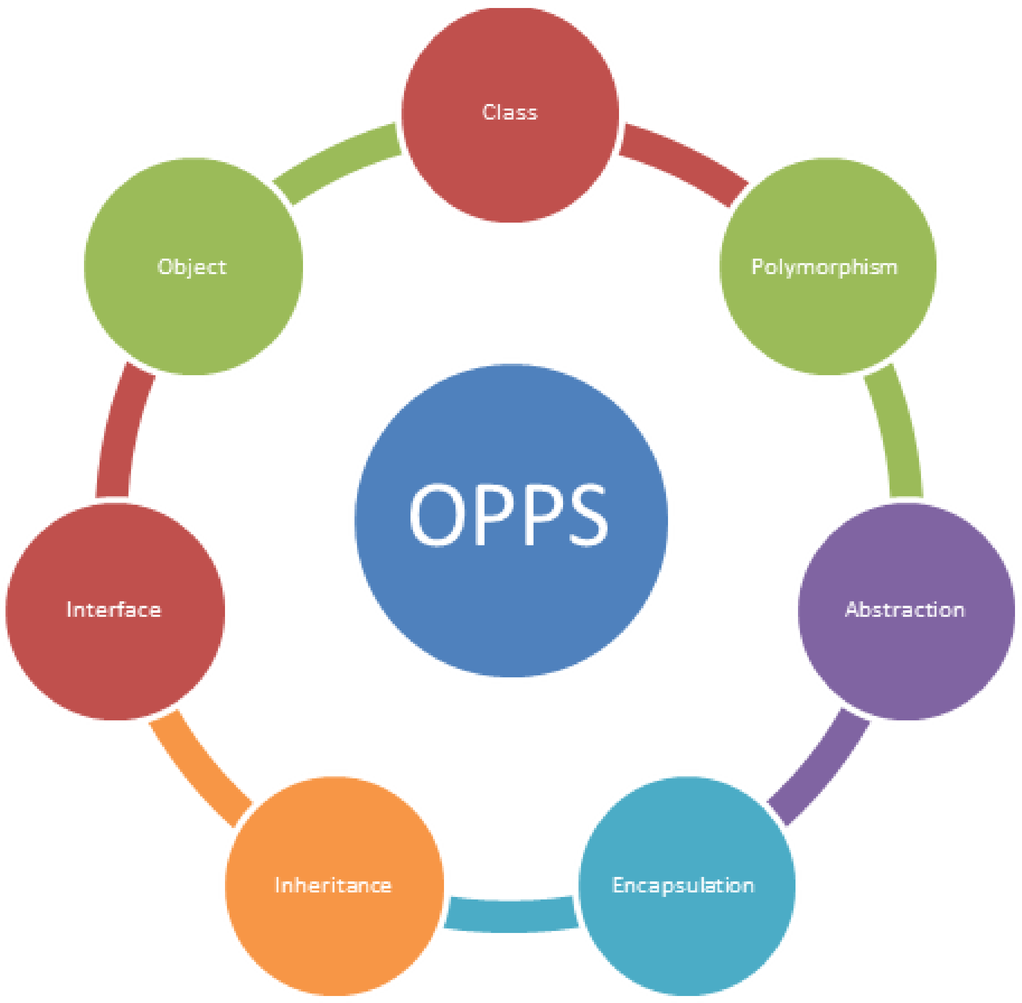
OBJECT-ORIENTED PROGRAMMING SYSTEM



❖ **OOP**S refers to a programming methodology based on objects, instead of just functions and procedures. These objects are organized into classes, which allow individual objects to be grouped together.

- ❖ OOPS simplifies the software development and maintenance by providing some concepts:
  1.Object
  2.Class
  3.Inheritance
  4.Polymorphism
  5.Abstraction
  6.Encapsulation

DEFINITION'S:

Object:

- ❖ Any entity that has state and behavior is known as an object. For java: chair, pen, table, Keyboard, bike etc. It can be physical and logical.

Class:

- ❖ Collection of objects is called class. It is a logical entity.

Inheritance:

- ❖ When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism:

- ❖ When one task is performed by different ways i.e. known as polymorphism.
  For java: to convince the customer differently, to draw something e.g. shape or rectangle etc.In java, we use method overloading and method overriding to achieve polymorphism. Another java can be to speak something e.g. cat speaks meaw, dog barks woof etc.

Abstraction:

- ❖ Hiding internal details and showing functionality is known as abstraction.

For java: phone call, we don't know the internal processing.

