A decorative vertical bar on the left side of the slide. It consists of a thick dark teal bar with two thin white vertical lines running through it. To the right of the teal bar, there are several orange circles of varying sizes, some overlapping the teal bar and others floating to the right. The entire slide is framed by thin orange vertical lines on the far left and far right.

# *OBJECT ORIENTED PROGRAMMING CONCEPTS (JAVA) (UNIT-II)*

*C is **procedural** language.  
whereas  
Java is **object-oriented**  
language.*



## WHAT IS JAVA ???

- *Java is an object oriented programming language developed by Sun Micro Systems. There are different types of programming but Java is unique among them. With Java a developer can write an application and run that on different operating systems including windows, MAC OS, UNIX and main frames without any modification in the code.*
- *When developing an application in any programming language the source code is passed through a compiler that transforms the code into a set of native instructions.*



- *The compiler converts the source code into instructions which the processor can understand but when writing Java application the developer does not need to directly call windows or other operating system library function.*



## HOW JAVA DIFFERS FROM C ???

- Java does not include the C unique statement keywords **sizeof** and **typedef**.
- Java does not contain the data type **struct** and **union**.
- Java does not define the type modifiers keywords **auto**, **extern**, **register**, **signed** and **unsigned**.
- Java does not support any explicit pointer type.
- Java does not have a preprocessor and therefore we cannot use **#define**, **#include** statements.
- Java declares function with no arguments with empty parenthesis and not with **void** keywords as done in C.
- Java adds new operators such a **instanceof**.



# ***JAVA ENVIRONMENT***



You can set environment for Java programming language, Following are the steps to set up the environment:

- Java SE is freely available from the link [Download Java](#). You can download a version based on your operating system.
- Follow the instructions to download Java and run the **.exe** to install Java on your machine. Once you installed Java on your machine, you will need to set environment variables to point to correct installation directories –

### *Setting Up the Path for Windows :*

- Assuming you have installed Java in *c:\Program Files\java\jdk* directory –
- Right-click on 'My Computer' and select 'Properties'.
- Click the 'Environment variables' button under the 'Advanced' tab.
- Now, alter the 'Path' variable so that it also contains the path to the Java executable. Example, if the path is currently set to 'C:\WINDOWS\SYSTEM32', then change your path to read 'C:\WINDOWS\SYSTEM32;c:\Program Files\java\jdk\bin'.

# ***JAVA PROGRAM DEVELOPMENT***





- We can write a simple hello java program easily after installing the JDK.
- To create a simple java program, you need to create a class that contains main method. Write down following code in notepad:

- For Example :

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



- Save this file as simple.java
- Compile this program on command prompt.
- To Compile : javac simple.java  
To Execute : java simple.java



## ***JAVA CLASS LIBRARY :-***

These packages consist of a large number of classes which are a part of Java **API**. Some of the commonly used built-in packages are:

- 1) **java.lang**: Contains language support classes(e.g classed which defines primitive data types, math operations). This package is automatically imported.
- 2) **java.io**: Contains classed for supporting input / output operations.
- 3) **java.util**: Contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations.
- 4) **java.applet**: Contains classes for creating Applets.
- 5) **java.awt**: Contain classes for implementing the components for graphical user interfaces (like button , ;menus etc).
- 6) **java.net**: Contain classes for supporting networking operations.



# ***OBJECT ORIENTED CONCEPTS :***

- *Basic concepts of OOP are :*
  1. *Objects, Method and Classes*
  2. *Data Abstraction*
  3. *Encapsulation*
  4. *Inheritance*
  5. *Polymorphism*
  6. *Dynamic Binding*

