Inheritance:

Inheritance is the mechanism of organizing and structuring program.

Objects are distinguished from each other by some additional features, but there are objects that share certain things common.

In Object oriented Programming classes can inherit some common behavior and state from others. Inheritance in OOP allows to define a class and later to organize some other classes simply adding some details with the old class definition.

In object oriented methodology, inheritance enables you to extend the functionally of an existing class. Create a class that inherits the attributes and behavior of another class. In addition, the new class can consist of a few new attributes and behavior that are specific to the class.

extends keyword: the extends keyword use to inherit the properties of another class.

public class alphabet{

}

publicclass smallAlpha extends alphabet {

}

public class capitalAlpha extends smallAlpha{

}

Invoking Super class Constructor:

If a class inherits properties of another class, then the sub class automatically acquires the properties of the super class. If you want to call parameterized constructor of super class, then we use Super keyword.

super(values);

Super and Subclasses:

In inheritance, a super class or parents class is the one form which another class inherit attributes and behavior. A subclass or child class is a class that inherits attributes and behavior from a super class.

Class: Super class

Air Ticket

Attribute

Date

Destination

Time

Class: Requested

Air Ticket

Attribute

Date

Destination

Time

Own attribute:-

Seat Number

Air Ticket

Attribute

Date

Destination

Time

Own Attribute:-

Status

Class: Confirmed

Sub Class

Super Class inherited by other Two classes :

Sub Class

Example:

class One {

int x, y;

}

class Two extends One {

public void set(int a, int b) {

x=a;

y=b;

}

public void show() {

System.*out*.print(x+ "\t" +y);

}

}

class UseInheritance {

public void main (String[] args) {

Two t=new Two ();

t.set (10, 20);

t.show();

}

}

Output :

10

20

Types of Inheritance:

1). Single Inheritance:

Class A

Class B

2). Multi Level Inheritance:

CLASS A

CLASS B

CLASS C

3). Hierarchical Inheritance:

CLASS A

CLASS A

CLASS C

CLASS B

4). Multiple Inheritance:

CLASS B

CLASS A

CLASS A

CLASS C

Note: JAVA does not support multiple inheritance, This mean a class cannot extends more than one class. To Achieve Multiple Inheritance you have to use Interface.

Super Keyword:

The Keyword **super** provides a reference to the current object as an instance of its super class. In method invocation with **super**, the method from the super class is simply invoked regardless of the actual type of the object or whether the current class overrides the method. It is typically used to invoke methods that are overridden and to access members that are hidden in the subclass.

Unlike the keyword, the **super** keyword cannot be used as an ordinary reference.

Example.2:

class One {

int x;

}

class Two extends One {

int x;

public void setx(int a, int b) {

x=a;

super.x=b;

}

public void showx() {  
 System.out.println(x+”\t”+super.x);

}

}

class useSuper {

public static void main(String[] args){

Two obj=new Two();

obj.setx(10, 20);

obj.showx();  
 }

}

Output : x 20

Polymorphism:

Polymorphism means one name many forms. Any things that exists in more than one form is known as a polymorphism. Polymorphism applies only to non-private instance method it does not apply to variable or static method. Inherited method in the case of polymorphism cannot private. There are two types of polymorphism one is compile time polymorphism and the other is run time polymorphism.

Compile time polymorphism is function and operators overloading. Runtime polymorphism is done using Inheritance

Overriding and Overloading:

Method overriding required the same method signature and the same return type. Only non-final instance methods in the superclass that are directly accessible from the subclass are eligible for overriding.

Overloading occurs when the method names are the same, but the parameters lists differ. Therefore to overloads methods, the parameters must differ in type, order or number. As the return type is not a part of the signature , having different return types is not enough to overload methods.

* Method overloading is possible in same class and in case of inheritance.
* Method overriding is possible in only case of inheritance .
* When sub and super class both have method with the same name, same argument and same return type it is called method overriding.
* If return type is not same in the case of method overriding compile time error will be occur.
* In the case of method overriding, the overridden method of sub class can be less restricted, but it cannot be more restricted.
* Ordinary method can inherit but the constructer of a class never inherited regardless their attribute.
* Static method cannot overridden.

class One {

private int x;

public void set( int a){

x=a;

}

public void show() {

System.out.println(“X=”+x);

}

}

class Two extends One {

int x;

public void set(int a, int b){

{

set(a);

x=a;

}

public void show(){

super.show();

System.out.println(“Y=”+x);

}

}

class Use{

public static void main(String[] args){

obj.set(10, 20);

obj.show();

}

}

Output :

X = 10

Y = 10;

Abstraction:

Hide internal implementation details and just highlight the service what we are offering is called Abstraction.

Example- By Bank ATM machine, Bank people Highlight set of Services what they offering without highlighting internal implementation. This concept is Abstraction

By using Interface and Abstract class we can achieve Abstraction. Main advantages of abstraction are:

We can achieve security as no one is allowed to know our internal implementation.

Without effecting outside person we can change internal implementation. Hence will become very easy.

