Encapsulation in Java

**Encapsulation in Java** is a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines.



We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

Encapsulation is a fundamental principle of object-oriented programming (OOP) that involves bundling the data (variables) and methods (functions) that operate on the data into a single unit called a class. In Selenium WebDriver with Java for an e-commerce application, encapsulation can be effectively used to manage web elements and actions. Here's an explanation of how encapsulation can be implemented with a practical example:

**Example Scenario**

Let's consider an e-commerce application with a login page and a product search page. We'll create encapsulated classes for these pages.

**Steps to Implement Encapsulation**

**Create Page Object Model (POM) Classes:**

* Each page in the application will have a corresponding Java class.
* These classes will contain private member variables for web elements.
* Public methods will be provided to interact with these elements.

**Encapsulate Web Elements and Methods:**

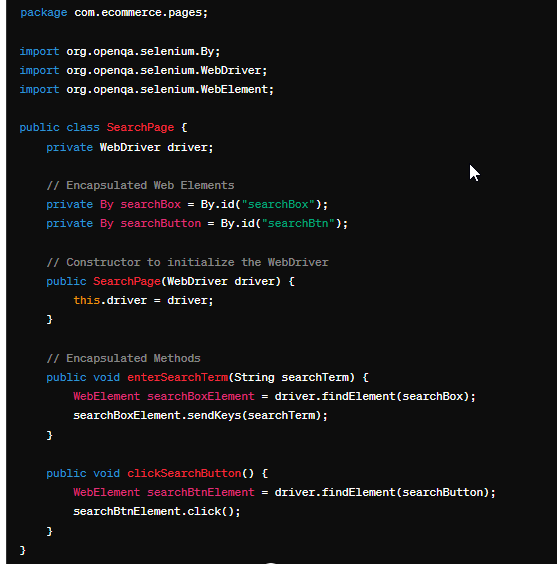
Use private access modifiers for web elements.

Provide public methods to perform actions on these elements.

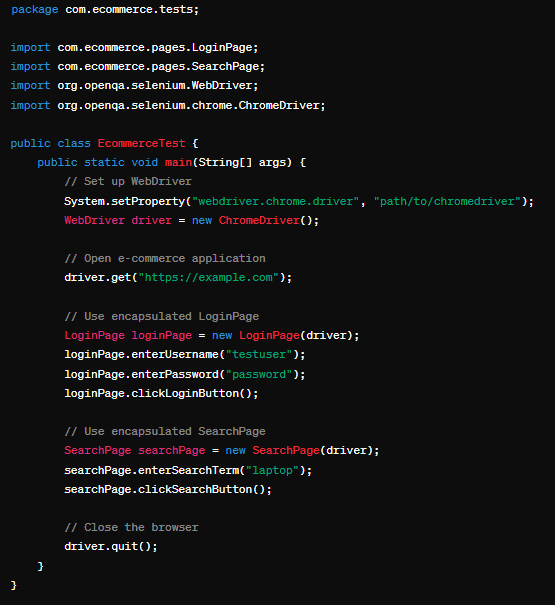
**Login Page Example**

**1. Create LoginPage Class**

**2. Create 2.SearchPage Class**

****

**3. Using the Encapsulated Classes in a Test**

****

**Explanation**

**Encapsulation in the LoginPage and SearchPage Classes:**

**Web Elements:**

The web elements (usernameField, passwordField, searchBox, etc.) are declared as private members. This restricts direct access to these elements from outside the class.

**Methods:**

Public methods (enterUsername, enterPassword, clickLoginButton, etc.) are provided to interact with these web elements. This ensures that any interaction with the elements goes through controlled methods, maintaining the integrity of the interaction logic.

**Benefits of Encapsulation:**

Maintainability: If the locator for a web element changes, you only need to update it in one place (within the page class).

Reusability: The methods in page classes can be reused across different test cases.

Readability: Test scripts become cleaner and easier to read, as they abstract the underlying implementation details.

By following this encapsulation approach, we can create a robust and maintainable Selenium test automation framework for any e-commerce application.

# Interface in Java

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is a mechanism to achieve [*abstraction*](https://www.javatpoint.com/abstract-class-in-java). There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple [inheritance in Java](https://www.javatpoint.com/inheritance-in-java).

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

It cannot be instantiated just like the abstract class.

Since Java 8, we can have **default and static methods** in an interface.

Since Java 9, we can have **private methods** in an interface.

## Why use Java interface?

There are mainly three reasons to use interface. They are given below.

* It is used to achieve abstraction.
* By interface, we can support the functionality of multiple inheritance.
* It can be used to achieve loose coupling.



## How to declare an interface?

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

In an e-commerce application, testing the user interface (UI) is crucial to ensure that customers have a smooth and error-free experience while browsing and purchasing products. Selenium, a popular open-source tool for automating web browsers, is commonly used for UI testing. Using interfaces in Selenium with Java allows for a more modular, flexible, and maintainable codebase. Below is an example demonstrating the use of interfaces in Selenium for an e-commerce application.

**Example Scenario** Let's assume we are testing the login functionality of an e-commerce application.

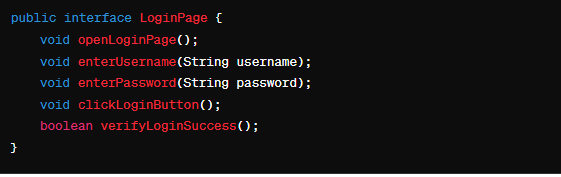
**The process involves:**

* Opening the e-commerce website.
* Navigating to the login page.
* Entering the username and password.
* Clicking the login button.
* Verifying the login success.