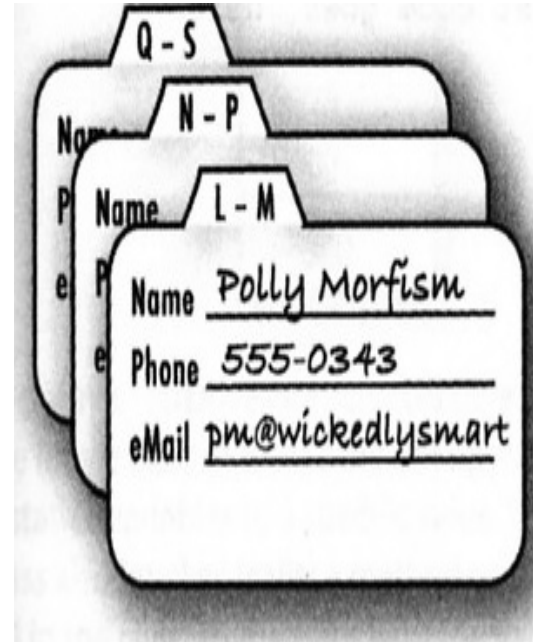


## *1. Objects, methods and classes :-*

- *A class is a blueprint from which individual objects are created.*
- *Objects are basic runtime entities in an object-oriented system.*
- *They may represent a person, a place, a bank account, a table of data or any item that the program may handle.*
- *Once a class has been defined we can create any number of objects belonging to that class.*
- *Objects have unique identity, state and behavior. Example: A dog has states - color, name, breed as well as behaviors – wagging the tail, barking, eating.*
- *method is nothing but the operation that an object can perform it.*



- *Entire address book is your class.*
- *An object is like one entry in your address book.*
- *Methods of the class are the things done to a particular class i.e., getName(), setName().*



## ***2.Data Abstraction:-***

- ✓ *Abstraction is process of hiding the implementation details and showing only the functionality.*
- ✓ *A doctor sees (abstracts) the person as patient name, height, weight, age, blood group, previous or existing diseases etc.*
- ✓ *An employer sees (abstracts) a person as Employee. name, age, health, degree of study, work experience etc.*



### **3. Encapsulation:-**

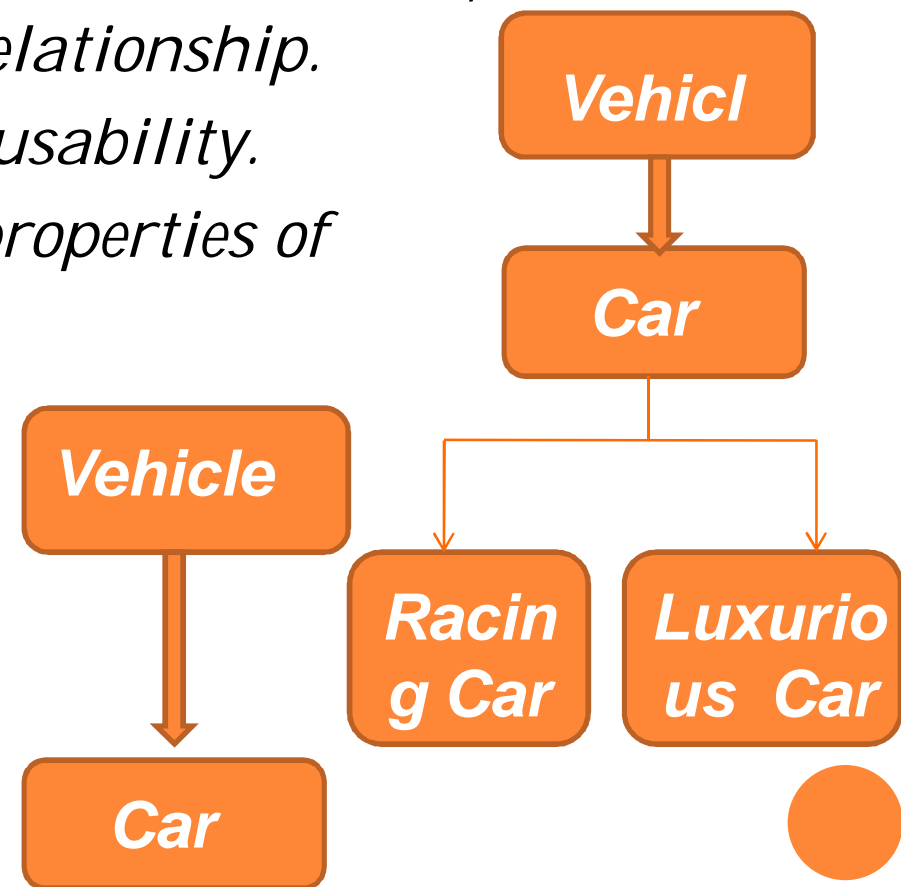
- ✓ *Wrapping up to member data and member function in a single unit (class) is called encapsulation.*
- ✓ *The data is not accessible to the outside world and only those methods, which are wrapped in the class, can access it.*
- ✓ *These methods provide the interface between the object's data and the program.*
- ✓ *This insulation of the data from direct access by the program is called data hiding.*
- ✓ *In Encapsulation data only performs a specific task and is not concerned about the internal implementation.*





