**Object Oriented Programing Concepts (OOPs)**

OOPs is the methodology to design a program using classes and objects.

OOP’s is the methodology to way of programming.

There are four major pillars are there.

* **Inheritance**
* **Polymorphism**
* **Abstraction**
* **Encapsulation**

Apart from that

* **Coupling**
* **Cohesion**
* **Association**
* **Aggregation**
* **Composition**

**Object**

Object is an entity that has some state and behaviour.

Object are instance of classes.

**Class**

Class is Blueprint or Template to define the Objects.

It has Fields and Method.

**Inheritance (IS-A)**

Inheritance means one Object can acquires all the properties and behaviours of another Object.

It provide code reusability.

With the help of inheritance we can achieve run time polymorphism using method overriding.

It represent IS-A relationship.

We can use the extends keyword for making inheritance.

For 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);  
 }  
}

In this Example **Programmer IS-A Employee** relationship.

Types of Inheritance is Single, Multilevel, Hierarchical.

Multiple inheritance is not support in Java.

**Why Multiple Inheritance is not support in Java?**

To reduce Complexity and Ambiguity.

**Inheritance (IS-A)**

**Aggregation (HAS-A)**

**Aggregation (HAS-A)**

It means one class Has a property of second class.

In simple way Person class HAS-A Address class.

**Polymorphism**

Polymorphism means one task can perform in different ways.

We can achieve Polymorphism by using term Method Overloading and Method Overriding.

**Upcasting – A a = new B();**

If the reference variable of parent class is refer to the object of child class is known as Upcasting.

**Downcasting**

**A a = new B();**

B b = (B) a;

It means we convert superclass type back to the subclass type.

If we have Upcast object refence the we can convert this to subclass where we will create reference of subclass and typecast to upcasted refence.

**Method Overloading – Compile Time Polymorphism**

**Method Overriding – Runtime Polymorphism**

**Method Overloading**

One class having more than one methods having same name but different argument is called Method Overloading

No. of Argument id different

Argument type is different

Arguments order is different

If No. of Arguments is same and also type then we can’t Change Return type.

**Method Overriding**

Where sub class or childe class having same method of super class or parent class is called Method Overriding.

Method must have same name as in parent class.

Method must have same parameter as in parent class.

There must be IS-A relationship.

**Covariant Return Type**

After Java 5 it is possible to change return type of method in method overriding but the return type has Non primitive and subclass of the parent class.

**Abstraction**

In simple way Abstraction means Hiding internal details and showing only functionality.

We use Abstraction Class and Interface to achieve Abstraction.

Using Abstract keyword on class is known as Abstract Class.

Abstract class have Abstract method and Non-Abstract method.

Abstract Class – 0 to 100

Interface – 100%

**Interface**

Interface is like blueprint of class.

Interface has Static constant and abstract methods.

Same as Abstraction Hiding internal details and showing only functionality.

Interface used to achieve Abstraction.

In all method is abstract by default means no body but Since Java 8 we have default method and Static methods.

And also since Java 9 Private Method is in Interface.

Interface is indirectly achieve Multiple Inheritance.

**Marker Interface**

An interface which has no member is known as Marker Interface for example Serializable, Cloneable etc.

Marker Interface is use to provide some essential information to JVM so that JVM will perform some useful operation.

**Encapsulation**

Encapsulation means the process of biding or wrapping all the code and data in single unit.

We can create fully Encapsulated class by making all data members private in class.

We can use the getter and setter for get the data and set the data.

**Association**

Association represent the relationship between the objects.

Means one object can associate with other object.

**Aggregation** and **Composition** is the way of archive Association

In Aggregation represent week relation in the objects and other hand Composition represent strong relation in between objects.

**Constructor**

Constructor is a block of code similar to method.

It has same name as class name.

Constructor is create at the time creation class instance and memory for object is allocated in memory.

Every time we create an instance of class then it will automatically invoke or call constructor.

If constructor is not present then java compiler is provide default constructor.

Constructor cannot be Static, Abstract, Final and Synchronized.

Purpose of default constructor is to provide the default value to the object like 0, null etc. Deepening on its type.

Two type of Constructors

Default and Parameterized

**Constructor Overloading**

Constructor Overloading is just like method overloading without return type.

**Method hiding** can be defined as, "**if a subclass defines a static method with the same signature as a static method in the super class, in such a case, the method in the subclass hides the one in the superclass**."

**Static Keyword**

Static keyword is belong to method area where method or variables is belong to class itself rather than instance of that class.

When member (variable or method) is declared as Static then that there is only one instance is shared by all instance of the class.

Beast Advantage of Static is make the program Memory Efficient.

Static can be :

* **Static Variable**
* **Static Method**
* **Static Block**
* **Nested Static Class**

**Static Variable**

It also called class Variable means it belong to class level.

It means the variable is associated with the class itself rather than instance of that class.

There will be only one copy of that variable shared by all other instance of the class.

