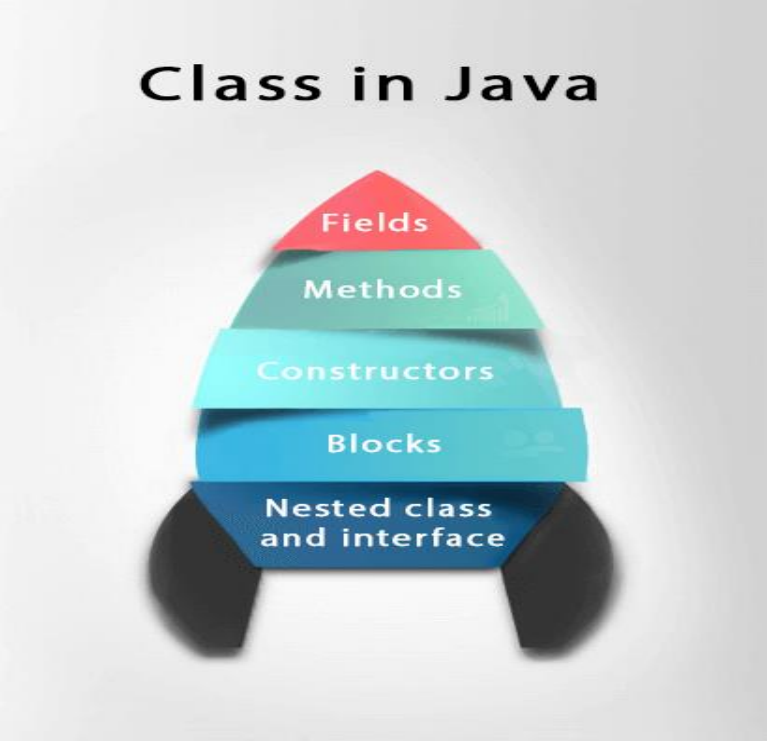
Unit-2

Classes, Arrays, Strings and Vectors

Class

- A class is a group of objects which have common properties.
- It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.



Syntax:

```
Class  <class_name>
{
   field;
   method;
}
```

```
Class Student
{
int name="raj";
}
```

Methods Declaration

The general form of a method declaration is

```
type method name (parameter-list)
{
    method-body;
}
```

Method declarations have four basic parts:

1. The name of the method (method name)
2. The type of the value the method returns (type)
3. A list of parameters (parameter-list)
4. The body of the method

```
Class Rectangle
{
   int length;
   int width;
       Void getData(int x, int y)   // Method declaration
       {
        length=x;
       Width=y;
       }
       int rectArea( ) // Declaration of another method
       {
          int area=length*width;
          return(area);
       }
}
```

Object

- **An object is an instance of a class. Using class object is created.**
- Let us now look deep into what are objects. If we consider the real-world, we can find many objects around us, cars, dogs, humans, etc. All these objects have a state and a behavior.
  - **State:** represents the data (value) of an object.
  - **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- If we consider a dog, then its state is - name, breed, color, and the behavior is - barking, wagging the tail, running.
- If you compare the software object with a real-world object, they have very similar characteristics.
- Software objects also have a state and a behavior. A software object's state is stored in fields and behavior is shown via methods.

Creating objects

- Objects in java are created using the new operator.
- The new operator creates an object of the specified class and returns a reference to that object

Syntax:

Class_name obj=new Class_Name();

Student s=new Student();

We can create any number of objects of Rectangle.

Rectangle rect1= new Rectangle();

Rectangle rect2=new Rectangle( );

Accessing Class Members

Instance variables ad methods cannot be accessed directly. To access we must use object and the dot operator

Objectname.variablename=value;

Objectname.methodname(parameter-list);

- Example:

  rect1.length=15;

  rect1.width=10;

- Example:

  Rectangle rect1=new Rectangle( );

  rect1.getData(15,10);

```
class Student
{
        int id;
        String name;
       public void display()
       {
              System.out.println(s1.id+" "+s1.name);

       }

        public static void main(String args[])
       {
               Student s1=new Student();
               s1.id=101;
               s1.name=“Priya";
               s1.diaplay();
              }
}
```

Constructors

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

Rules for creating Java constructor

There are basically three rules defined for the constructor.

- Constructor name must be same as its class name
- Constructor must have no explicit return type. i.e Constructors do not have a return type — not even void
- Constructors are invoked using the new operator when an object is created. Constructors play the role of initializing objects.

