**OOP Interview   
Preparations   
Question**

# 

### Object Oriented Programming (OOP)

* + OOP is a programming language model organized around object rather than actions (logic and functions).
  + In other words, OOP mainly focuses on the objects that are required to be manipulated instead of logic. This approach is ideal for the programs large and complex codes and needs to be actively updated or maintained.;
  + It makes development and maintenance easier - It provides data hiding - It provides ability to simulate real-world.

**OOP language follow 4 principles**:

* + - **Encapsulation** : We can hide direct access to data by using private key and we can access private data by using getter and setter method.
    - **Abstraction** : It is a process of hiding implementation details and showing only functionality to the user. Abstraction lets you focus on what the object does instead of how it does it.
    - **Inheritance** : It is used to define the relationship between two classes. When a child class acquires all properties and behaviors of parent class known as inheritance. Child class can reuse all the codes written in parent class. It provides the code reusability.
    - **Polymorphism** : It is an ability of object to behave in multiple form. The most common use of polymorphism in Java is when a parent class reference type of variable is used to refer to a child class object.

*Example*

WebDriver driver = new ChromeDriver();

We use method overloading and overriding to achieve Polymorphism.

### What is encapsulation and how did you use it?

* + Data hiding by making variables private and providing public getter and setter methods.
  + In my project I created multiple POJO/BEAN classes in order to manage test data and actual data.
* EX: I take JSON from API response and convert to object of my POJO class all variables are private with getters and setter.

### What is the concept of Abstraction?

* + In OOP, abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user.
  + In other words, the user will have the information on what the object does instead of how it does it.
  + In Java, abstraction is achieved using Abstract classes and interfaces.
  + For example: when you log in to your bank account online, you enter your user-id and password and press the login. What happens then, how the input data sent to the server, how it gets verified are all abstracted away from you.

### Difference between Abstraction and Encapsulation?

* + **Abstraction** lets you focus on **what** the object does instead of **how** it does it.
    - **Encapsulation** means hiding the internal details of how the object does something.
  + **Abstraction** is used for hiding the **unwanted** data and giving relevant data.
    - **Encapsulation** means hiding the code and data, and to protect the data from outside.
  + **Abstraction** can achieved by using Abstract class and Interfaces
    - **Encapsulation** can achieved by using “private” keyword.

### Difference between Abstract Class and Interface?

* + Main difference is methods of a Java interface are implicitly abstract and cannot have implementations. A Java abstract class can have instance methods that implements a default behavior.
  + A class that is declared with abstract keyword, is known as abstract class. It can have abstract and non--abstract methods.
  + An Interface is a blueprint of a class. It is a template and it is declared with interface keyword. It can have abstract methods, default methods, static methods and public final static variables
  + When we want to use Abstract class, we use “**extend**” keyword. When we want to use Interface, we use “implement” keyword.
  + Abstract class and interface both are used to achieve abstraction Both cannot be instantiated; we cannot create an object.

### What is Polymorphism?

* + Polymorphism is a very important concept in OOP because;
    - it enables to change the behavior of the applications in the run time based on the object on which the invocation happens.
    - by Polymorphism; one object can have different forms
  + Two types  **Compile Time** which is Static and **Run Time** Polymorphism which is related with child and parent class.
  + Polymorphism is implemented using the concept of Method overloading and method overriding. This can only happen when the classes are under the parent and child relationship using inheritance.

### What is Inheritance?

* + Inheritance represents the **IS-A** relationship which is also known as a parent-child relationship.
  + It is the mechanism in java by which one class is allowed 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 the parent class.
  + Moreover, you can add new methods and fields in your current class also.
  + Code reuse is the most important benefit of inheritance because subclasses inherits the variables and methods of superclass.

### Important terminology in Inheritance?

* + **Class:** the group of objects which have common properties. It is a template or blueprint from which objects are created.
  + **SuperClass**: the class being inherited from (or a base class or a parent class).
  + **SubClass**: the class that inherits from another class(or a derived class, extended class, or child class).
    - The subclass can add its own fields and methods in addition to the superclass fields and methods.
  + **Reusability**: 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 the previous class.

### Difference between Polymorphism and Inheritance

* + Like in real world, Inheritance is used to define the relationship between two classes. It is similar to Father-Son relationship. In Java, we have Parent class (also known as super class) and child class (also known as subclass). Similar to the real-world, Child inherits Parents qualities, methods and codes.
    - A child class can reuse all the codes written in Parent class and only write code for behavior which is different than the Parent.
    - Inheritance is actually meant for code reuse.
  + On the other hand, Polymorphism is an ability of object to behave in multiple form.
    - It is classified as overloading and overriding.
  + By the way, they are actually related to each other, because its inheritance which makes Polymorphism possible, without any relationship between two class. It is not possible to write polymorphic code.
    - Dynamic Polymorphism  Overriding
    - Static Polymorphism  Overloading

### Difference between method Overloading and method Overriding?

* + First and most important difference between overloading and overriding is that,
    - in case of overloading , method name must be the same, but the parameters must be different;
    - in case of overriding , method name and parameters must be same
  + Second major difference between method overloading and overriding is that;
    - We can overload method in the same class but method overriding occurs in two classes that have inheritance relationship.
  + We cannot override static, final and private method in Java, but we can overload static, final and private method in Java.
  + In method overloading , return type can be same or different. In method overriding , return type must be same or covariant type.

### Difference between Object and Class?

* + Class is a blueprint or template which you can create as many objects as you like Object is a member or instance of a class
  + Class is declared using class keyword, Object is created through new keyword mainly.

