OOPS Concepts:-

1. Encapsulation :- Wrapping data members & its functions in to a single unit.

Creating a Class with Private variables & these Private Variables can be Accessed using Public setters & getters methods

Setter methods are used to set values for private variables(Class Level) of a class.

Seter method Return Type will be always void.

Getter Method return type will be always the return type of its variable

1. Abstraction :-

Hiding the Implementation of the functionality . It Tells what is does or how it functions or how it is works & but it dosent tell how it was Implemented.

Ex;- clicking on login button(we don’t know how it is validating the Authentication)

Ex:- clicking on Submit button(we don’t know how all the data is saved & submitted in the backend)

This is Achieved by Interfaces & Abstract Classes.

100% Abstraction can be Achieved with Interface

100% Abstraction cannot be achieved with Abstract Class because it might/can contain defined methods because of which 100% Abstraction is not possible

|  |  |
| --- | --- |
| Abstract Class | Interface |
| It Contains Abstract Methods & Defined Methods & Constants | It contains only Abstract Methods & Constants(public static final) |
| Abstract Keyword has to applied at class level & Method Level for Abstract methods | Abstract Keyword is optional at Interface level & method level because JVM will anyways provide . |
| Abstract class can contain Constructor but it will be used by its subclasses | Interface cannot contain Constructor because it will give compilation error if we add Constructor |
| Class &Abstract Class will not participate in Multiple Inheritance | Interface can be used in multiple Inheritance. |
|  |  |
|  |  |
|  |  |

Similarity

object Cannot be created for Both Abstract class & Interface.

Abstract class objects can be create by using its sub classes.

Interface methods are accessed by using its sub class objects.

Inheritance :- Inheritang the Properties and behaviours(variables and methods) from Parent to its children

1. InHeritance :- Inheriting the variables& methods from SuperClass to its SubClasses(Parent to Child)

Private methods & private variables cannot be accessed/Visible in SubClasses.

Inheritance can be achieved through **extends** keyword between classes(A,B)& Interfaces(I1,I2)

Ex:- **Class A extends B**

Interface I1 extends I2

Class A can Implement Interface I1

Ex:- Class A Implements I1

Interface I1 cannot Implement or Extend Class A

Interface I1 implements class A (Not Possible)

Single Inheritance:-

We can Create objects of SuperClass with SuperClass Reference so that we can access all the Public methods in Super Classs.

We can Create objects of SubClass with SubClass Reference so that we can access all the Public methods in Sub Classs & SuperClass

If we Create objects of SubClass with SuperClass Reference then we can access all the Public methods of SuperClass But we will not be able to access methods from Sub Class.

**Multiple inheritance**  in the below diagram A and B are interfaces C is sub class of A and B.

**Note**: If a class (C) implements an interface(I) it has to override all the methods of interface.

It gives compilation error if we donot override any method of the interface.

If an abstract class (C) implements an interface(I) overriding the methods of interface is optional.

**Hybrid inheritance** is the combination of more than one inheritences

In Hybrid Inheritance, A,B,C should be Interfaces because if A is Class B & C cannot implement class A.

If a Class A extends ClassB and implements Interfaces I1,I2 we should write the syntax as below:-

Class A extends B implements I1,I2

Class A implements I1 extends B ------not possible throws compilation error



4.Polymorphism :-

Same Action Can be performed in multiple ways

We can achieve this feature in java using methods

1. **Compile time Polymorphism** :- This can be Achieved by Method Over Loading Concept.

Method Over Loading Concept :- It is performed in a Class

Methods Name are Same but Signature should be different.

Method Signature:- Number of arguments, data types of Arguments, order of the Arguments

If two methods have same name with same signature, Access Specifiers & return types doesn’t make a difference.

1. **Run time Polymorphism**:- This can be Achieved by Method Over Riding Concept.

Method Over Riding:- It can be Achieved using Inheritance concept

Methods Name are Same & Signature are also Same . Along with these method return types & Access Specifiers should be also the same.

**Constructor**:-

Constructor is used for Allocating Memory for the Class.

Rules for Constructor:-

1. Constructor name should be exactly same as ClassName(CaseSensitive)
2. Constructor will not have return type(like void, int, String)

Default Constructor( 0 Arguments Constructor) is optional which is provided by JVM

Class can Have Default Constructor & OverLoaded Constructors.

For OverLoaded Constructor, Signature should be different.(Number of arguments, data types of Arguments, order of the Arguments)

In a class if we override overloaded constructor(1 or 2 arg constructor) and if we try to call default constructor(creating object with default constructor) it will give compilation error because jvm will not provide default constructor in this case

We can add return statement inside a constructor it does not give compile time or runtime error

Class A

A(){

………………….

return;

}

*Static* :-

Static is a Keyword which is used at Variable level & method level but not at the Class Level. Which means class can contain static variables ,Static methods & static blocks also.

JVM Allocates memory for static variables Static methods & static blocks at compile time so that we can access these without creating objects(using Class name, we can access these.)

If we try to access these with objects name it does not give any compile time or run time error. It just gives Warning.

Static methods cannot be Overridden in its subclasses because the methods are at class level but they can be accessed in Subclasses.

A Class can contain instance variables, static variables ,constants, instance methods, static methods, constructors, static blocks & instance blocks.

If a Class contains all the above details then it will be executed in the following order:-

1. All the Static Blocks in the order
2. When we create a object of the class all the instance blocks in the order are executed first
3. If constructor is overridden in class that will be executed.
4. Methods will be executed if they are called.

Abstract class can contain static method , abstract method ,define method (instance method).

If we give static for an abstract method it will throw compilation error. Because abstract method memory is allocated at runtime but static method memory loads at compile time.

**Final** is a Keyword which is used at Variable level & method level & Class Level . In a Class if a variable is declared as Final the value will not be changed once it is assigned.

We can not assign final variable multiple times as below

Class A {

Final String s = null;//the value is assigned as null,can not be changed any where.

Final int I;//this is compilation error

Final Methods cannot be overridden in its subclasses but they can be accessed in Subclasses.

Final Classes cannot participate in inheritance that means we cannot extend final class.

**this :** this isa keyword which is used to refer current class object. This is used at variable level and method level and **this()** will be treated as default constructor.

Generally **this()** is used as first statement in a constructor otherwise it will give compile time error.

**super** super is a keyword which is used to refer super class variable or method in a

Generally **super()** is used as first statement in a constructor otherwise it will give compile time error

**Note** :constructor we can add either this() or super() but not both.

**Access specifiers:** we have4 types of access specifiers in java which is used at variable level method level ,constructor level and class level

**Private**: access only inside the class

**Defau**lt: if we donot give any access specifier it will be treated as default. This is package level specifier that means we can access any variable or method of a class p1.A(p1 is package ) in any class inside the package p1

**Protected**: we can access a variable of a class p1.A(p1 is package ) in any of the class inside the package p1 and any it sub packages class(p1.p2.B)

**Public:** we can access public variables or methods in any other class of any package