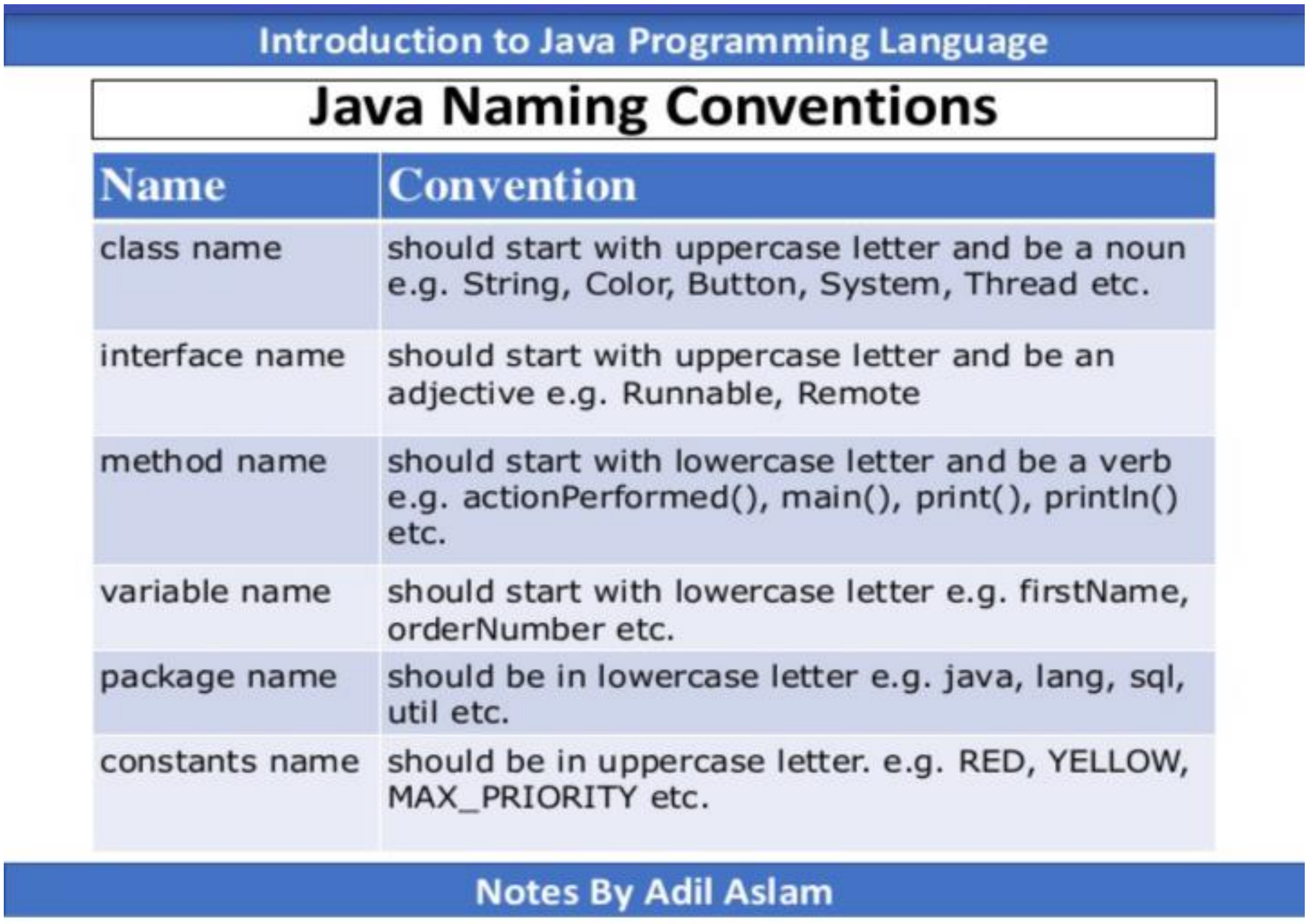
In java, we use abstract class and interface to achieve abstraction.

Encapsulation:

Binding (or wrapping) code and data together into a single unit is known as encapsulation.

For java: capsule, it is wrapped with different medicines.

Naming Conventions:



Introduction to Java Programming Language

Java Naming Conventions

Name	Convention
class name	should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.
interface name	should start with uppercase letter and be an adjective e.g. Runnable, Remote
method name	should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc.
variable name	should start with lowercase letter e.g. firstName, orderNumber etc.
package name	should be in lowercase letter e.g. java, lang, sql, util etc.
constants name	should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc.

Notes By Adil Aslam

- ❖ Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method etc.

All the classes, interfaces, packages, methods and fields of java programming language are according to java naming convention.

class name:
should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.

interface name:
should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc.

method name:
should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc.

variable name:
should start with lowercase letter e.g. firstName, orderNumber etc.

package name:
should be in lowercase letter e.g. java, lang, sql, util etc.