## 4. Inheritance:-

- ✓ *Inheritance is the process by which objects of one class acquire the properties of objects of another class.*
- ✓ *Here we have base class and subclass, which have parent - child relationship.*
- ✓ *It provides the concept of reusability.*
- ✓ *A subclass inherits all the properties of base class.*
- ✓ *In addition to this, it can add its own properties and behavior.*



## 5. Polymorphism:-

- ✓ *Polymorphism means the ability to take more than one forms.*
- ✓ *An operation may exhibit different behavior in different instances.*
- ✓ *Eg: Consider operation of addition. For 2 numbers the operation will generate a sum. If the operands are string, then the operation would produce a third string by concatenation.*



## 6. *Dynamic Binding :-*

- *Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run time.*



## ***STRUCTURE OF JAVA PROGRAM :***

Documentation Section	←	Suggested
Package Statement	←	Optional
Import Statements	←	Optional
Interface Statements	←	Optional
Class Definitions	←	Optional
Main Method Class { Main Method Definition }	←	Essential

## **Documentation Section:**

- This section contains set of comments. We can write comment in java within two ways:
  1. Line comment - //
  2. Block Comment - /\* \*/
  3. Documentation Comment is new comment in java - /\*\* \*/

## **Package Statement:**

- This statement declare the package name and inform the compiler that classes
- declared is contain into this package.
- Ex: package student;

## **Import statement:**

- This is similar to the #include statement in the c language. It instruct to the interpreter to load the class contained in the given package.
- Ex: import student.test;
- This statement instruct the interpreter to load the test class contained in the student package.



## **Interface Statement:**

- An interface is like a class but include group of methods. Interface is a new concept of java. This is optional and implement when we want to implement multiple inheritance.

## **Class Definition:**

- Classes are primary and essential elements of java. Class keyword is used to declare the class. Declare multiple classes in a java program.

## **Main method class**

- It is essential part of java program. It is starting point of java program. The main method creates objects of various classes and establishes communication between them. On reaching the end of main, the program terminate and the control passes back to the operating system

## EXAMPLE OF JAVA PROGRAM :

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

- *Class = Keyword*
- *Public= Access specifier*
- *Static = Keyword*
- *Void = type of modification, it doesn't returns value*
- *(String args[]) = array of string type*
- *System.out.println("Hello"); = Output line*
- *System. = class*
- *out. = Object*
- *println = method or function*



# CLASS

- A class is a user defined data type.
- A *class* is a collection of *fields* (data) and *methods* (procedure or function) that operate on that data.
- The basic syntax for a class definition :

```
class ClassName
{
    [member(field) declaration]
    [methods declaration]
}
```

Example:

```
public class Circle
{
    // my circle class
}
```





## ❖ Add Fields (Data) in Class :


```
public class Circle
{
    double x, y; // centre coordinate
    double r;    // radius of the circle
}
```

## ❖ Add Methods in Class :

```
public class Circle
{
    double x, y; // centre of the circle
    double r;    // radius of circle
    //Methods to return circumference and area
    public double circumference()
    {
        return 2*3.14*r;
    }
    public double area()
    {
        return 3.14 * r * r;
    }
}
```



# *OBJECT*

- Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.
  - Object can be defined as an instance of a class. An object contains an address and takes up some space in memory.
  - An object has three characteristics:
    - 1) state:** represents data (value) of an object.
    - 2) behavior:** represents the behavior (functionality) of an object such as deposit, withdraw etc.
    - 3) identity:** Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But, it is used internally by the JVM to identify each object uniquely.
- 

- For Example: Pen is an object. Its name is Reynolds, color is white etc. known as its state. It is used to write, so writing is its behavior.
- Object in java is created using **new** operator.
- Example:  
    Circle c1;     // Declare an object  
    c1 = new Circle ( ) ;     // instantiate the object
- Both statements can be combined into one as shown below:  
    Circle c1 = new Circle();  
    Circle c2 = new Circle();
- We can create **any number** of objects of the class.



## ❖ Accessing Class Members :

- Each object contains class variables and methods before using this we should assign values to these variables in order to use them in our program.
- Here . (dot) operator is used to access class variables and methods with an object.