Types of Java constructors

There are two types of constructors:

- Default constructor (no-arg constructor)
- Parameterized constructor

Default constructor

```
class Bike1
{
    Bike1()
    {
    System.out.println("Bike is created");
    }
    public static void main(String args[])
    {
    Bike1 b=new Bike1();
    }
}
```

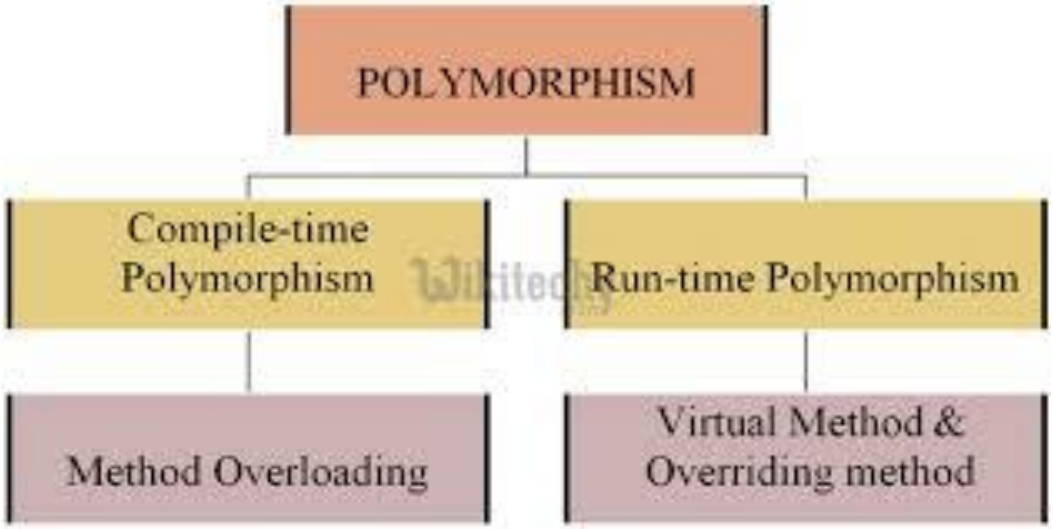
O/P:Bike is created

Parameterized constructor

```
class Student
{
   int id;
   String name;
            Student(int i,String n)
           {
            id = i;
            name = n;
            }
      void display()
        {
        System.out.println(id+" "+name);
        }
   public static void main(String args[])
        {
            Student s1 = new Student(111,"Karan");
           Student s2 = new Student(222,"Aryan");
            s1.display();
           s2.display();
          }
}
```

Polymorphism

- Polymorphism in Java is a concept by which we can perform a single action in different ways.
- Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.



Method Overloading(Compile Time Polymorphism)

- If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.

Advantage of method overloading

- Method overloading increases the readability of the program.

Different ways to overload the method

There are two ways to overload the method in java

- By changing number of arguments
- By changing the data type

Method Overloading: changing no. of arguments

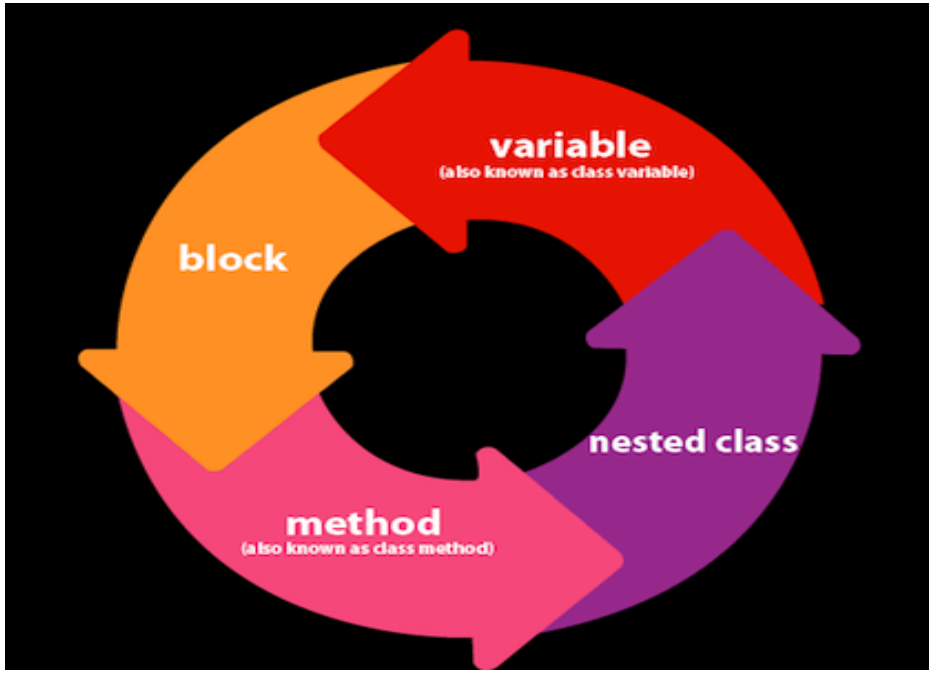
```
class TestOverloading
{
        void add(int a,int b)
       {
       int c=a+b;
       System.out.println("result with 2 parameter"+c);
       }
        void add(int a,int b,int c)
       {
       int d=a+b+c;
       System.out.println("result with 3 parameter"+c);
       }
       public static void main(String[] args)
       {
       TestOverloading t=new TestOverloading();
       t.add(11,11);
       t.add(11,11,11);
       }
}
```