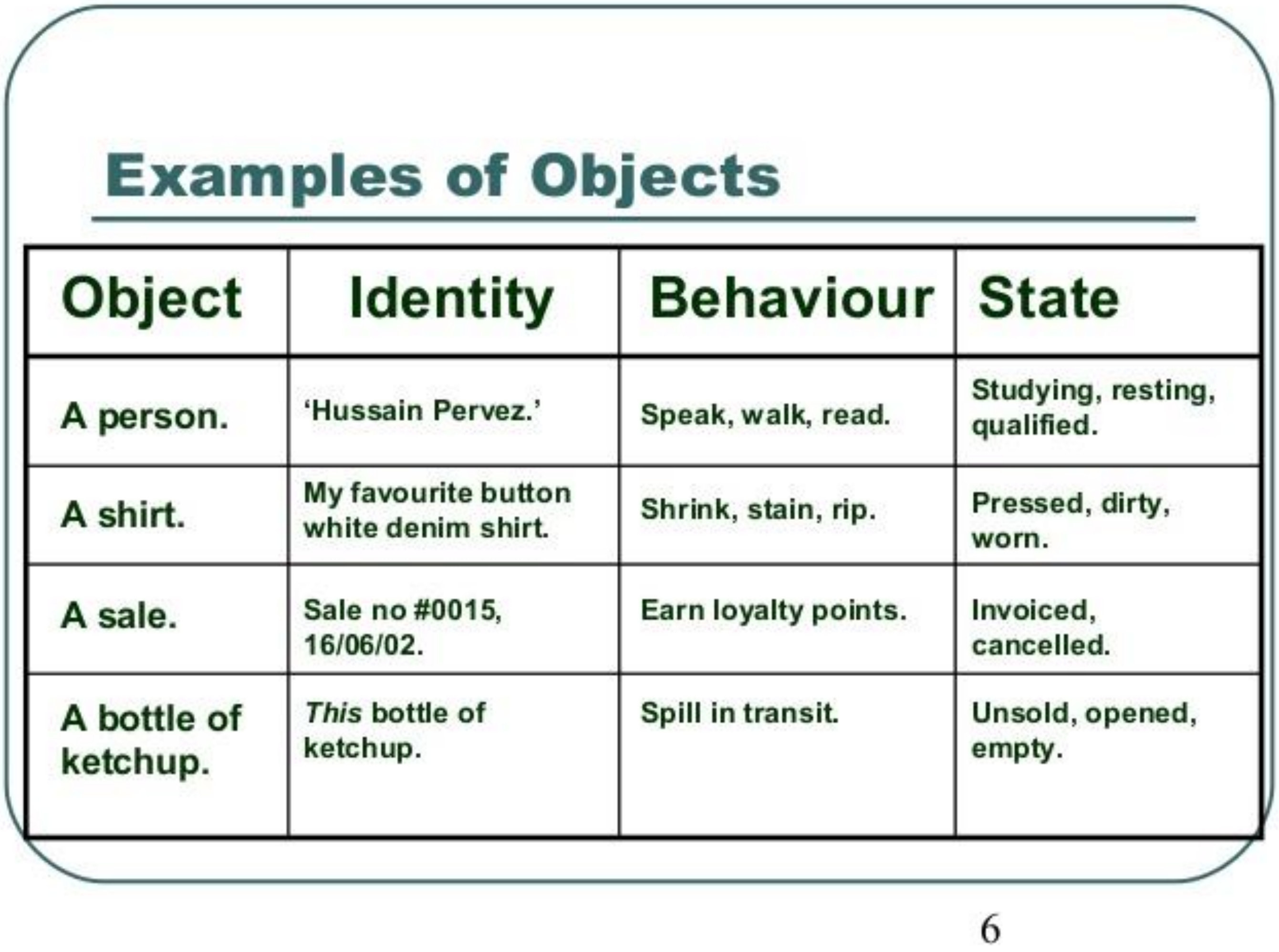
constants name:
should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc.

Object

An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tangible and intangible)

An object has three characteristics:
state:represents data (value) of an object.
behavior: represents the behavior (functionality) of an object such as deposit, withdraw etc.
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.



Examples of Objects

Object	Identity	Behaviour	State
A person.	'Hussain Pervez.'	Speak, walk, read.	Studying, resting, qualified.
A shirt.	My favourite button white denim shirt.	Shrink, stain, rip.	Pressed, dirty, worn.
A sale.	Sale no #0015, 16/06/02.	Earn loyalty points.	Invoiced, cancelled.
A bottle of ketchup.	*This* bottle of ketchup.	Spill in transit.	Unsold, opened, empty.

6

Class

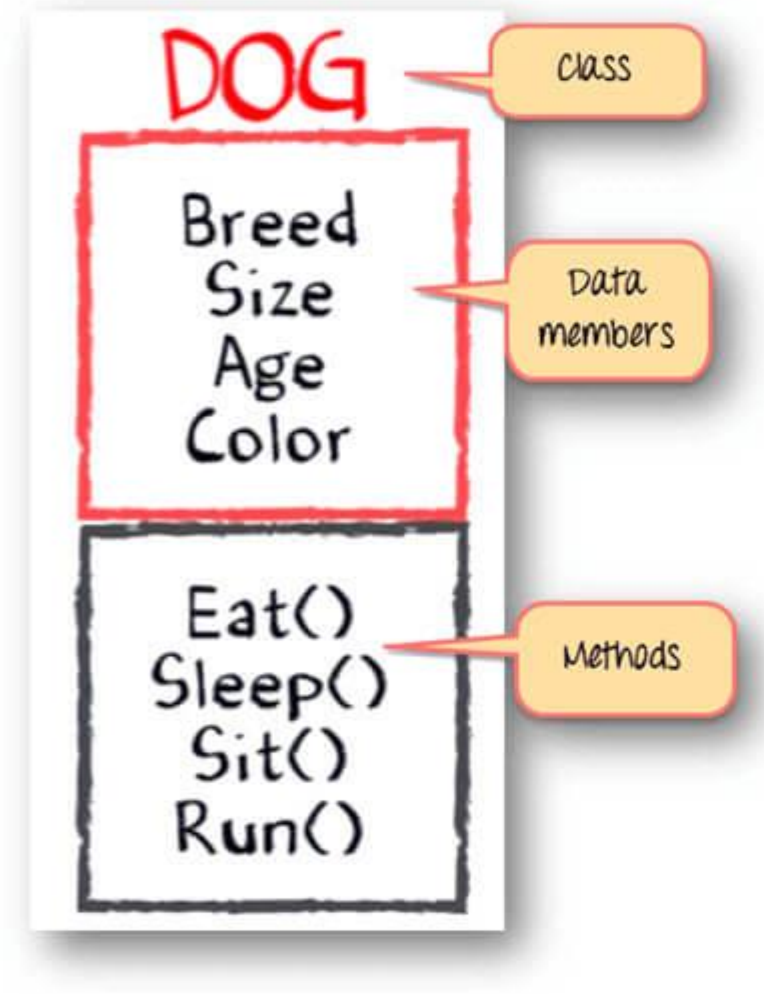
A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:
1.fields
2.methods
3.constructors
4.blocks
5.nested class and interface

```
class Student{
int id;//field or data member or instance variable
String name;

public static void main(String args[]){
  Student s1=new Student();//creating an object of Student
  System.out.println(s1.id);//accessing member through reference variable
  System.out.println(s1.name);
}
}
```

Output:

```
0
null
```



Constructor

1.In Java, constructor is a block of codes similar to method. It is called when an instance of object is created and memory is allocated for the object.