A class is a template for objects. A class defines object properties including a valid range of values, and a default value. A class also describes object behavior. An object is a member or an "instance" of a class and has states and behaviors in which all of its properties have values that you either explicitly define or that are defined by default settings.

Class - A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support. If we compare them there are many differences but let me tell you some of them which are important to know;

* + There are many ways to create object in java such as new keyword, newInstance() method, clone() method, factory method and deserialization. There is only one way to define class in java using class keyword.
  + Object is created many times as per requirement. Class is declared once.
  + Object is an instance of a class. Class is a blueprint or template from which objects are created.
  + Object is a physical entity. Class is a logical entity.

**For Example:**

Class: Human Object: Man, Woman

Class: Fruit Object: Apple, Banana, Mango, Guava

Class: Mobile phone Object: iPhone, Samsung, Moto

Class: Food Object: Pizza, Burger, Samosa

### How do you use an abstract class in your project give me an example?

* + These concepts are commonly used in framework development. Abstract class is used in defining a common super class while writing Page Object Model layer of the framework. We usually create an abstract class named BasePage to have all common members for every page written in this class example **getPageTitle().**
  + Then each Page class (HomePage, LoginPage, DashboardPage etc.) inherit from BasePage. Sometimes one may need to change the behavior of methods implemented in superclass. So, subclass has freedom to override that method where we use polymorphism. This is how we use Abstract class in real projects.

## ABSTRACTION

* Abstraction means we **focus on the essential** qualities of something rather than one specific example.
  + Focus on the essential (we know phone need to call(), but we don't know how do iphone, samsung call())
  + Ignore the irrelevant
  + Ignore the unimportant
* In Java, abstraction is achieved **by interfaces and abstract classes.** We can achieve 100% abstraction using interfaces.

## ABSTRACT CLASSES

|  |  |
| --- | --- |
| **CREATING ABSTRACT CLASSES**   * **abstract** keyword is used to create abstract class. * An abstract class **can not** be instantiated. * Goal is to provide reusable variables and methods to sub classes.   **ABSTRACT CLASS RULES REVIEW**   * + It cannot be instantiated directly.   + It may be defined with any number, including zero, of abstract and non­abstract methods.   + It may not be marked as private or final.   + An abstract class that extends another abstract class inherits all of its abstract methods as its own abstract methods.   + The first concrete class that extends an abstract class must provide an implementation for all of the inherited abstract methods | **public abstract class Student{**  **}** |
| **CREATING ABSTRACT METHODS**   * **abstract** keyword is used to create abstract method. * Abstract method **does not have body**, only have signature.   **ABSTRACT METHOD RULES REVIEW**   * + It may only be defined in abstract classes.   + It may not be declared private or final.   + It must not provide a method body / implementation in the abstract class for which it is declared.   + Implementing an abstract method in a subclass follows the same rules for overriding a method. | **public abstract class Student{**  **public abstract void attendClass();**  **}** |
| **CREATING CONCRETE CLASS**   * A subclass of abstract class is called concrete class * A first concrete class must implement all inherited abstract | **public abstract class Student{**  **public abstract void attendClass();**  **}** |
| methods | **public class LocalStudent extends Student{ @Override** |
|  | **public void attendClass(){** |
|  | **System.out.println("attending in person");** |
|  | **}** |
|  | **}** |
| **EXTENDING ANOTHER ABSTRACT CLASS**   * An abstract class can extend another abstract class. If so it is optional to implement abstract methods from abstract super class. * A first concrete class must implement all inherited abstract methods. | **public abstract class LocalStudent extends Student{**  **public void attendClass();**  **}** |

1. **INTERFACE**
   * Contract between a class and outside world
   * Provide set of abstract methods
   * A class implements an interface
   * The class provides the behaviors included in the interface
   * Interface can also store constants

|  |  |
| --- | --- |
| **CREATING AN INTERFACE** | **public interface Teachable{**  **public static final boolean STUDY\_HARD = true; public abstract void canLean();**  **public abstract void doHomework();**  **}** |
| **IMPLEMENTING AN INTERFACE**   * A class can implements more than one interface.   **public class Student implements Teachable, Dreamer{**  ***//implementation code***  **}**   * If a class both extend a class and implement an interface, extends should come first then implements keyword   **public class Student extends Person implements Teachable, Dreamer{**  ***//implementation code***  **}** | **public interface Teachable{**  **public static final boolean STUDY\_HARD = true;**  **public abstract void canLean(); public abstract void doHomework();**  **}**  **public class Student implements Teachable{**  **@Override**  **public void canLean(){**  ***//code***  **}**  **@Override**  **public void doHomework(){**  ***//code***  **}**  **}** |
| **INTERFACE RULES**   * It is a abstract type and can not be instantiated. → * An interface is abstract by default and may not be → marked as final * All fields in interface is automatically public static final even no declared such → | **public interface Teachable{}**  **Teachable t = new Teachable(); *//DOES NOT COMPILE***  **public final interface Teachable{} *//DOES NOT COMPILE***  **public final interface Teachable{ boolean STUDY\_HARD = true; void canLean();**  **public abstract void doHomework();**  **}** |
| **EXTENDING ANOTHER INTERFACE**   * An interface can extends another interface using   **extends** keyword to share functionality.   * Unlike class, an interface can **extends multiple interfaces.** * First concrete class has to implement all the abstract methods from both interface. | **public interface Teachable{ void canLearn();**  **}**  **public interface Mentorship extends Teachable,Bright{ void mentor();**  **}** |