**Object-oriented programming system** is a programming based on the concept of "objects" that contain **data** and **methods**.

The primary purpose of oops is to increase the flexibility and maintainability of programs

The **OOPS** brings together data and its behaviour (methods) in a single location (i.e objects) makes it easier to understand how program works.

**Object:** is a bundle of data and its behaviour(methods), object is an entity.

Object has two characteristics. They have state and behaviour.

**Characteristics of Objects**

1. Abstraction

2. Encapsulation

3. Message passing

1. **Abstraction:** is a process where we show only relevant data and hide unnecessary details of an object from the user.

2. **Encapsulation:** is simply binding objects state (fields) and behaviour(methods) together.

3. **Message Passing:** A single object itself may not be useful. An application contains many objects.

Object interact with another object by invoking methods on that object. it is referred as Method Invocation.

**Class:** can considered as blueprint using which we can create as many as objects

**Constructor:** is a block of code that initializes the newly created Object.A constructor resembles an instance method in java but it's not a method.

Looks like method but in fact it not a method. it's name is same as class name and it does not have return value.

for example to create object -- > ClassName object= new ClassName();

we are calling a default constructor to create a new object.

**Types of Constructors**

1.**Default** - if we do not implement any constructor in our class. Java compiler inserts a default constructor in the code on our behalf.

2. **No-arg** - constructor with no arguments is known as no-arg constructor. the signature is same as default consturctor, however can have any code in the constructor body.

3. **Parameterized** - constructor with arguments ( parameters ).

Constructor Overloading : a class can have more than one constructor with different paramters.

Constructor overloading is possible but overriding is not possible.

The Purpose of constructor is to initialize the object of a class while the purpose of a method is to perform a task by executing java code.

Constructor cannot be abstract, final, static and synchronised while methods can be.

**Super** : whenever a child class constructor get invoked it implicitly invokes the constructor of a parent class. we also say compile inserts a super() statement at the beginning of child class constructor.

**this()** and super() should be the first statement in the constructor code.

**Object-oriented programming features :**

1. **Abstraction** : is a process where we show only relevant data and hide unnecessary details of an object from the user.

2. **Encapsulation** : is simply binding objects state ( fields) and behaviour(methods) together.

3. **Inheritance** : the process by which one class acquires the properties and functionalities of another class.

4. **Polymorphism** : allows us to perform a single action in different ways.

Types of Polymorphism

1. Static Polymorphism : Polymorphism which is resolved during compile time is known as static Polymorphism

**Method Overloading :** is considered as static polymorphism.

**Method Overloading** : is a feature that allows a class to have more than one method have same name, it their arguments lists are different.

**Type Promotion** : when a data type of smaller size is promoted to the data type of bigger size than this is called Type Promotion.

2. **Dynamic Polymorphism** : is a process in which a call to an overridden method is resolved at runtime it is decided by JVM. Dynamic Polymorphism is also referred as Dynamic method dispatch, runtime polymorphism.

Method Overriding : When the parent class reference points to the child class object then the call to the overridden method is determined at runtime, because during method call which method ( parent class or child class) is to be executed is determined by the type of object.

Method Overriding : Declaring a method in sub class which is already present in parent class is known as method overriding.

**Abstract Class and methods in OOPs**

A class that is defined using abstract keyword is known as abstract class, it can have methods (**regular methods without body**) as well as concrete methods (**regular method with body**)

**Abstract methods** : A method declared but not defined. Only method signature no body. Declared using the abstract keyword.

Abstract Class : outlines the methods but no necessarily implements all the methods.

Why we need abstract class : all the child classes will and should override this method then there is no point to implement this method in parent class. thus making method abstract would be good choice as by making this method abstract we force all the sub classes to implement its own implementation of this method.

Abstract class cannot be instantiated which means we cannot create the object of abstract class because these classes are incomplete, and they have abstract methods that have no body

To use this class we need to create another class that extends this abstract class provides the implementation of abstract methods, then we can use the object of the child class to call non abstract parent class methods as well as implemented methods (those that were abstract in parent but implemented in child class.

Example :

public class A

{

public abstract myMethod();

}

public class B extends A

{

public void myMethod()

{

System.out.println ("child class method");

}

B obj = new A();

obj.myMethod();

}

**Concrete Class** : A class which is not abstract is referred as concrete class.

**Interface** : is blue print of a class, which can be declared by using interface keyword.

Interface contains only constants and abstract methods (**methods with only signatures no body**)

Like abstract class Interfaces cannot be instantiated, they can only be implemented by classes or extending by other interface

Interfaces is a common way to achieve full abstraction.

**Differences between abstract class and interface**

Abstract class can extend only one class or one abstract class at a time. Interface can extend any number of interfaces at a time.

Abstract class can be extended where as Interfaces has to implemented instead of extend.

Abstract class can have both abstract methods and concrete methods where as Interfaces can only have abstract methods not concrete methods.