2. It is a special type of method which is used to initialize the object.

3. Everytime an object is created using new() keyword, at least one constructor is called. It is called a default 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

There are two types of constructors in java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

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

Output:

```
Bike is created
```

Parameterized Constructor

1.A constructor which has a specific number of parameters is called parameterized constructor.

2. Parameterized constructor is used to provide different values to the distinct objects.

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

Just like method overloading, constructors also can be overloaded. Same constructor declared with different parameters in the same class is known as constructor overloading.

```
class Student5{
    int id;
    String name;
    int age;

    Student5(int i,String n){
```

```
    id = i;
    name = n;
    }

    Student5(int i,String n,int a){
    id = i;
    name = n;
    age=a;
    }
    void display()
{
System.out.println(id+" "+name+" "+age);
}
    public static void main(String args[]){
    Student5 s1 = new Student5(111,"Karan");
    Student5 s2 = new Student5(222,"Aryan",25);
    s1.display();
    s2.display( );
   }
}
```

Output:

```
111 Karan 0
222 Aryan 25
```

Copy Constructor

1. There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++.

2. There are many ways to copy the values of one object into another in java. They are:

By constructor
By assigning the values of one object into another
By clone( ) method of Object class

Example :

```
class Student6{
    int id;
    String name;
    Student6(int i,String n){
    id = i;
    name = n;
    }

    Student6(Student6 s){
    id = s.id;
    name =s.name;
    }
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){
    Student6 s1 = new Student6(111,"Karan");
    Student6 s2 = new Student6(s1);
    s1.display();
    s2.display();
   }
}
```

Output:

```
111 Karan
111 Karan
```

This Keyword

1.In java, this is a reference variable that refers to the current object.this is a keyword.

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.

6. this can be used to return the current class instance from the method.

```
class Test
{
    int a;
    int b;

    // Parameterized constructor
    Test(int a, int b)
    {
        this.a = a;
        this.b = b;
    }
    void display()
    {
        //Displaying value of variables a and b
        System.out.println("a = " + a + "  b = " + b);
    }
    public static void main(String[] args)
    {
        Test object = new Test(10, 20);
        object.display();
    }
```

Output:

```
a = 10  b = 20
```

Static Variable

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

2. 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.

3.The static variable gets memory only once in class area at the time of class loading.

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

```
class Student8{
  int rollno;
  String name;
  static String college ="ITS";
  Student8(int r,String n){
  rollno = r;
  name = n;
  }
void display (){System.out.println(rollno+" "+name+" "+college);}

public static void main(String args[]){
Student8 s1 = new Student8(111,"Karan");
Student8 s2 = new Student8(222,"Aryan");

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

Output:

```
111 Karan ITS
222 Aryan ITS
```

Inheritance

Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.

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 parent class, and you can add new methods and fields also

Terms used in Inheritence:

Class:

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

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.

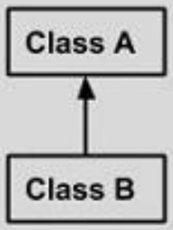
Reusability:

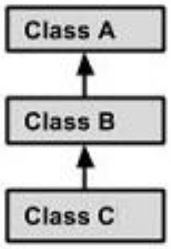
As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in previous class.

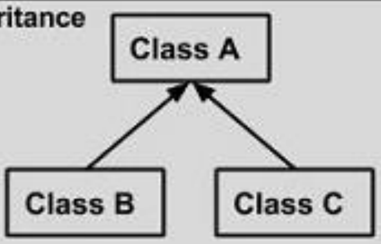
```
class Employee{
float salary=40000;
}
class Programmer extends Employee{
int bonus=10000;
public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

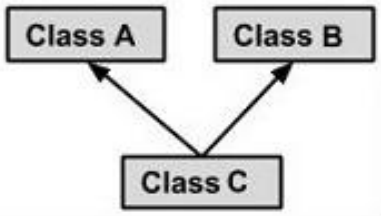
Output:

```
Programmer salary is:40000.0
Bonus of programmer is:10000
```