### Syntax:

Objectname.variblename = vaue ;

Objectname.methodname(parameter list) ;

### ○ Example:

```
Circle c1 = new Circle();
```

```
Circle c2 = new Circle();
```

```
c1.r = 5;
```

```
c2.r = 3;
```



## ❑ Example:-

```
class Student
{
    int id;
    String name;
}
class TestStudent2
{
    public static void main(String args[])
    {
        Student s1=new Student();
        s1.id=101;
        s1.name="Sonu";
        System.out.println(s1.id+" "+s1.name); //printing members
    }
}
```

## Output:-

101 Sonu



## ***THIS KEYWORD***

- There can be a lot of usage of **java this keyword**. In java, this is a **reference variable** that refers to the current object.
- If instance variable and local variables are same then you can use this keyword.
- ❖ *use of java this keyword :-*
  - this can be used to refer current class instance variable.
  - this can be used to invoke current class method (implicitly)
  - this() can be used to invoke current class constructor.



### □ Example :-

```
class demothis
{
    int x,y;
    demothis(int x,int y)
    {
        this.x=x;
        this.y=y;
    }
}
class demotest
{
    public static void main(String args[])
    {
        demothis d1=new demothis(20,10);
        System.out.println("The value of x is" +d1.x);
        System.out.println("The vlaue of y is" +d1.y);
    }
}
```

### Output :-

**The value of x is20**

**The vlaue of y is10**



## *INNER CLASS*

- Creating an inner class is quite simple. You just need to write a class within a class. Unlike a class, an inner class can be private and once you declare an inner class private, it cannot be accessed from an object outside the class.
- Following is the program to create an inner class and access it.





```
class Outer
{
    int data=30;
    class Inner
    {
        void msg()
        {
            System.out.println("data is "+data);
        }
    }
    public static void main(String args[])
    {
        Outer out=new Outer();
        Outer.Inner in=out.new Inner();
        in.msg();
    }
}
```

**Output :-**  
data is 30



# ***INHERITANCE***

- Inheritance in java is that you can create new classes from existing classes.
- When you inherit from an existing class, you can reuse methods and fields of parent class. Moreover, you can add new methods and fields in your current class also.
- It is also known as *parent-child* relationship.
- In the terminology of Java, a class that is inherited is called a superclass. The new class is called a subclass.
- **Syntax:-**

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



### **Example:**

```
class Employee
{
    float salary=40000;
}

class Programmer extends Employee
{
    int bonus=10000;

    public static void main(String args[])
    {
        Programmer p=new Programmer();
        System.out.println("Programmer salary is:"+p.salary);
        System.out.println("Bonus of Programmer is:"+p.bonus);
    }
}
```

### **Output:**

Programmer salary is:40000.0

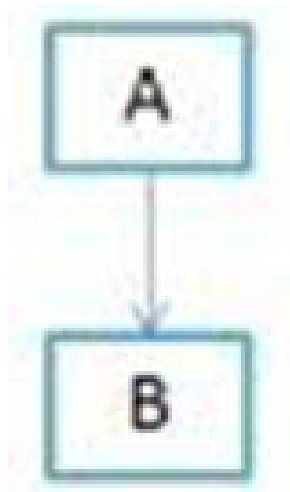
Bonus of programmer is:10000



## TYPE OF INHERITANCE

### 1) Single Inheritance :-

**Single inheritance** is easy to understand. When one class extends another class then it is known as single inheritance. The below 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.