Method Overloading: changing data type of arguments

```
class TestOverloading1
{
        int add(int a,int b)
       {
       int c=a+b;
       return c;
       }
        double add(int a,double b)
       {
       int d=a+b+c;
          return d;
       }
       public static void main(String[] args)
       {
       TestOverloading1 t=new TestOverloading1();
       t.add(11,11);
       t.add(11,11.5);
       }
}
```

Static Keyword

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



Static Variables

- Static variables are also known as Class variables.
- These variables are declared similarly as instance variables, the difference is that static variables are declared using the static keyword within a class outside any method constructor or block.
- To access static variables, we need not create an object of that class, we can simply access the variable as
- class_name.variable_name;

- static String name=”raj”;

Static Method

It is a method which belongs to the class and no need to create the object(instance)

A static method can access only static data. It can not access non-static data (instance variables)

A static method can call only other static methods and can not call a non-static method from it.

A static method can be accessed directly by the class name and doesn't need any object

Ex: static int max(int x,int y);

```
class Test
{
   // static method
   static void m1()
   {
      System.out.println("from m1");
   }

   public static void main(String[] args)
   {
        // calling m1 without creating
        // any object of class Test
         m1();
   }
}
```

```
// Java program to demonstrate use of static blocks
class Test
{
    static int a = 10;
    static int b;
    static {
            System.out.println("Static block initialized.");
            System.out.println(a+””+b);
    }
    public static void main(String[] args)
    {
    System.out.println("from main");

    }
}
```

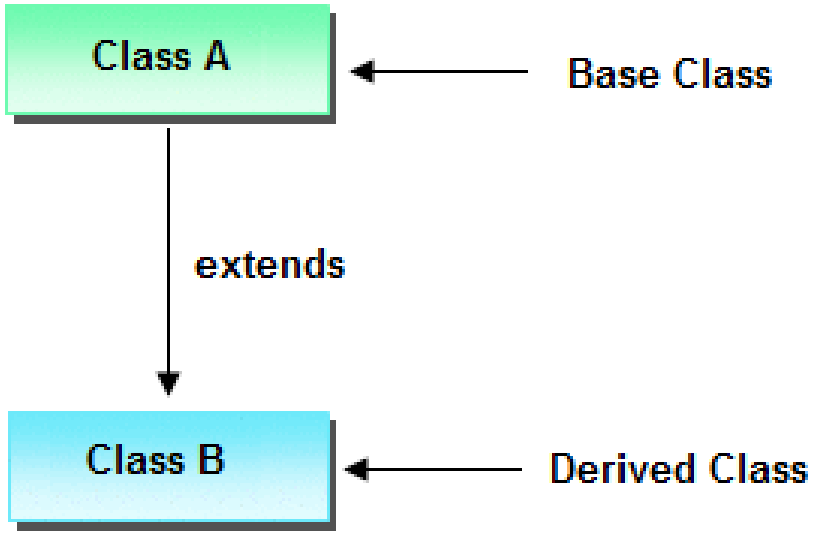
Inheritance

- Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.
- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class.