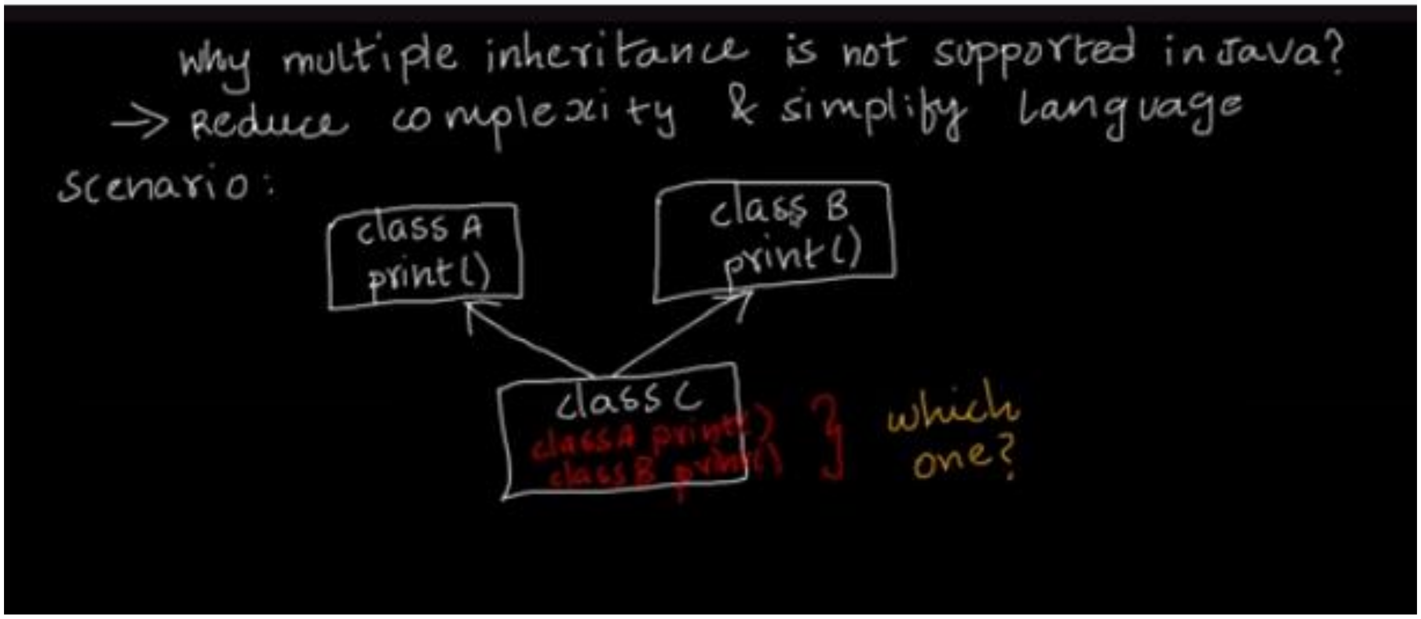
Single Inheritance 	public class A {
	  ....
	}
	public class B **extends** A {
	  ....
	}

Multi Level Inheritance 	public class A { ....}
	
	public class B **extends** A {....................}
	
	public class C **extends** B {..................... }

Hierarchical Inheritance 	public class A { ....}
	
	public class B **extends** A {....................}
	
	public class C **extends** A {..................... }

Multiple Inheritance 	public class A { ....}
	
	public class B {....................}
	
	public class C **extends** A,B {
	  ....
	} // Java does not support mutiple Inheritance

why multiple inheritance is not supported in java ?


why multiple inheritance is not supported in Java?

→ Reduce complexity & simplify language

Scenario :

Single Inheritance

When a class **extends** another one class only then we call it a single inheritance.

```
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
  }
}
```

Output:

```
Base class method
Child class method
```

Multilevel Inheritance

When a class extends a class, which extends anther class then this is called multilevel inheritance.

```
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(); //calling grand parent class method
     obj.methodY(); //calling parent class method
     obj.methodZ(); //calling local method
```

```
  }
}
```

Output:

```
class X method
class Y method
class Z method
```

Hierarchical Inheritance

If more than one class is inherited from the base class, it's known as hierarchical inheritance. In hierarchical inheritance, all features that are common in child classes are included in the base class. For java: Physics, Chemistry, Biology are derived from Science class.

```
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 Javajava
{
  public static void main(String args[])
  {
     B obj1 = new B();
     C obj2 = new C();
     D obj3 = new D();
     //All classes can access the method of class A
     obj1.methodA();
     obj2.methodA();
     obj3.methodA();
  }
}
```

Output:

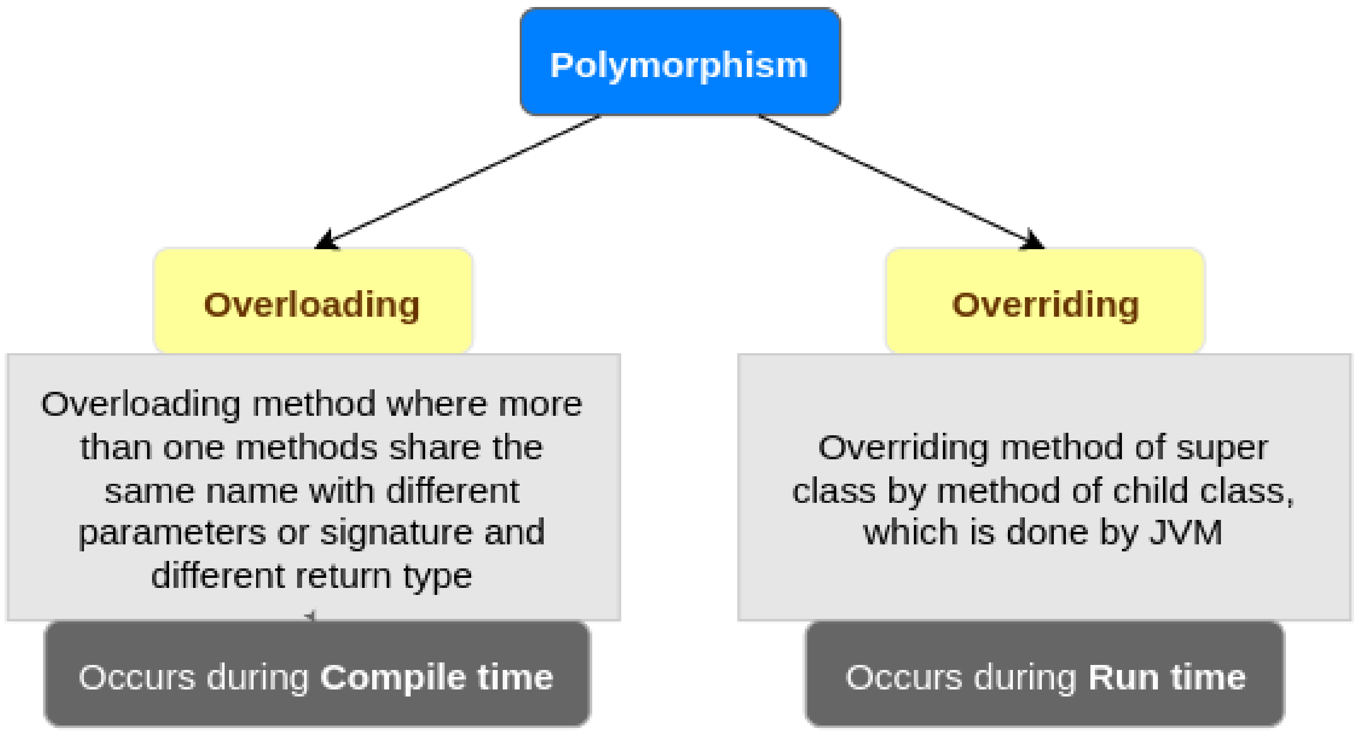
```
method of Class A
method of Class A
method of Class A
```

Polymorphism

- Polymorphism in java is a concept by which we can perform a single action by 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.

- 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.

❖ If you overload static method in java, it is the java of compile time polymorphism. Here, we will focus on runtime polymorphism in java.


Polymorphism
Overloading
Overriding
Overloading method where more than one methods share the same name with different parameters or signature and different return type
Overriding method of super class by method of child class, which is done by JVM
Occurs during Compile time
Occurs during Run time

Method Overriding

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

Usage of Java Method Overriding:

1. Method overriding is used to provide specific implementation of a method that is already provided by its super class.
2. 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.

```
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
```

Method Overloading

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

There are two ways to overload the method in java:

1. By changing number of arguments
2. By changing the data type

1) Method Overloading: changing no. of arguments:

1. In this java, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.
2. In this java, we are creating static methods so that we don't need to create

instance for calling methods.

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[ ] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
```

Output:

```
22
33
```

2) Method Overloading: changing data type of arguments:

In this java, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

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

{
return a+b;
}
}
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}}
```

Output:

```
22
24.9
```

Final Variable

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

```
class Bike9{
final int speedlimit=90;//final variable
void run(){
  speedlimit=400;
}
public static void main(String args[]){
Bike9 obj=new  Bike9();
obj.run();
}  }//end of class
```

Output:

```
Compile Time Error
```

Final Class

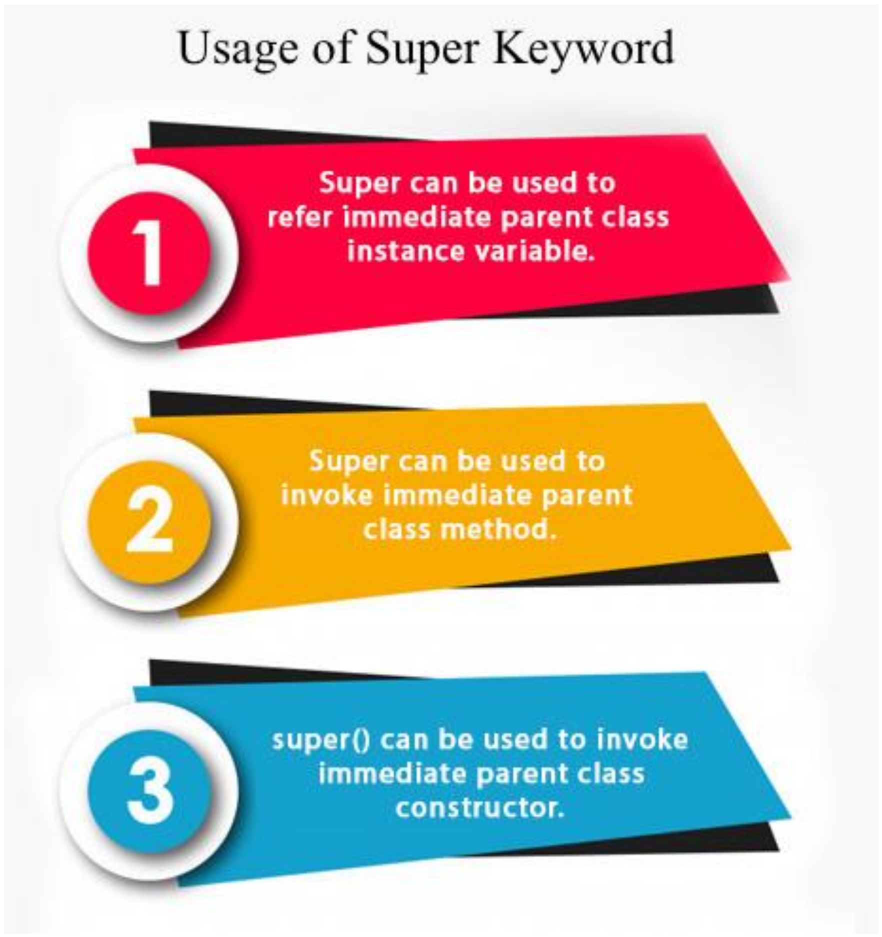
If you make any class as final, you cannot extend it.

```
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();
  }  }
```

Output:

```
Compile Time Error
```

Super Keyword


Usage of 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.