*Parent Class or Super Class or Base Class*

*Child Class or derived class or sub class*



## ❑ Example:-

Class A

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

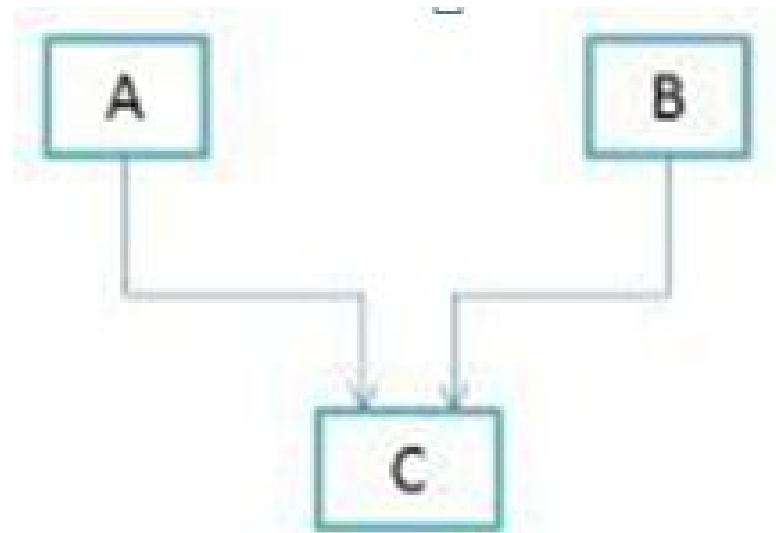
Class B extends A

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



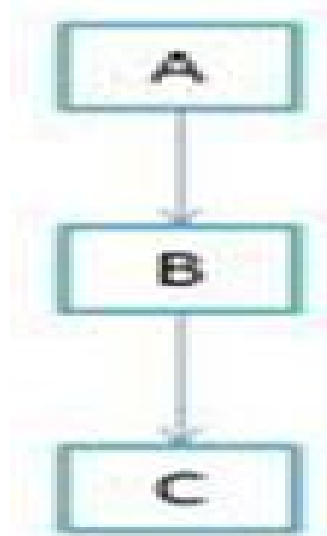
## 2) Multiple Inheritance :-

**“Multiple Inheritance”** refers to the concept of one class extending (Or inherits) more than one base class. The problem with “multiple inheritance” is that the derived class will have to manage the dependency on more than one base classes.



### 3) Multilevel Inheritance :-

One class is derived from base class and another class is derived from derived class. As you can see in below flow diagram C is subclass or child class of B and B is a child class of A.



## ❑ Example :-

Class X

```
{  
    public void methodX()  
    {  
        System.out.println("Class X method");  
    }  
}
```

Class Y extends X

```
{  
    public void methodY()  
    {  
        System.out.println("class Y method");  
    }  
}
```

Class Z extends Y

```
{  
    public void methodZ()  
    {  
        System.out.println("class Z method");  
    }  
}
```

public static void main(String args[])

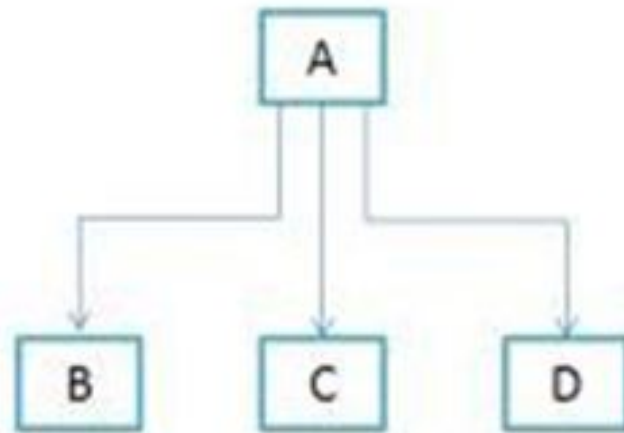
```
{  
    Z obj = new Z();  
    obj.methodX();  
    obj.methodY();  
    obj.methodZ();  
}  
}
```





## 4) Hierarchical Inheritance

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.



❑ Example :-

Class A

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

Class B extends A

```
{  
    public void methodB()  
    {  
        System.out.println("method of Class B");  
    }  
}
```

Class C extends A

```
{  
    public void methodC()  
    {  
        System.out.println("method of Class C");  
    }  
}
```

Class D extends A

```
{  
    public void methodD()  
    {  
        System.out.println("method of Class D");  
    }  
}
```



```
Class MyClass
{
    public static void main(String args[])
    {
        B obj1 = new B();
        C obj2 = new C();
        D obj3 = new D();

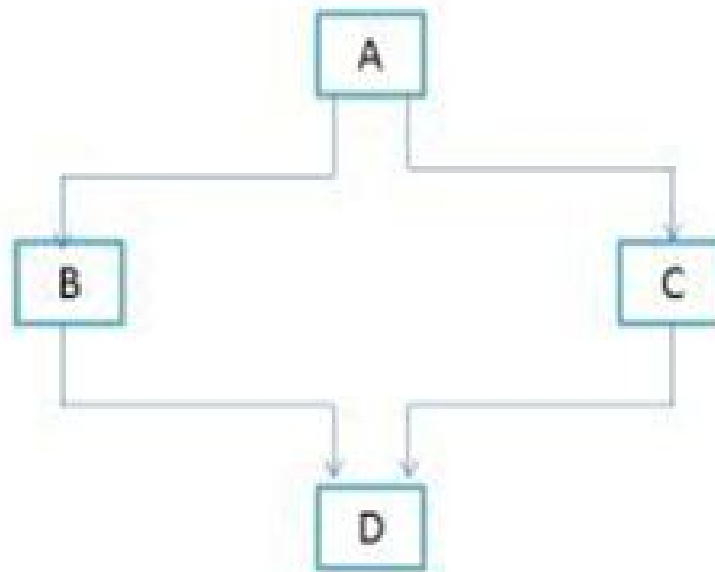
        obj1.methodA();
        obj2.methodA();
        obj3.methodA();
    }
}
```