Main disadvantage of Encapsulation is it increase length of code and slow down execution.

Tightly Encapsulated class:

A class is declared as tightly encapsulated if all the data members declared as private.

Whether class contain getter and setter methods are not and

Abstract Class:

An abstract class is a class that cannot be Instantiated. Abstract class is defined just to achieve polymorphism. It contains method that has no implementation. The methods that have no implementation in the abstract class are called abstract method. An abstract method is implementation in the sub class instead of being implemented in the abstract class. An abstract class is always inherited by other class.

abstract class Alphabet {

abstract public void small();

abstract public void capital();

}

* If a method that has no definition than it is called an abstract method and it must be declared using the keyword abstract otherwise compile time Error generated.
* You can declare a reference variable of an abstract class but cannot create object of abstract class because abstract class cannot instantiated.

Alphabet a;

A=new Alphabet(); // compile Time Error

* If a class Inherits an abstract class then it must be defined all abstract method in the sub class. If any of the abstract method remains undefined in the sub class must be declared abstract otherwise compile Time Error. And if sub class declared abstract we will not be able to create object of sub class
* Abstract method cannot be private, since private method never inherited.
* Abstract class cannot be final, because final methods cannot override.
* Abstract class cannot be final, because final class never Inherited.
* Abstract class can have instance variable , methods and constructor and they will be utilizing when we create an object of sub class.

abstract class Alphabet {

private Boolean b;

abstract public void fruit ();

abstract public void toy ();

}

class Small extends Alphabet {

public void fruit () {

System.out.println(“Orange”);

}

public void toy () {

System.out.println (“Pineapple”);

}

}

class Capital{

public static void main(String[] args) {

Alphabet a;

a=new Small();

a.fruit();

a.toy();

}

}

Output :

Orange

Pineapple

**Encapsulation**:

Encapsulating data and methods into a single model is called “Encapsulation”. If any JAVA class follows Data Hiding and Abstraction such type of class is said to encapsulate class.

Hide data behind method is the control concept of Encapsulation

Advantages of Encapsulation are:

* We can achieve security.
* Enhancement will become very easy.
* Improve modularity of the application.

Example:

class Account {

private double balance;

public double getBalance() {

//validate user

return balance;

}

public void setBalance() {

//validate user

this.balance = balance;

}

}

Output : getBalance() method will return the balance variable value and setBalance() method will set value to the balance variable;

**Tightly** **Encapsulated** **Class**: a class is said to be Highly Encapsulated if every data member declared as private. Whether the class contain getter and setter method are not and whether those methods declared as public or not these are not required to check.

Ex:- class A {

private int balance;

public int getBalance() {

return balance;

}

}

**Interface**:

An Interface in java programming language is an abstract type that is used to specify an interface that class must implement. Interface is declared using the interface keyword, and may only contain method signature and declarations (variable declaration that are declared to be both static and final).

Advantages:-

* Easy to achieve security because we are not highlighting inner implementation.
* Enhancement will become very easy, because without effecting outsider we can change internal implementation.
* Two different systems’ can communicate via Interface.

(Java application can talk with mainframe system through interface)

We can declared an interface by using interface keyword and implement by using Implements keyword.

Example:

interface Inter {

void water(); // by default public abstract void water();

void juice();

}

Abstract class SelectDrink implements Inter {

Public void water() {

System.out.println(“This is water.”);

}

}

Output : This is Water.

If a class implements an Interface compulsory you have to provide implementation for every method of that interface otherwise you have to declared class as abstract.

Whenever you are implementing an interface method compulsory it should be declared as public otherwise compile Time error generated.

A class can extends only one class at a time

A class can implements any number of interface at a time.

A class can extends class and implement any number of interface.

**Static and Non-static method:-**

**Static** method belong to the class not to the object. static method can access only static data, it cannot access **non-static** data. A static method can call only other static method and cannot call other non-static method.  
static method can be accessed directly by the class name and doesn't need class object static method can't refer to **this** and **super** keywords.

<classname><method name>

exmple:-

class StaticDemo

{

public static void copyArg(String str1, String str2)

{

//copies argument 2 to arg1

str2 = str1;

System.out.println("First String arg is: "+str1);

System.out.println("Second String arg is: "+str2);

}

public static void main(String agrs[])

{

//StaticDemo.copyArg("XYZ", "ABC");

copyArg("XYZ", "ABC");

}

}

Output :

First String arg is: XYZ

Second String arg is: XYZ

**non static method:-**

non-static method never be preceded by static keyword, and memory allocated multiple time whenever method is calling. It is specific to an object so that these are also known as instance method. If any method wants to be execute multiple time that can be declare as non static. These methods always access with object reference

Syntax:

Objref.methodname();

exmple:-

class A {

void fun1() {

System.out.println("Hello I am Non-Static");

}

static void fun2() {

System.out.println("Hello I am Static");

}

}

class Person {

public static void main(String args[]) {

A a=new A();

a.fun1(); // non static method

A.fun2(); // static method

}

}

Output :

Hello I am Non-Static.

Hello I am Static