**OOPs concepts with respect to Framework**

**ENCAPSULATION :**

Binding properties and behaviour of an object in a programming structure or class is called as Encapsulation.

**Rules to achieve fully encapsulated class**

* Class must be declared with public keyword.
* There must be at least one constructor in the class and must be declared with public keyword.
* All the class members must be declared with private keyword.
* We need to use getters() and setters() to access and initialize value in another class.

**Encapsulation in my framework**

I have used the Encapsulation in my framework in the PageObjectModel class.

In the pom class at first we will find the location of the virtual object by findBy annotations and after finding the location virtual object becomes web element and we declare the web elements initially with private keyword.

In the pom class we use the constructor, and whenever we create the object of the pom class of particular webpage in our main test script then the constructor of that particular pom class will be executed in which we have PageFactory.initElement(driver,this) from which ‘this’ keyword will hold the object address of current pom class and initializes to the driver.

**Advantages of using pom class and Advantages of using Encapsulation rules**

* Maintenance and Modification of all the web elements with respect to each and every webpage will be easy.
* We can avoid StaleElementReferenceException(Whenever we execute the constructor of pom class, the driver focusses exactly on the current webpage by this we can avoid the exception)
* We can restrict or limit the access scope of web elements of each and every webpage by using private keyword to the users or to a team.
* We can achieve the code optimization by using business logics in POM class with respect to scripts.
* Debugging is easy.

**POLYMORPHISM :**

Objects behaving differently based on the input and with same behaviour provides multiple outputs and it can happen during runtime or compile time is called as polymorphism.

1. **Compile Time Polymorphism** : The method declaration binds with the method implementation during the compile time by the compiler based on the arguments is called as compile time polymorphism.

Example: Method Overloading and Constructor Overloading.

**Method Overloading :** Declaring multiple methods in a same class with same method name but with different arguments is called as method overloading.

I have used Method overloading in my framework for the dropdowns handling to select the required dropdown with the help of Select Class.

public void handlingDropdown(WebElement ele ,String vistext) {

Select s = new Select(ele);

s.selectByVisibleText(vistext);

}

public void handlingDropdown(WebElement ele , int indexNo) {

Select s = new Select(ele);

s.selectByIndex(indexNo);

}

1. **Run Time Polymorphism:** The method declaration binds with the method implementation during the execution time by the jvm based on the object creation is called as Run Time Polymorphism.

Example: Method Overriding.

**MethodOverriding :** Giving the new implementation from subclass to the superclass method is called as method overriding.

\*To achieve method overriding we have a criteria to follow -

* We should follow inheritance relationship
* Method name, parameters and return type pf both super class and sub class method should be same
* Visibility of subclass method should be equal or higher than the super class method
* To achieve method overriding, super class method should not be declared as final.

I have used method overriding in my framework where the all the abstract methods present in WebDriver(interface) are given implementation in the RemoteWebDriver(implementation class) and further for their usage we have inherited all implemented methods to classes like ChromeDriver(class),FireFoxDriver(class)....etc....

I have used method overriding in my framework for launching the browser where driver object behaves differently with respect to particular class.

Example : ChromeDriver driver=new ChromeDriver();

WebDriver driver=new ChromeDriver();

**INHERITANCE :**

Acquiring the properties i.e., state and behaviour of a class from one class to another class is called as inheritance.

I have used hierarchal inheritance in my framework to inherit the pre-similar conditions and post-similar which are same for the entire framework by storing all those pre-similar conditions and post-similar conditions with TestNG annotations in a BaseClass which need to be present in generic utility so that all team members can access it and through inheritance only test scripts will be executed from Test classes and rest all will be executed from BaseClass through inheritance.

Ex: Database connection, Launching the browser, Login to application and logout from the application, closing the browser and closing database connection are stored with TestNG annotations and whenever we execute the main test scripts since it is extended to BaseClass, all the codes present in BaseClass and main test script will also be executed.

**ABSTRACTION :**

The process of hiding the complexity of the application and providing only the functionalities to the end user is called as abstraction.

To achieve Abstraction we need

1. Abstract class (which cannot be 100% abstract but 0 to 100%)
2. Interface (partial and it consists of only abstract methods)

**Disadvantages of Abstract class**

* We cannot achieve 100% abstraction but 0 to 100% .
* Multiple inheritance is not possible.
* We can create a constructor of abstract class but we cannot create the object as abstract keyword present in abstract class restricts the object creation.
* We cannot make abstract class as final or private but it must be always public.
* We cannot create a object in Abstract class so no one can access data by creating the object.
* Tight coupling is possible but where as loose coupling is not possible in Abstract class so we go for interface.

I have used abstraction in my framework where all the abstract methods present in WebDriver interface were provided with new implementation in RemoteWebDriver class without creating the object of RemoteWebDriver but we only create ChromeDriver object of chromedriver class. We do upcasting,

ChromeDriver driver=new ChromeDriver();

WebDriver driver=new ChromeDriver();

so now all the abstract methods present in (driver). We are getting only the functionality to use in main test scripts but we exactly don’t know where implementation is happening.