Output:  
method of Class A  
method of Class A  
method of Class A



## 5) Hybrid Inheritance :-

In simple terms you can say that Hybrid inheritance is a combination of **all type of inheritance**. By using **interfaces** you can have multiple as well as **hybrid inheritance** in Java.



# ***POLYMORPHISM***

- One form can be used for more than one purpose then it is known as polymorphism.
- For Example : (+) sign is used to addition of two numbers as well as concatenate two strings.
- Polymorphism is derived from 2 greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.
- There are two types of polymorphism in java: compile time polymorphism and runtime polymorphism.
- We can perform polymorphism in java by method overloading and method overriding.



## OVERLOADING :-

- If a class has multiple methods having same name but different in parameters and return type, it is known as **Method Overloading**.
- Suppose you have to perform addition of the given numbers but there can be any number of arguments, you can write the method such as `add(int,int)` for two parameters, and `add(int,int,int)` for three parameters.
- There are two ways to overload the method in java
  - 1)By changing number of arguments
  - 2)By changing the data type



## ❑ Example :-

```
class Adder
{
    int add(int a,int b)
    {
        return a+b;
    }
    int add(int a,int b,int c)
    {
        return a+b+c;
    }
}


class Test1
{
    public static void main(String args[] )
    {
        Adder a1=new Adder();
        System.out.println(a1.add(5,10));
        System.out.println(a1.add(10,20,30));
    }
}
```

## Output :-

15  
60



## OVERRIDING :-

- 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.
  - 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)
  - One significant advantage of method overriding is that a class can give its specific execution to an inherited method without having the modification in the parent class
  - It also hide methods and fields of parent class.
- 



### ❑ Example :-

```
class Vehicle
```

```
{  
    void run()  
    {  
        System.out.println("Vehicle is running");  
    }  
}
```

```
class Bike2 extends Vehicle
```

```
{  
    void run()  
    {  
        System.out.println("Bike is running safely");  
    }  
  
    public static void main(String args[])  
    {  
        Bike2 obj = new Bike2();  
        obj.run();  
    }  
}
```

### **Output :-**

Bike is running safely



## ***SUPER KEYWORD***

- The **super** keyword in java is a reference variable which is used to refer immediate parent class object.
- Usage of java super Keyword
  - 1) super can be used to refer immediate parent class instance variable.
  - 2) super can be used to invoke immediate parent class method.
  - 3) super() can be used to invoke immediate parent class constructor.



### ❑ Example :-

```
class Animal
{
String color="white";
}
class Dog extends Animal
{
String color="black";
    void printColor()
    {
        System.out.println(color);    //prints color of Dog class
        System.out.println(super.color);    //prints color of Animal class
    }
}
class TestSuper1
{
public static void main(String args[])
{
Dog d=new Dog();
d.printColor();
}
}
```

### Output:-

black  
white



## ***FINAL KEYWORD***

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

- 1) variable
- 2) method
- 3) class

- **Final Variable:-**

If you make any variable as final, you cannot change the value of final variable(It will be constant).



## ❑ Example:-

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

## Output :-

Compile Time Error



## ❑ Final Method:-

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

- **Example:-**

```
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 h= new Honda();  
        h.run();  
    }  
}
```



## ❑ Final Class:-

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

### • **Example:-**

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



## *ANONYMOUS INNER CLASS*

- A class that have no name is known as anonymous inner class in java. It should be used if you have to override method of class or interface. Java Anonymous inner class can be created by two ways:

- 1) Class (may be abstract or concrete).