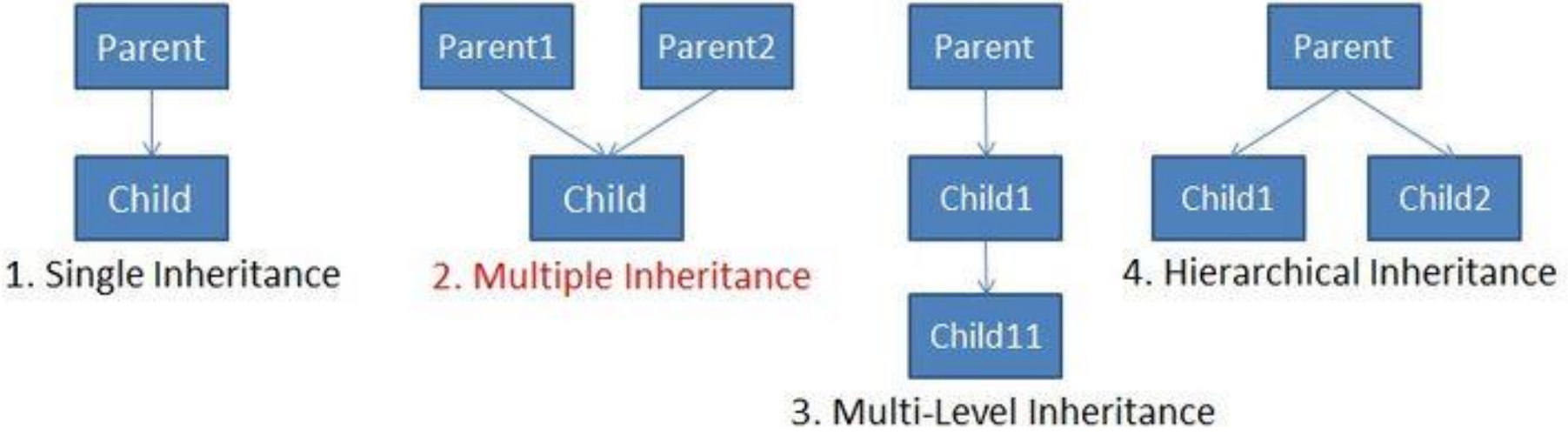
  - **Sub Class/Child Class**: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
  - **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

The **extends** keyword indicates that you are making a new class that derives from an existing class.

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

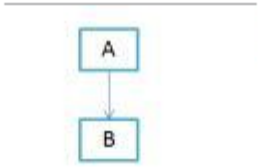


Types of Inheritance



Single Inheritance

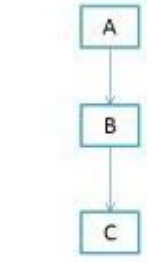
```
class Animal
{
        void eat()
        {
        System.out.println("eating...");
        }
}
class Dog extends Animal
{
        void bark()
        {
        System.out.println("barking...");
        }
        public static void main(String args[])
        {
        Dog d=new Dog();
        d.bark();
        d.eat();
        }
}
```



(a) Single Inheritance

Multilevel Inheritance

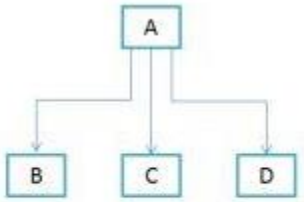
```
class Animal
{
        void eat()
        {
        System.out.println("eating...");
        }
}
class Dog extends Animal
{
        void bark()
        {
        System.out.println("barking...");
        }
}
class BabyDog extends Dog
{
        void weep()
        {
        System.out.println("weeping...");
        }
        public static void main(String
            args[])
        {
        BabyDog d=new BabyDog();
        d.weep();
        d.bark();
        d.eat();
        }
}
```



(d) Multilevel Inheritance

Hierarchical Inheritance

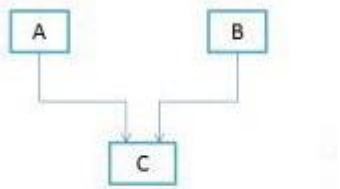
```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();

}}
```



