Selenium framework’s code structure helps you to reuse the code, reduce code maintenance, higher code readability, and allows multiple users to work on the same piece of the program.

* Reduced script maintenance cost
* Reduced tests’ time execution
* Reduced human resources
* Easy reporting

**Language:**In our Selenium Project we are using Java language. Even though Selenium supports multiple languages, we are using Java language is just because most of the automation developers have knowledge on Selenium with Java.

**Type of Framework:** In our project, we are using [Data-driven Framework](https://www.softwaretestingmaterial.com/data-driven-framework-selenium-webdriver/) by using [Page Object Model design pattern](https://www.softwaretestingmaterial.com/page-object-model/) with Page Factory.

**POM:**As per the Page Object Model, we have maintained a class for every web page. Each web page has a separate class and that class holds the functionality and members of that web page. Separate classes for every individual test.

**Packages:** We have separate packages for Pages and Tests. All the web page related classes come under the **Pages** package and all the tests related classes come under **Tests** package.

**Test Base Class:**Test Base class (TestBase.java) deals with all the common functions used by all the pages. This class is responsible for loading the configurations from properties files, Initializing the WebDriver, Implicit Waits, Extent Reports, and also to create the object of FileInputStream which is responsible for pointing towards the file from which the data should be read.

**Utility**

**Properties file:**This file (**config.properties**) stores the information that remains static throughout the framework such as browser-specific information, application URL, screenshots path, etc.

**Screenshots:** Screenshots will be captured and stored in a separate folder and also the screenshots of failed test cases will be added to the extent reports.

**Test Data:** All the historical test data will be kept in an excel sheet (controller.xlsx). By using ‘controller.xlsx’, we pass test data and handle data-driven testing. We use [Apache POI](https://www.softwaretestingmaterial.com/handling-excel-files-using-apache-poi/) to handle excel sheets.

**Maven:** Using Maven for build, execution, and dependency purpose. Integrating the TestNG dependency in the POM.xml file and running this POM.xml file using Jenkins.

**Version Control Tool:** We use Git as a repository to store our test scripts.

**Extent Reports:** For the reporting purpose, we are using Extent Reports. It generates beautiful HTML reports. We use the extent reports for maintaining logs and also to include the screenshots of failed test cases in the Extent Report.

**Jenkins:** By using Jenkins CI (Continuous Integration) Tool, we execute test cases on a daily basis and also for nightly execution based on the schedule. Test Results will be sent to the peers using Jenkins.

JAVA

Oops concept

### ****. 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.

### ****. INTERFACE****

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

***Detailed explanation on why we write***[***WebDriver driver = new FirefoxDriver();***](https://www.softwaretestingmaterial.com/webdriver-driver-new-firefoxdriver/)***in Selenium.***

WebDriver itself is an Interface. So based on the above statement **WebDriver driver = new FirefoxDriver();** we are initializing 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.

### ****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 initialize WebDriver interface, WebDriver waits, Property files, Excels, etc., in the Base Class.

### ****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

Learn more on [Overloading here](https://www.softwaretestingmaterial.com/method-overloading-in-java/).

#### ****#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](https://www.softwaretestingmaterial.com/method-overriding-in-java/)

### ****#5. ENCAPSULATION****

All the classes in a framework are an example of Encapsulation. In [POM classes](https://www.softwaretestingmaterial.com/page-object-model/), we declare the data members using **@FindBy** and initialization of data members will be done using [Constructor](https://www.softwaretestingmaterial.com/java-tutorial/#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 class

ints for creating objects. In a Selenium framework, you might have classes like

* Page Objects (representing web pages)
* Test Cases
* Utility Classes (for common functions)

Objects are instances of classes, allowing you to work with specific elements.

**3. Difference between Array and ArrayList?**

| **Array** | **ArrayList** |
| --- | --- |
| Array is static | ArrayList is dynamic |
| Size of the array should be given at the time of array declaration. We cannot change the size of array after creating it | Size of the array may not be required. It changest the size dynamically. Capacity of ArrayList increases automatically whenever we add elements to an ArrayList |
| Array can contain both primitive data types as well as objects | ArrayList cannot contain primitive data types. It contains only objects |
| Arrays are multidimensional | ArrayList is always single dimension |

**44. Difference between ArrayList and HashSet in Java?**

| **ArrayList** | **HashSet** |
| --- | --- |
| ArrayList implements List interface | HashSet implements Set interface |
| ArrayList allows duplicates | HashSet doesn’t allow duplicates |
| ArrayList is an ordered collection and maintains insertion order of elements | HashSet is an unordered collection and doesn’t maintain insertion order |
| ArrayList is backed by an Array | HashSet is backed by an HashMap instance |
| ArrayList is an index based | HashSet is object based |
| In ArrayList, we can retrive object by calling get() method or remove object by calling remove() method | In HashSet, we can’t achieve get() method |
| **ArrayList** | **HashSet** |
| ArrayList implements List interface | HashSet implements Set interface |
| ArrayList allows duplicates | HashSet doesn’t allow duplicates |
| ArrayList is an ordered collection and maintains insertion order of elements | HashSet is an unordered collection and doesn’t maintain insertion order |
| ArrayList is backed by an Array | HashSet is backed by an HashMap instance |
| ArrayList is an index based | HashSet is object based |
| In ArrayList, we can retrive object by calling get() method or remove object by calling remove() method | In HashSet, we can’t achieve get() method |

Learn more on [Array](https://www.softwaretestingmaterial.com/arrays-in-java/) and [ArrayList](https://www.softwaretestingmaterial.com/arraylist-in-java/) with sample programs

| **ArrayList** | **HashSet** |
| --- | --- |
| ArrayList implements List interface | HashSet implements Set interface |
| ArrayList allows duplicates | HashSet doesn’t allow duplicates |
| ArrayList is an ordered collection and maintains insertion order of elements | HashSet is an unordered collection and doesn’t maintain insertion order |
| ArrayList is backed by an Array | HashSet is backed by an HashMap instance |
| ArrayList is an index based | HashSet is object based |
| In ArrayList, we can retrive object by calling get() method or remove object by calling remove() method | In HashSet, we can’t achieve get() method |

ArrayArrayList

**hrow keyword**

* The throw keyword is used to explicitly throw an exception in the program inside a function or inside a block of code.
* The checked exceptions cannot be propagated with throw only.
* The throw keyword is followed by an instance.
* The throw keyword is used within the method.
* You cannot throw multiple exceptions.

**throws keyword**

* The throws keyword is used in the method signature to declare an exception which might get thrown by the function while executing the code.
* The checked exception can be propagated with throws
* The throws keyword is followed by class.
* The throws keyword is used with the method signature.
* You can declare multiple exceptions, e.g., public void method()throws IOException, SQLException

Top of Form