome");//creates two objects and one reference variable

In such case, JVM will create a new string object in normal(non pool) heap memory and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in heap(non pool).

Example

```
public class StringExample
{
    public static void main(String args[])
    {
            String s1="java";//creating string by java string literal
            char ch[]={'s','t','r','i','n','g','s'};
            String s2=new String(ch);//converting char array to string

            String s3=new String("example");//creating java string by new keyword

            System.out.println(s1);
            System.out.println(s2);
            System.out.println(s3);
    }
}
```

Output

java
strings
example

Java String class methods **[5 Marks]**

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

Method Call	*Taskperformed*
`s2 = s1.toLowerCase;`	Converts the string s1 to all lowercase
`s2 = s1.toUppercCase;`	Converts the string s1 to all Uppercase
`s2 = s1.replace('x', 'y');`	Replace all appearances of x with y
`s2 = s1.trim();`	Remove white spaces at the beginning and end of the string s1
`s1.equals(s2)`	Returns 'true' if s1 is equal to s2
`s1.equalsIgnoreCase(s2)`	Returns 'true' if s1= s2, ignoring the case of characters
`s1.length()`	Gives the length of s1
`s1.ChartAt(n)`	Gives nth character of s1
`s1.compareTo(s2)`	Returns negative if s1 < s2, positive if s1 > s2, and zero if s1 is equal s2
`s1.concat(s2)`	Concatenates s1 and s2
`s1.substring(n)`	Gives substring starting from n^{th} character
`s1.substring(n, m)`	Gives substring starting from n^{th} character up to m^{th} (not including m^{th})
`String.Valueof(p)`	Creates a string object of the parameter p (simple type or object)
`p.toString()`	Creates a string representation of the object p
`s1.indexof('x')`	Gives the position of the first occurrence of 'x' in the string s1
`s1.indexof('x', n)`	Gives the position of 'x' that occurs after nth position in the string s1
`String.Valueof (Variable)`	Converts the parameter value to string representation

Lab 4 . Program to implement at least 10 string operations on Strings [10 Marks]

```
class StringDemo
{
	public static void main(String args[])
	{
		String s1="Bachelor";
		String s2="Of";
		String s3="Computer";
		String s4="Application";

		System.out.println(s1.toUpperCase());
		System.out.println(s1.toLowerCase());
		System.out.println(s3.concat(s4));
		System.out.println(s4.length());
		System.out.println(s1.charAt(2));
		System.out.println(s1.indexOf('c'));
		System.out.println(s2.replace('f','o'));
		System.out.println(s1.substring(5));
		System.out.println(s4.substring(0,3));
		System.out.println(s1.compareTo(s4));
		System.out.println(s1.equals(s2));
		System.out.println(s4.startsWith("App"));
		System.out.println(s1.endsWith("or"));
	}
}
```

Output

```
C:\Program Files\Java\jdk1.7.0\bin>javac StringDemo.java
C:\Program Files\Java\jdk1.7.0\bin>java StringDemo
BACHELOR
bachelor
ComputerApplication
11
c
2
Oo
lor
App
1
false
true
true
```

DIFFERENCE BETWEEN STRING AND STRINGBUFFER [2 or 5 Marks]

There are many differences between String and StringBuffer. A list of differences between String and StringBuffer are given below:

No.	String	StringBuffer
1)	String class is immutable.	StringBuffer class is mutable.
2)	String is slow and consumes more memory when you concat too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when you cancat strings.
3)	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.

VECTORS IN JAVA **[2 Marks]**

Vector implements a dynamic array. It is similar to ArrayList, but with two differences −

- Vector is synchronized.
- Vector contains many legacy methods that are not part of the collections framework.

Vector proves to be very useful if you don't know the size of the array in advance or you just need one that can change sizes over the lifetime of a program.

Wrapper class in Java **[2 Marks]**

Wrapper class in java provides the mechanism to convert primitive data types into object and object into primitive data types.

Since J2SE 5.0, autoboxing and unboxing feature converts primitive into object and object into primitive automatically. The automatic conversion of primitive into object is known as autoboxing and vice-versa unboxing.

The eight classes of java.lang package are known as wrapper classes in java. The list of eight wrapper classes are given below:

Primitive Type	Wrapper class
boolean	Boolean
Char	Character
Byte	Byte
Short	Short
Int	Integer
Long	Long
Float	Float
Double	Double

Wrapper class Example: Primitive to Wrapper

```
public class WrapperExample1
{
        public static void main(String args[])
        {
                //Converting int into Integer
                int a=20;
                Integer i=Integer.valueOf(a);//converting int into Integer
                Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally
                System.out.println(a+" "+i+" "+j);
         }
}
```

Output:

20 20 20

Wrapper class Example: Wrapper to Primitive

Wrapper class Example: Wrapper to Primitive

```
public class WrapperExample2
{
	public static void main(String args[])
	{
		//Converting Integer to int
		Integer a=new Integer(3);
		int i=a.intValue();//converting Integer to int
		int j=a;//unboxing, now compiler will write a.intValue() internally
		System.out.println(a+" "+i+" "+j);
	}
}
```

Output:

```
3 3 3
```