(c) Hierarchical Inheritance

Multiple Inheritance

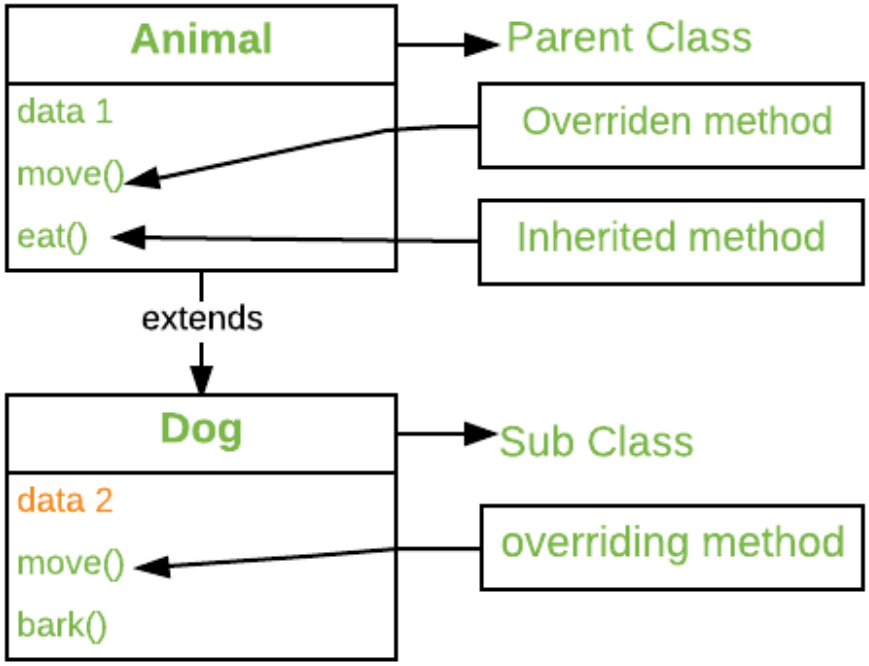
```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{

 public static void main(String args[]){
   C obj=new C();
   obj.msg();//Now which msg() method would be invoked?
}
}
```



(b) Multiple Inheritance

Overriding

- If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.
- When a method in a subclass has the same name, same parameters or signature, and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.



```
class Human{
  //Overridden method
  public void eat()
  {
    System.out.println("Human is eating");
  }
}
class Main extends Human{
  //Overriding method
  public void eat(){
    System.out.println("Boy is eating");
  }
  public static void main( String args[]) {
    Main obj = new Main();
    //This will call the child class version of eat()
    obj.eat();
  }
}
```

```
// Driver class
class Main {
    public static void main(String[] args)
    {
        // If a Parent type reference refers
        // to a Parent object, then Parent's
        // show is called
        Parent obj1 = new Parent();
        obj1.show();

        // If a Parent type reference refers
        // to a Child object Child's show()
        // is called. This is called RUN TIME
        // POLYMORPHISM.
        Parent obj2 = new Child();
        obj2.show();
    }
}
```

Final Keyword

The final keyword in java is used to restrict the user. It is used to make a variable as a constant, Restrict method overriding, Restrict inheritance.

Final variables

- A variable declared with the final keyword is known as a final variable.
- It may be member variable or local variable.
- If you make any variable as final, you cannot change the value of a final variable(It will be constant).
- A variable declared with the final keyword cannot be modified by the program after initialization.

- **Example:** public final double PI=3.142;

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

Output:Compile Time Error

final method

- When a method is declared with final keyword, it is called a final method.
- A final method cannot be overridden. Which means even though a sub class can call the final method of parent class without any issues but it cannot override it.
- The compiler checks and gives an error if you try to override the method.
- When we want to restrict overriding, then make a method as a final.
- Ex:

```
final void demo()
{
…. .
}
```

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

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

Output:Compile Time Error

final class

- When a class is declared with the final keyword, it is called a final class.
- It makes a class final, meaning that the class can not be inherited by other classes.
- We cannot extend a final class.
- When we want to restrict inheritance then make the lass as a final.

```
final class Bike
{
}

class Honda1 extends Bike
{
        void run()
       {
       System.out.println("running safely with 100kmph");
       }
            public static void main(String args[])
       {
        Honda1 honda= new Honda1();
        honda.run();
        }
}
```

Finalizer Methods

- Java supports a concept called finalization which is just opposite to initialization.
- Java run-time is automatic garbage collecting system. It automatically frees up the memory resources used by the objects.
- java.lang.Object.finalize()
- Sometimes an object will need to perform some action when it is destroyed. For example closing an open connection or releasing any resources held. To handle such situations, Java provides a mechanism called finalizer. Method is simply **finalize()**
- It is similar to destructor in C++

```
public class JavafinalizeExample1
{
    public static void main(String[] args)
   {
       JavafinalizeExample1 obj = new JavafinalizeExample1();
       obj = null;
       System.gc();
       System.out.println("end of garbage collection");

   }
      protected void finalize()
   {
       System.out.println("finalize method called");
   }
}
```

Java Abstract Classes and Methods

- Data abstraction is the process of hiding certain details and showing only essential information to the user.
- Abstraction can be achieved with either abstract classes or interfaces.

Abstract class in Java

- A class that is declared as abstract is known as abstract class.
- Abstract classes may or may not contain *abstract methods*, i.e., methods without body (public void get();)
- But, if a class contain at least one abstract method, then the class must be declared abstract.
- If a class is declared abstract, it cannot be instantiated.
- To utilize an abstract class, you have to inherit it from another class, provide implementations for the abstract methods in it.