Static variable initialize only once when the execution of program.

class MyClass  
{  
 static int *i* = 10;  
 public MyClass()  
 {  
 System.*out*.println(*i* = *i* + 10); // 20 30 40  
 } // if not static int i then o/p :- 20 20 20  
}  
public class StaticKeyword {  
 public static void main(String[] args) {  
 MyClass m1 = new MyClass();  
 MyClass m2 = new MyClass();  
 MyClass m3 = new MyClass();  
 }  
}

**Static Method**

Static method belong to class it will call by class name rather than instance of that class.

Static method can access only static variables.

This and Super keyword is not used in this context.

class MyClassStaticMethod  
{  
 static int *i* = 10;  
// int i = 10;  
 public static void myStaticMethod()  
 {  
 System.*out*.println(*i* = *i* + 10);  
 }  
}  
public class StaticKeyword {  
 public static void main(String[] args) {  
  
 MyClassStaticMethod.*myStaticMethod*(); // O/P :- 20  
 MyClassStaticMethod.*myStaticMethod*(); // O/P :- 30  
 MyClassStaticMethod.*myStaticMethod*(); // O/P :- 40  
  
 MyClassStaticMethod staticInstance = new MyClassStaticMethod();  
// staticInstance.myStaticMethod(); // we cant call this by using instance  
 }  
}

**Static Block**

Static Block is execute before main method at the time classloading.

public class StaticKeyword {  
 static {  
 System.*out*.println("Static Block is called");  
 }  
 public static void main(String[] args) {

System.*out*.println("Main method is called");  
 }  
}

O/P 🡪

Static Block is called

Main method is called

**Nested Static Class**

In Java, a class can be defined within another class. When a nested class is declared as static, it is called a static nested class. Static nested classes are associated with the outer class rather than with instances of the outer class.

class OuterClass {  
 static class StaticNestedClass {  
 // static nested class  
 }  
}

It's important to note that since static members belong to the class itself, they can be accessed using the class name directly, without needing to create an instance of the class. However, they cannot access non-static members of the class directly because static members are not associated with any particular instance.

**This Keyword**

This keyword can be use for refer the current class instance variable.

It also refers the current class constructor and method.

**Super Keyword**

Super keyword can be use to refer super class method and contractors and invoke parent class instance variable.

**Final Keyword**

Final keyword is used to restrict the user.

Final Variable – If we declare variable as final then we cannot change the value of that variable.

Final Method – We cannot override it.

Final Class – We cannot extend it.

**Static Binding (also known as Early Binding).** – Normal we create class object which is not extend any class.

**Dynamic Binding (also known as Late Binding).** – Like Runtime polymorphism where we create reference of parent class and instance of chilled class.

**How to create Immutable Class in Java?**

To create an immutable object in Java, you should follow these steps:

* Make class final.
* Declare all fields of the class as final and private.
* Initialize the fields in the constructor.
* Do not provide setter methods for changing the state of the object.

**Object Class**

In java Object class is parent class of all the classes in Java by default.

class A  
{  
 public void test()  
 {  
 System.*out*.println("Class A");  
 }  
}  
class B extends A  
{  
 @Override  
 public void test()  
 {  
 System.*out*.println("Class B");  
 }  
}  
public class ObjectClassExample {  
 public static void main(String[] args) {  
 Object A = new B();  
 System.*out*.println(A.getClass()); // class org.example.B  
 }  
}

**Object Cloning**

Object cloning mean the creating of the exact copy of the existing object.

clone() method is use to clone an object.

java.lang.Clonable interface must be implemented by the class which we want to create clone object.

If we do not implement cleanable interface then it will throw CloneNotSupportException.

Main use of cloning is if there is some lengthy or time consuming task is present so it will reduce this and clone it for us.

When we clone the object then it will create copy of that object.

There is two type which is Shallow copy and Deep copy.

**Shallow Copy**

Simply means it will create copy of that object and also copy all reference of object which is in object.

If we make some changes in copied object then it will reflect main object.

**Deep Copy**

In this copied object will create new objects and if we make changes in copied object then it will not reflect in main object.

**Wrapper Class**

Wrapper class is provide a way to use primitive data types as objects.

Each primitive data type has a corresponding wrapper class.

Wrapper class provide a method to convert primitive data type to object (Boxing) and Object into primitive (unboxing).

We are use wrapper class in situation where we need Object such as Collection like ArrayList or working with generic which does not support directly primitive data type.

**Why use instance initializer block?**

**class** Bike7{

**int** speed;

    Bike7(){System.out.println("speed is "+speed);}

**{speed=100;}  // instance initializer block**

**public** **static** **void** main(String args[]){

    Bike7 b1=**new** Bike7();

    Bike7 b2=**new** Bike7();

    }      `

}

**Anonymous Object**

It mean Nameless object which will No reference of Object.

new objectA.callMethode(“Anonymous”);