**Step-by-Step Explanation**

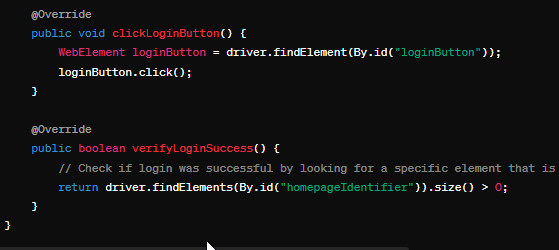
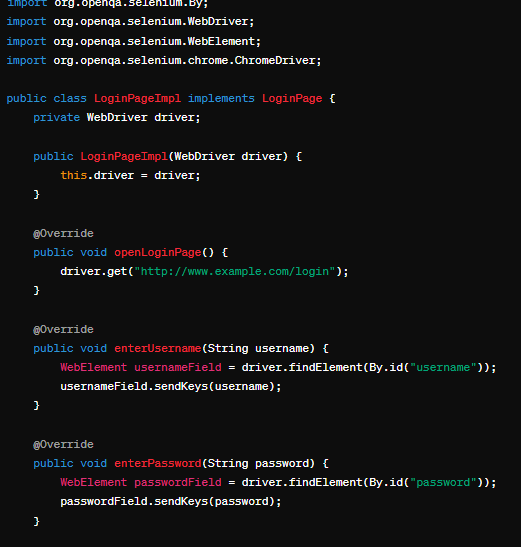
**Define the Interface:**

The interface will outline the methods for common actions on the login page.



**Implement the Interface:**

Create a class that implements this interface. This class will contain the actual Selenium WebDriver code to interact with the web elements.



**3. Use the Implementation in a Test:**

Write a test case that uses the **LoginPageImpl** class to perform the login operation.

****

**Explanation**

**1. Interface Definition:**

**LoginPage** interface declares methods for actions related to login functionality. This allows us to define what actions are needed without worrying about how they will be implemented.

**2. Interface Implementation:**

**LoginPageImpl** class implements the **LoginPage** interface, providing concrete implementations for each method. This class uses Selenium WebDriver to interact with web elements on the login page.

**3. Test Case:**

**LoginTest** class sets up the WebDriver, initializes the **LoginPageImpl**, and defines a test method **testLogin** that performs the login action and verifies the result using assertions.

**Benefits of Using Interfaces**

**Modularity:** The code is organized into separate components, making it easier to manage and understand.

**Flexibility:** Different implementations of the interface can be created for different browsers or different versions of the login page.

**Maintainability:** Changes in the UI (e.g., changing element locators) need to be updated only in the implementation class, not in the test cases.

**Reusability:** The interface can be reused across multiple test cases or even different projects if the login functionality is similar.

By using interfaces in Selenium with Java, you can create a clean, organized, and scalable framework for testing e-commerce applications or any other web applications.

# Polymorphism in Java

**Polymorphism in Java** is a concept by which we can perform a single action in different ways. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

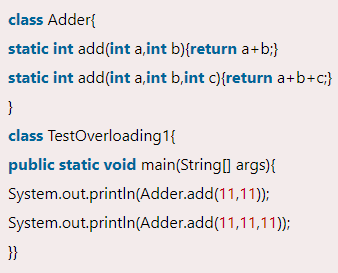
There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

If you overload a static method in Java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

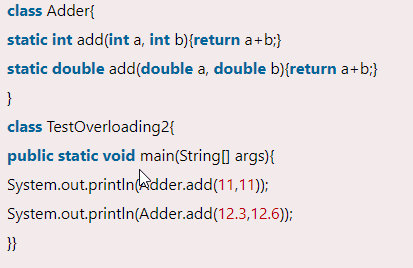
### Different ways to overload the method

There are two ways to overload the method in java

1. **By changing number of arguments**

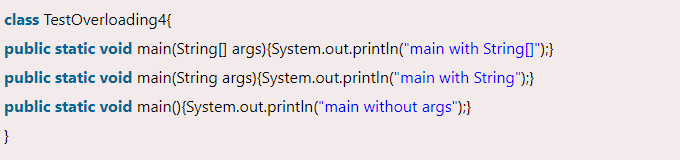


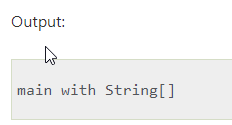
1. **By changing the data type**



### Can we overload java main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. But [JVM](https://www.javatpoint.com/jvm-java-virtual-machine) calls main() method which receives string array as arguments only. Let's see the simple example:





Here’s how polymorphism can be used in Selenium Java code for an eCommerce application:

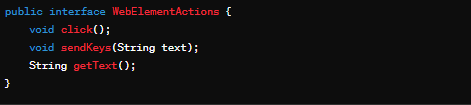
**Example Scenario**

Consider an eCommerce application where we have different types of elements like buttons, text fields, and dropdowns that we need to interact with. Instead of writing separate methods for each type of element, we can use polymorphism to create a unified approach to handle these interactions.

**Step-by-Step Explanation**

**Define a Common Interface**

First, create a common interface that will be implemented by all element classes. This interface will declare methods that all element types must implement.



**Implement Concrete Classes**

Next, implement the interface in different classes for each type of web element. Each class will provide its own implementation of the interface methods.





**Use Polymorphism in Test Scripts**

In your test scripts, use the interface type to interact with different web elements. This allows you to handle different types of elements in a unified way.

**Benefits of Using Polymorphism**

Code Reusability: You can reuse the same interface and methods for different types of web elements.

Maintainability: Easier to maintain and extend the code. Adding a new type of web element is straightforward—simply implement the interface.

Readability: Makes the code more readable and organized by abstracting away the details of each web element type.