- 2) Interface

- Syntax :-

// Test can be interface,abstract/concrete class

```
Test t = new Test()
```

```
{
```

```
    // data members and methods
```

```
    public void test_method()
```

```
    {
```

```
        ..... 
```

```
    }
```

```
};
```





## ❑ Example Using Class :-

```
abstract class Person
{
    abstract void eat();
}
class Test
{
    public static void main(String args[])
    {
        Person p=new Person()
        {
            public void eat()
            {
                System.out.println("nice fruits");
            }
        };
        p.eat();
    }
}
```

**Output :-**  
nice fruits





# *GENERIC INTERFACE*

## ***LIST IMPLEMENTATIONS***

- All methods in the Collection interface are also available in the List interface.
- You can choose between the following List implementation in the Java Collections API :
  - 1) `java.util.ArrayList`
  - 2) `java.util.LinkedList`
  - 3) `java.util.Vector`
  - 4) `java.util.Stack`
- Here are a few examples of how to create a List instance:  
`List listA = new ArrayList();`  
`List listB = new LinkedList();`  
`List listC = new Vector();`  
`List listD = new Stack();`



## ❑ Adding and Accessing Elements :

```
List listA = new ArrayList();

listA.add("element 0");
listA.add("element 1");
listA.add("element 2");

//access via index
String element0 = listA.get(0);
String element1 = listA.get(1);
String element3 = listA.get(2);

//access via Iterator
Iterator iterator = listA.iterator();
while(iterator.hasNext()){
    String element = (String) iterator.next();
}

//access via new for-loop
for(Object object : listA) {
    String element = (String) object;
}
```



## ❑ Removing Elements :

You can remove elements in two ways:

1) `remove(Object element)` :-

removes that element in the list, if it is present.

2) `remove(int index)` :-

removes the element at the given index.

## ❑ Clearing List :

`clear()` method which removes all elements from the list.

Ex :- `list.clear();`

## ❑ List Size :

`Size()` is used to obtain number of elements in list.

Ex :- `int size = list.size();`



# GENERICS

- Before generics, we can store any type of objects in collection i.e. non-generic. Now generics, forces the java programmer to store specific type of objects.
- classes like HashSet, ArrayList, HashMap, etc use generics very well. We can use them for any type.
- we use <> to specify parameter types in generic class creation. To create objects of generic class, we use following syntax.
- **Syntax:-** BaseType <Type> obj = new BaseType <Type>()
- ❖ Advantages of Generics :-
  - 1) **Type-safety** : We can hold only a single type of objects in generics. It doesn't allow to store other objects.
  - 2) **Type casting is not required:** There is no need to typecast the object.
  - 3) **Compile-Time Checking:** It is checked at compile time so problem will not occur at runtime.



## ○ Type Parameters :-

The type parameters naming conventions are important to learn generics thoroughly. The commonly type parameters are as follows:

<T> - Type

<E> - Element

<K> - Key

<N> - Number

<V> - Value



### ❑ Example :-

```
import java.util.*;
class Test
{
    public static void main(String args[])
    {
        ArrayList<String> list=new ArrayList<String>();
        list.add("Prakash");
        list.add("Rohan");

        String s=list.get(1);    //type casting is not required
        System.out.println("element is: "+s);

        Iterator<String> itr=list.iterator();
        while(itr.hasNext())
        {
            System.out.println(itr.next());
        }
    }
}
```

### Output :-

element is: Rohan  
Prakash  
Rohan





## *ADAPTER CLASSES*

- Java adapter classes provide the default implementation of listener interfaces. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it saves code.
- The adapter classes are found in `java.awt.event`, `java.awt.dnd` and `javax.swing.event` packages. The Adapter classes with their corresponding listener interfaces are given below.



## ***JAVA.AWT.EVENT ADAPTER CLASSES***

<b>Adapter class</b>	<b>Listener interface</b>
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

# ***JAVA MOUSEADAPTER EXAMPLE***

```
import java.awt.*;
import java.awt.event.*;
public class MouseAdapterExample extends MouseAdapter{
    Frame f;
    MouseAdapterExample(){
        f=new Frame("Mouse Adapter");
        f.addMouseListener(this);

        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
    public void mouseClicked(MouseEvent e) {
        Graphics g=f.getGraphics();
        g.setColor(Color.BLUE);
        g.fillOval(e.getX(),e.getY(),30,30);
    }
}
```



# ***JAVA MOUSEADAPTER EXAMPLE***

```
public static void main(String[ ] args) {  
    new MouseAdapterExample();  
}  
}
```

***OUTPUT:***