  Ex: abstract class class_name { }

Abstract method

- Method that are declared without any body within an abstract class are called abstract method.
- The method body will be defined by its subclass.
- Abstract method can never be final and static.
- The abstract keyword is used to declare the method as abstract.
- You have to mention the abstract keyword before the method name in the method declaration.
- An abstract method has a method signature, but no method body.
- Instead of curly braces, an abstract method will have a semicolon (;) at the end.
- Syntax: abstract return_type function_name (); // No definition

```
abstract class Animal
{
        public abstract void animalSound();
        public void sleep()
       {
          System.out.println("Zzz");
        }
}

Animal myObj = new Animal(); // will generate an error
```

```
abstract class ClassA
 {
       abstract void callme();
 }
 class ClassB extends ClassA
 {
          void callme()
          {
          System.out.println("this is call me inside child.");
         }
}
class abs
{
          public static void main(String[] args)
          {
          ClassB b = new ClassB();
         b.callme();
          }
 }
```

Output:

this is call me inside child.

Visibility Control

If we don't want the objects of class directly alter the value of a variable or access method. We can achieve this in Java by applying visibility modifiers to instance variable and methods. The visibility modifiers are also known as access modifiers.

Java has four access modifiers:

1. public
2. default (Friendly)
3. private
4. protected

Private

The private access modifier is accessible only within the class.

```
class A
{
        private int data=40;
        private void msg()
        {
        System.out.println("Hello java");
        }
}

public class Simple
{
         public static void main(String args[])
        {
          A obj=new A();
          System.out.println(obj.data);//Compile Time Error
          obj.msg();//Compile Time Error
          }
}
```

Default

If you don't use any modifier, it is treated as default by default. The default modifier is accessible only within package. It cannot be accessed from outside the package.

```
package pack;
class A
{
  void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.*;
class B
{
          public static void main(String args[])
        {
          A obj = new A();//Compile Time Error
          obj.msg();//Compile Time Error
          }
}
```

Protected

- The protected access modifier is accessible within package and outside the package but through inheritance only.
- The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

```
package pack;
public class A
{
        protected void msg()
        {
        System.out.println("Hello");
        }
}

package mypack;
import pack.*;

class B extends A
{
          public static void main(String args[])
        {
           B obj = new B();
           obj.msg();
          }
}
```

Public

The public access modifier is accessible everywhere. It has the widest scope among all other modifiers.

```
package pack;
public class A
{
        public void msg()
        {
        System.out.println("Hello");
        }
}
```

```
 package mypack;
import pack.*;
  class B
{
          public static void main(String args[])
        {
          A obj = new A();
          obj.msg();
          }
}
```

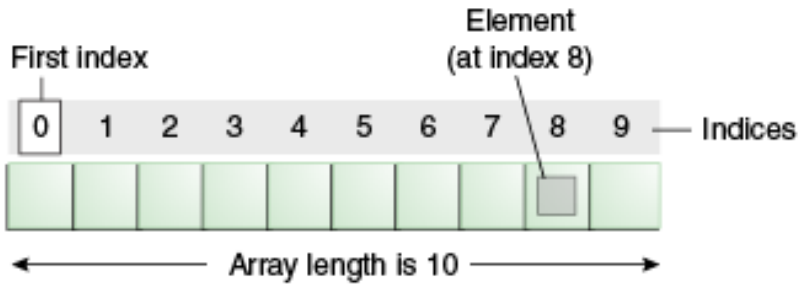
Array

- Array is a group of continuous or related data items that share a common name.
- Syntax:

```
dataType[] arrayRefVar;
        or
dataType arrayRefVar[];
```

- Example:

```
int salary[10];
```



Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Single Dimensional Array

- one dimension means it has only one value per location or index.
- One-dimensional array in Java programming is an array with a bunch of values having been declared with a single index.

