Concepts of OOPS in Selenium Automation Framework

OOP (Object-Oriented Programming) concepts are fundamental to building robust and maintainable Selenium automation frameworks. Let me explain the key OOP concepts and how they apply to Selenium automation:
In this post, we will discuss how and where we applied following OOPs concepts in an Automation Framework.

#1. ABSTRACTION

Abstraction is the methodology of hiding the implementation of internal details and showing the functionality to the users.

Let's see an example of data abstraction in Selenium Automation Framework.
In Page Object Model design pattern, we write locators (such as id, name, xpath etc.,) and the methods in a Page Class. We utilize these locators in tests but we can't see the implementation of the methods. Literally we hide the implementations of the locators from the tests.
Learn more on Abstraction

I don't know how my phone works internally but I know how to make calls
STM
© www.SoftwareTestingMaterial.com

For example, we don't know how our phone works internally. We don't bother about the internal mechanism but still we can make calls.

/* It's our Abstract Class. Here we are creating an Abstract class Phone which contains methods such as turnon(), makeCall(), turnoff(). There is no implementation of these methods here*/

```
public abstract class Phone {
   public abstract void turnon();
   public abstract void makeCall();
   public abstract void turnoff();
}

/*iPhone is inheriting class Phone and it is implementing the method turnoff(), makeCall(), turnoff() */

public class iPhone extends Phone {
   @Override
   public void turnon() {
      System.out.println("iPhone Turn ON");
   }

   @Override
   public void makeCall() {
      System.out.println("iPhone makes a call");
   }
```

```
    @Override
    public void turnoff() {
        System.out.println(“iPhone Turn OFF");
    }
}

/* We call methods using this abstractClass which is our Main Class */

public class abstractClass {
    public static void main(String args[])
    {
        Phone object = new iPhone();
        object.makeCall();
    }
}
```

In Java, abstraction is achieved by interfaces and abstract classes. Using interfaces, we can achieve 100% abstraction.

Let’s see interface concept below.

#2. INTERFACE

Basic statement we all know in Selenium is **WebDriver driver = new FirefoxDriver();**

Detailed explanation on why we write WebDriver driver = new FirefoxDriver(); in Selenium.

WebDriver itself is an Interface. So based on the above statement **WebDriver driver = new FirefoxDriver();** we are initialising Firefox browser using Selenium WebDriver. It means we are creating a *reference variable (driver)* of the *interface (WebDriver)* and creating an *Object*. Here *WebDriver* is an *Interface* as mentioned earlier and *FirefoxDriver* is a *class*.

An interface in Java looks similar to a class but both the interface and class are two different concepts. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract. We can achieve 100% abstraction and multiple inheritance in Java with Interface.

#3. INHERITANCE

The mechanism in Java by which one class acquires the properties (instance variables) and functionalities of another class is known as Inheritance.

We create a Base Class in the Automation Framework to initialise WebDriver interface, WebDriver waits, Property files, Excels, etc., in the Base Class.

We extend the Base Class in other classes such as Tests and Utility Class.
Here we extend one class (Base Class like WebDriver Interface) into other class (like Tests, Utility Class) is known as Inheritance.

#4. POLYMORPHISM

Polymorphism allows us to perform a task in multiple ways.
Combination of overloading and overriding is known as Polymorphism. We will see both overloading and overriding below.

#1. METHOD OVERLOADING

We use **Implicit wait** in Selenium. Implicit wait is an example of overloading. In Implicit wait we use different time stamps such as SECONDS, MINUTES, HOURS etc.,

Action class in TestNG is also an example of overloading.
Assert class in TestNG is also an example of overloading.

A class having multiple methods with same name but different parameters is called Method Overloading

#2. METHOD OVERRIDING

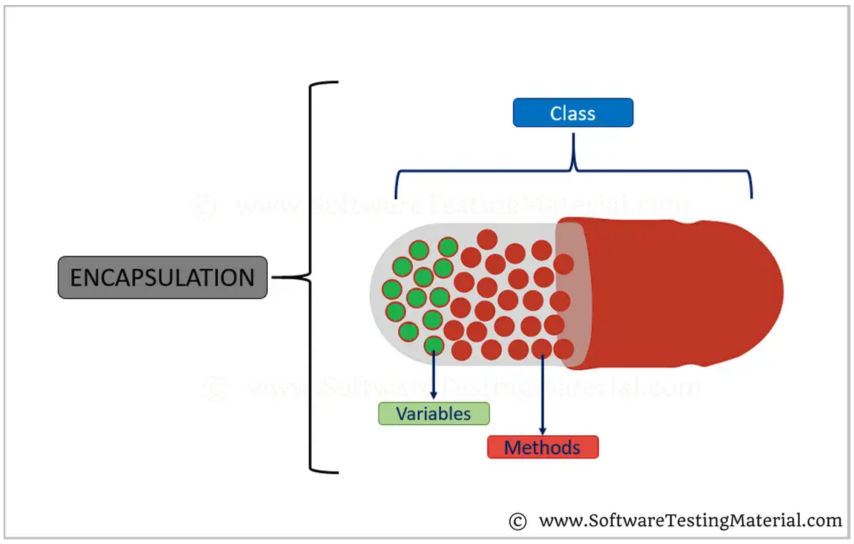
We use a method which was already implemented in another class by changing its parameters. To understand this you need to understand Overriding in Java.

Declaring a method in child class which is already present in the parent class is called Method Overriding. Examples are **get** and **navigate** methods of different drivers in Selenium .
Learn more on Overriding with examples here

#5. ENCAPSULATION

All the classes in a framework are an example of Encapsulation. In POM classes, we declare the data members using **@FindBy** and initialization of data members will be done using Constructor to utilize those in methods.

Encapsulation is a mechanism of binding code and data (variables) together in a single unit.



In Selenium, you can encapsulate web elements and their associated actions within Page Object classes.

/* Save this as - EmployeeDetails File */
/* Here we are creating a class "EmployeeDetails" and defining variables as private just to not let them modify outside of the class. */
/* The getter and setter methods used below can only read and modify the values only within the class. Here in this example, we used getEmployeeName() method to get Employee Name, and used setEmployeeName() method to set Employee Name */

```
public class EmployeeDetails {
  private String employeeName;
  private int employeeId;
  private String employeeYearOfJoining;

  public String getEmployeeName() {
    return employeeName;
  }

  public String getEmployeeId() {
    return employeeId;
  }

  public int getYearOfJoining() {
    return employeeYearOfJoining;
  }

  public void setEmployeeName(String inputEmployeeName) {
    employeeName = inputEmployeeName;
  }

  public void setEmployeeId String inputEmployeeId) {
    employeeId = inputEmployeeId;
  }

  public void setEmployeeYearOfJoining( int inputEmployeeYearOfJoining) {
    employeeYearOfJoining = inputEmployeeYearOfJoining;
  }

}
```

/* Save this as - EncapsulationExample File*/

```
public class EncapsulationExample{

  public static void main(String args[]) {
    EmployeeDetails employeeDetails = new EmployeeDetails();
    employeeDetails.setEmployeeName("Rajkumar");
    employeeDetails.setEmployeeId("EMP"001);
    employeeDetails.setEmployeeYearOfJoining(2020);

    System.out.print("Name : " + encap.getEmployeeName() + " Employee ID: " + encap.getEmployeeID()
+ " Year of Joining : " + encap.getYearOfJoining());
  }
}
```