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.

1) super is used to refer immediate parent class instance variable.

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

```
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
```

2) super can be used to invoke parent class method

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

```
class Animal{
void eat( ){System.out.println("eating...");}
}
class Dog extends Animal{
void eat( ){System.out.println("eating bread...");}
void bark( ){System.out.println("barking...");}
void work( ){
super.eat( );
bark( );
}
}
class TestSuper2{
public static void main(String args[ ]){
Dog d=new Dog( );
d.work( );
}}
```

Output:

```
eating...
barking...
```

3) super is used to invoke parent class constructor.

1. The super keyword can also be used to invoke the parent class constructor. Let's see a simple java:

```
class Animal{
Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
Dog(){
super();
System.out.println("dog is created");
}
}
class TestSuper3{
public static void main(String args[ ]){
Dog d=new Dog();  }}
```

Output:

```
animal is created
dog is created
```

Instanceof Operator

1. The java instanceof operator is used to test whether the object is an instance of the specified type (class or subclass or interface).
2. The instanceof in java is also known as type comparison operator because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has null value, it returns false.

```
class Animal{}
class Dog1 extends Animal{//Dog inherits Animal
public static void main(String args[]){
Dog1 d=new Dog1();
System.out.println(d instanceof Animal);//true
}
}
```

Output:

```
true
```

Encapsulation

1. Encapsulation in java is a process of wrapping code and data together into a single unit, for example capsule i.e. mixed of several medicines.
2. Encapsulation in java
We can create a fully encapsulated class in java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

The Java Bean class is the example of fully encapsulated class.

Advantage of Encapsulation in java:

By providing only setter or getter method, you can make the class read-only or write-only.
It provides you the control over the data. Suppose you want to set the value of id i.e. greater than 100 only, you can write the logic inside the setter method.

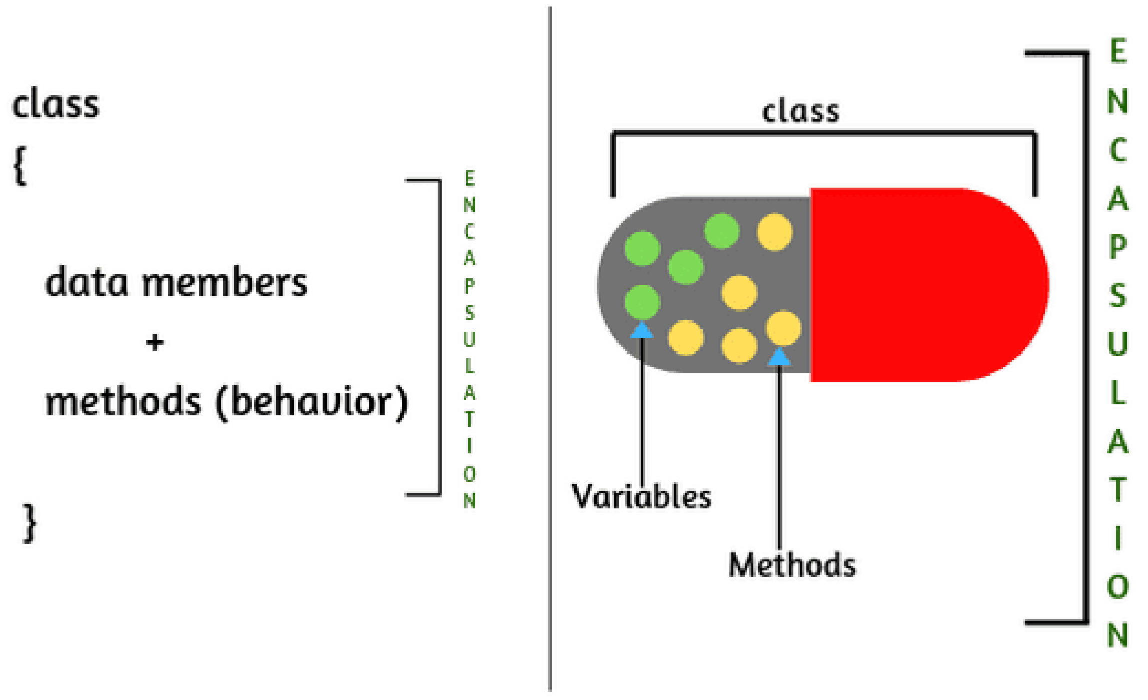


Fig: Encapsulation

```
package com.javatpoint;
public class Student{
private String name;

public String getName(){
return name;
}
public void setName(String name){
this.name=name
}  }
```

```
//save as Test.java
package com.javatpoint;
class Test{
public static void main(String[] args){
Student s=new Student();
s.setName("vijay");
System.out.println(s.getName());
}
}
```

Output:

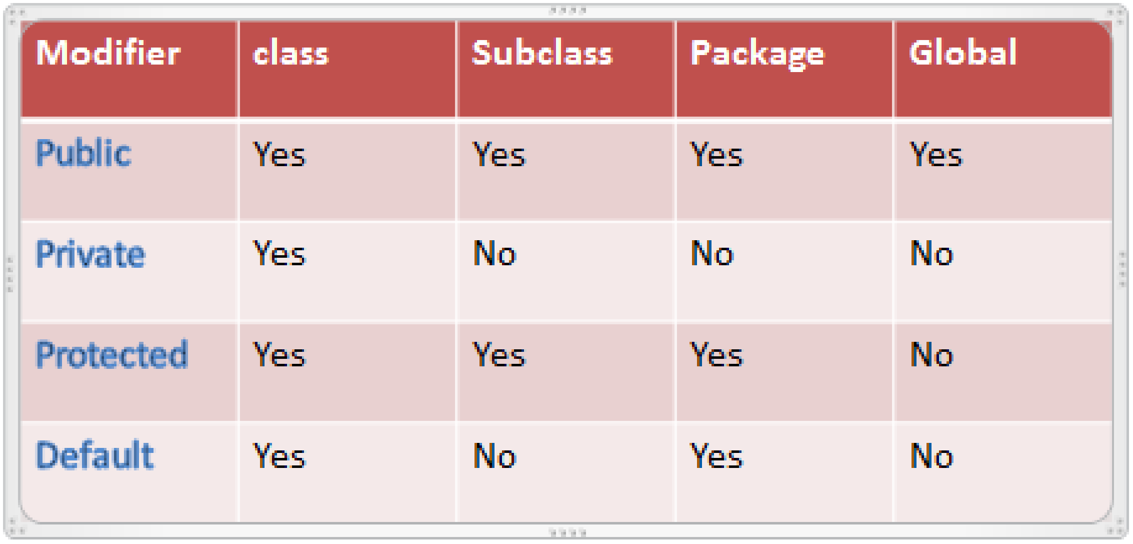
```
vijay
```

Access Modifiers

There are two types of modifiers in java: access modifiers and non-access modifiers. The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. Public



Modifier	class	Subclass	Package	Global
Public	Yes	Yes	Yes	Yes
Private	Yes	No	No	No
Protected	Yes	Yes	Yes	No
Default	Yes	No	Yes	No

Private access Modifier

The private access modifier is accessible only within 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
   }
}
```

Output:

```
Error
```

Default Access Modifier

If you don't use any modifier, it is treated as default bydefault. The default modifier is accessible only within package.

```
//save by A.java
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
  }  }
```

Output:

```
Error
```

Protected Access Modifier

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.

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

//save by B.java
package mypack;
import pack.*;

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

Output:

```
Hello
```

Public Access Modifier

The public access modifier is accessible everywhere. It has the widest scope among all other modifiers.
This modifier doesn't put any restriction on the access.

```
//save by A.java
package pack;
public class A{
public 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();
   obj.msg();
  }
}
```

Output:

```
Hello
```

Another Example

```
public class EncapTest {
   private String name;
   private String idNum;
   private int age;

   public int getAge() {
      return age;
   }
   public String getName() {
      return name;
   }

   public String getIdNum() {
      return idNum;
   }

   public void setAge( int newAge) {
      age = newAge;
   }

   public void setName(String newName) {
      name = newName;
   }

   public void setIdNum( String newId) {
      idNum = newId;
   }
}
public class RunEncap {

   public static void main(String args[]) {
      EncapTest encap = new EncapTest();
      encap.setName("James");
      encap.setAge(20);
      encap.setIdNum("12343ms");

      System.out.print("Name : " + encap.getName() + " Age : " +
encap.getAge());
   }
}
```

Output

```
Name : James Age : 20
```

Abstraction

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only important things to the user and hides the internal details for java sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.

There are two ways to achieve abstraction in java
1. Abstract class (0 to 100%)
2. Interface (100%)

Abstract Class

A class that is declared as abstract is known as abstract class. It needs to be extended and its method implemented. It cannot be instantiated.

```
//abstract parent class
abstract class Animal{
  //abstract method
  public abstract void sound();
}
//Dog class extends Animal class
public class Dog extends Animal{

  public void sound(){
System.out.println("Woof");
  }
```

```
 public static void main(String args[]){
Animal obj = new Dog();
obj.sound();
   }
}
```

Output:

```
Woof
```

Abstract Method

A method that is declared as abstract and does not have implementation is known as abstract method.

```
abstract class Bike{
  abstract void run();
}
class Honda4 extends Bike{
void run(){System.out.println("running safely..");}
public static void main(String args[]){
Bike obj = new Honda4();
obj.run();
}
}
```

Output:

```
running safely..
```

Interface

An interface in java is a blueprint of a class. It has static constants and abstract methods.

The interface in java is a mechanism to achieve abstraction. There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritance in Java.
In other words, you can say that interfaces can have methods and variables but the

methods declared in interface contain only method signature, not body.

There are mainly three reasons to use interface. They are given below:
1. It is used to achieve abstraction.
2. By interface, we can support the functionality of multiple inheritance.
3. It can be used to achieve loose coupling.

Interface is declared by using interface keyword. It provides total abstraction; means all the methods in interface are declared with empty body and are public and all fields are public, static and final by default. A class that implement interface must implement all the methods declared in the interface.

```
interface printable{
void print();
}
class A6 implements printable{
public void print(){System.out.println("Hello");}

public static void main(String args[]){
A6 obj = new A6();
obj.print();
}
}
```

Output:

```
Hello
```