- As you can see in the example given above, firstly, you need to declare the elements that you want to be in the specified array.
- Secondly, the location of each element needs to particularized as well, since that is where the elements will be stored respectively.
- Thirdly, you need to declare the array accordingly.

```
One dimensional array elements are
10
20
30
```

```
class OnedimensionalStandard
{
    public static void main(String args[])
    {
    int[] a=new int[3];//declaration
    a[0]=10;//initialization
    a[1]=20;
    a[2]=30;
    //printing array
    System.out.println("One dimensional array elements are");
    System.out.println(a[0]);
    System.out.println(a[1]);
    System.out.println(a[2]);
    }
}
```

```
class Testarray
{
    public static void main(String args[])
    {
    int a[]={33,3,4,5};//declaration, instantiation and initialization
    //printing array
        for(int i=0;i<a.length;i++)//length is the property of array
        {
        System.out.println(a[i]);
        }
    }
}
```

ArrayIndexOutOfBoundsException

```
public class TestArrayException
{
    public static void main(String args[])
    {
    int arr[]={50,60,70,80};
        for(int i=0;i<=arr.length;i++)
        {
        System.out.println(arr[i]);
        }
    }
}
```

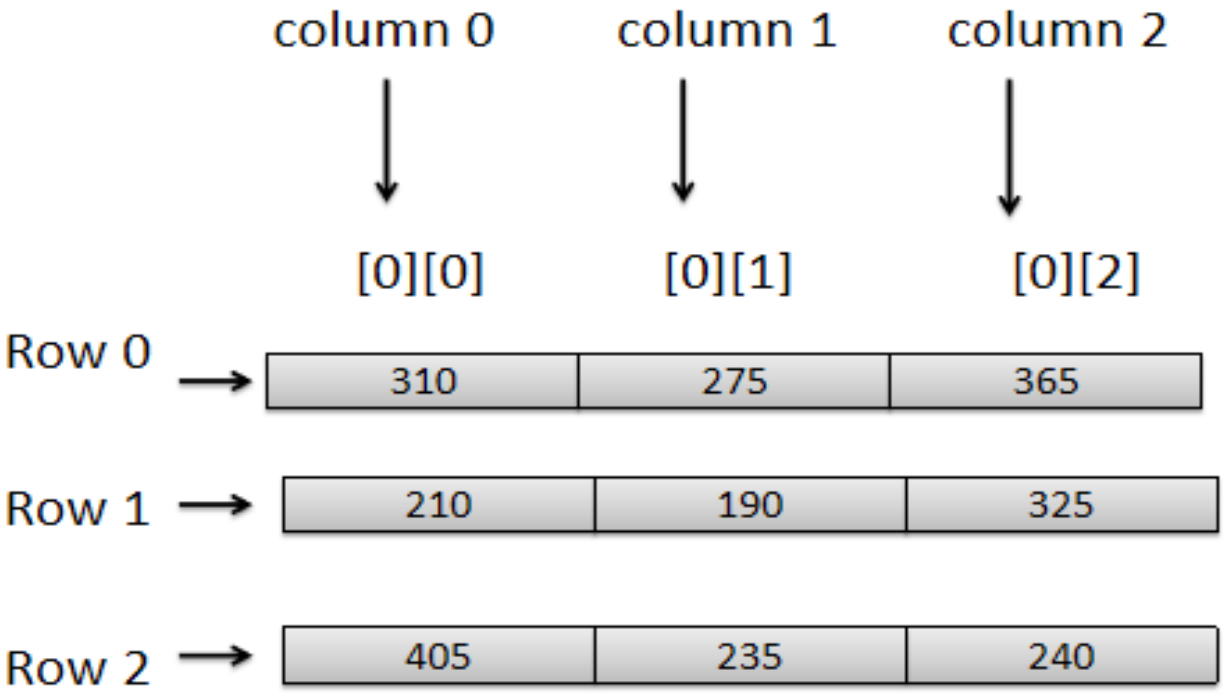
Multidimensional Array

In such case, data is stored in row and column based index (also known as matrix form).

Syntax:

```
dataType [][]arrayRefVar;
(or)
dataType arrayRefVar[][];
```

REPRESENTATION OF TWO-DIMENSIONAL ARRAY IN MEMORY



Example to instantiate Multidimensional Array in Java

```
int[][] arr=new int[3][3];//3 row and 3 column
```

Example to initialize Multidimensional Array in Java

```
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
arr[1][1]=5;
arr[1][2]=6;
arr[2][0]=7;
arr[2][1]=8;
arr[2][2]=9;
```

```
//Java Program to illustrate the use of multidimensional array
class Testarray
{
    public static void main(String args[])
    {
     int arr[][]={{1,2,3},{2,4,5},{4,4,5}};

        for(int i=0;i<3;i++)
        {
            for(int j=0;j<3;j++)
            {
              System.out.print(arr[i][j]+" ");
            }
         System.out.println();
        }
    }
}
```

String

- String are a sequence of characters. In Java programming language, strings are treated as objects.
- The Java platform provides the String class to create and manipulate strings.

Creating Strings

The most direct way to create a string is to write −

```
String greeting = "Hello world!";
```

```
public class StringDemo
{
        public static void main(String args[])
       {
          char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.' };
          String helloString = new String(helloArray);
          System.out.println( helloString );
        }
}
```

Strings is the length() method, which returns the number of characters contained in the string object.

```
public class StringDemo
{
  public static void main(String args[])
    {
      String palindrome = "Dot saw I was Tod";
      int len = palindrome.length();
      System.out.println( "String Length is : " + len );
    }
}
```

String Method

Method Call	Task performed
S2=s1.toLowerCase();	Converts the string s1 to all lowercase
S2=s1.toUppercase();	Converts the string s1 to all Uppercase
S2=s1.replace('x', 'y');	Replace all appearance 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 length of s1
S1.charAt(n);	Gives nth character of s1
S1.compareTo(s2)	Returns negative if s1<s2, positive if s1>s2 and
S1.concat(s2)	Concatenates s1 and s2

Vector

- Vector is class in java which is present in java.util pachage.
- Vector is like the dynamic array which can grow or shrink its size. Unlike array, we can store n-number of elements in it as there is no size limit.

Vector Creation

```
Vector intVect = new Vector( ); // declaring without size
Vector list = new Vector(3); // declaring with size
```

Vector Method

Method Call	Task performed
list.addElement(item)	Adds the item specified to the list at the end
list.elementAt(10)	Gives the name of the 10th object
list.size()	Gives the number of objects present
list.removeElement(item)	Removes the specified item from the list
List.removeElementAt(n)	Removes the item stored in the nth position of the list
List.removeAllElements()	Removes all the elements in the list
List.copyInto(array)	Copies all items from list to array
List.insertElementAt(item, n)	Inserts the item at nth position

```
import java.util.*;
public class VectorExample {
      public static void main(String args[]) {
        //Create a vector
        Vector<String> vec = new Vector<String>();
        //Adding elements using add() method of List
        vec.add("Tiger");
        vec.add("Lion");
        vec.add("Dog");
        vec.add("Elephant");
        //Adding elements using addElement() method of Vector
        vec.addElement("Rat");
        vec.addElement("Cat");
        vec.addElement("Deer");

        System.out.println("Elements are: "+vec);
      }
}
```

```
import java.util.*;
public class VectorExample {
      public static void main(String args[]) {

        Vector<String> vec = new Vector<String>();
         vec.add("Tiger");
        vec.add("Lion");
        vec.add("Dog");
        vec.add("Elephant");
        vec.addElement("Rat");
        vec.addElement("Cat");
        vec.addElement("Deer");
         Enumeration enu = vec.elements();
            System.out.println("The enumeration of values are:");
             while (enu.hasMoreElements()) {
            System.out.println(enu.nextElement());
      }
}
}
```

WRAPPER CLASSES

Since, vectors cannot handle primitive data types like **int**, **float**, **long** ,**char**, and **double**.

Primitive data types may be converted into object types by using the wrapper classes contained in the **java.lang** packages.

The wrapper classes have a number of unique methods for handling primitive data types and objects.

WRAPPER CLASSES FOR CONVERTING SIMPLE TYPES

Wrapper classes for converting simple types

Simple Type	Wrapper Class
boolean	Boolean
char	Char
double	Double
float	Float
Int	int
long	long

Converting numeric strings to primitive numbers using parsing methods

Method Calling	Conversion Action
Int i=Integer.parseInt(str);	Converts string to primitive integer
Long l= Long.parseLong(str)	Converts string to primitive long

- Converting numbers to string using toString() method

Method calling	Conversion Action
str= Integer.toString(i);	Primitive integer to string
Str = Float.toString(f);	Primitive float to string
str= Double.toString(d);	Primitive double to string
Str= Long.toString(l);	Primitive long to string

- Converting string object to numeric objects using the static method ValueOf()

Method calling	Conversion Action
DoubleVal=Double.Valueof(str)	Converts string to Double object
FloatVal=Float.Valueof(str)	Converts string to float object
IntVal=Integer.Valueof(str)	Converts string to Integer object
LongVal=Long.Valueof(str)	Converts string to Long object

Autoboxing

The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing, for example, byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short.

```
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 explicitly
    Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally
     System.out.println(a+" "+i+" "+j);
    }
}
```

Unboxing

The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing. It is the reverse process of autoboxing.

```
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 explicitly
    int j=a;//unboxing, now compiler will write a.intValue() internally

    System.out.println(a+" "+i+" "+j);
    }
}
```