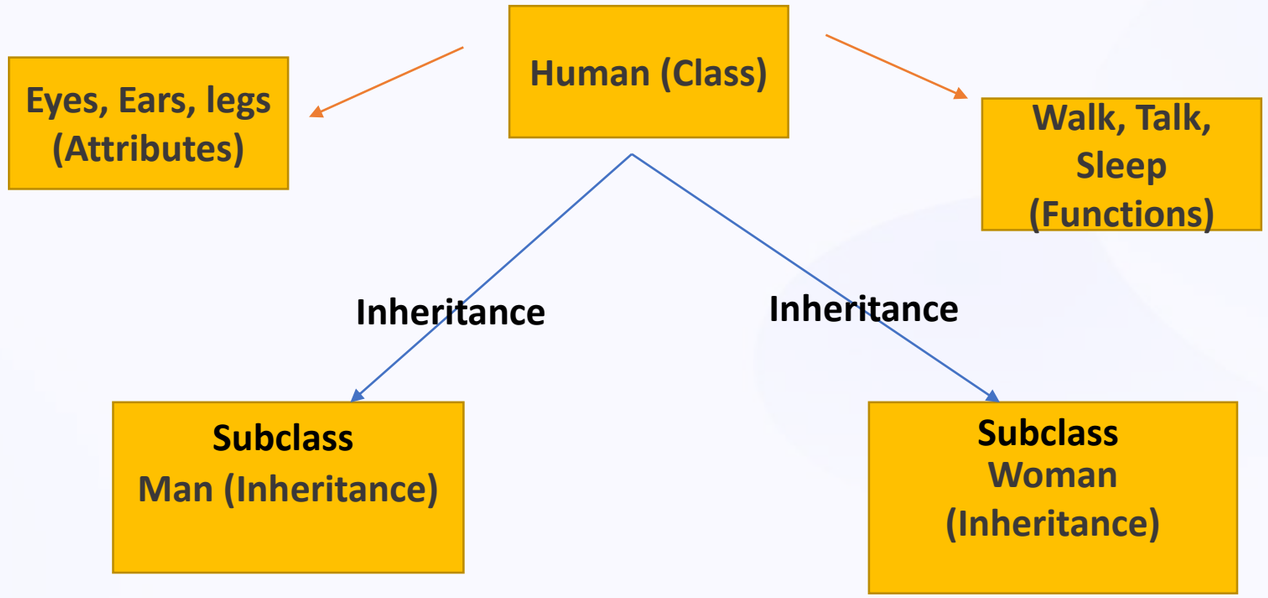
Introduction to OOPS concepts in Selenium Automation

Object-Oriented Programming (OOP) in Selenium Automation is crucial for building robust and scalable test scripts. It involves various principles and concepts that facilitate the efficient implementation of automation frameworks and testing suites.



Objects: God created Rishi, Raveena, Sunny, Mary, Salman, John, Samuel, Anoushka, Rama

Polymorphism: God gave both classes some polymorphic capabilities wherein they could act differently based on the input given. For instance, the function "ConsumeThroughMouth" could behave differently based on the type of input given. If food was given, the function has extra logic to use the teeth to grind it. It milk or water was given, the function used the lips in a pout shape instead of using the teeth. So different use of attributes. So God patented this fancy sounding term and further also brought in a new associated term called function overloading as he was overloading multiple capabilites onto the function "ConsumeThroughMouth"

Encapsulation: God's idea of combining the data and functions together to make it a working unit. God patented it with a fancy name "Encapsulation" for that.

Abstraction: Woman and the man can talk. The internal details of how the neural network in the brain processed the signals received and in turns sends a signal to produce a sound is not exposed to the recipient or the outside world.

God is an OOPS programmer

God is omnipotent, merciful, forgiving, almighty and a bunch of other things, but at heart he was an OOPS programmer, and he created the concept. Let us understand how this guy created humans, after his initial forays with algae and bacteria.

Class: He first created a class called Human. Added some attributes and functions. Attributes such as eyes, ears, brain, legs. Functions such as walk, talk, think, sleep etc.

Inheritance : He next created 2 sub classes - Man and Woman. They inherited the attributes and functions of class human. However he added some attributes and functions specific to these classes. For the man and the woman class, he added attributes specific to their anatomy. He added functions specific to these classes, for instance he gave the function "GiveBirthToNewHuman" to the woman class. It was a complex function intended to spawn new objects and he did a great job there.

Polymorphism : God gave both classes some polymorphic capabilities wherein they could act differently based on the input given. For instance, the function "ConsumeThroughMouth" could behave differently based on the type of input given. If food was given, the function has extra logic to use the teeth to grind it. It milk or water was given, the function used the lips in a pout shape instead of using the teeth. So different use of attributes. So God patented this fancy sounding term and further also brought in a new associated term called function overloading as he was overloading multiple capabilites onto the function "ConsumeThroughMouth"

Abstraction : God thought it would be good if not many details and only what was necessary was given to the external world. For instance the woman and the man can talk. The internal details of how the neural network in the brain processed the signals received and in turns sends a signal to produce a sound is not exposed to the recepient or the outside world. Hence the paradigm of abstraction.

Encapsulation: God's idea of combining the data and functions together to make it a working unit. God patented it with a fancy name "Encapsulation" for that, since he was on a roll with the names already - polymorphism, abstraction and what not!

Object: Finally, it was time to have some fun, and create some instances of the classes made. And viola, God created Adam, Eve, Rishi, Chunnu, Munni, Raveena, Sunny, Mary, Salman, John, Samuel, Anoushka, Rama and many many more. And that's how this OOPS programmer created *Human Life.*

Class

1

Blueprint

Acts as a blueprint or template for creating objects with similar attributes and behaviors.

2

State and Behavior

Encapsulates data and the operations that can be performed on the data.

3

Extensibility

Allows for extension and modification to meet specific requirements.

Abstraction

1 Data Hiding

Abstraction allows the concealment of unnecessary details while exposing essential functionalities, promoting simplicity and enhanced code readability.

2 Modularity

It enables the segregation of features into distinct modules, eliminating dependencies and promoting maintainability and reusability.

3 Real-world Example

An example could be an abstract class providing a common interface for multiple concrete classes with different implementations.

- **Abstraction:** 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.

- Selenium WebDriver interface has many abstract methods like get(String url), quit(), close(), getWindowHandle(), getWindowHandles(), getTitle() etc. WebDriver has nested interfaces like Window , Navigation , Timeouts etc. These nested interfaces are used to perform operations like back(), forward() etc.

- **Abstraction:** Abstract away common functionalities into base classes or interfaces. For example, you might create an interface for logging, and different classes can implement this interface based on the specific logging mechanisms used.

```java
public interface Logger {
    void logMessage(String message);
}

public class ConsoleLogger implements Logger {
    public void logMessage(String message) {
        System.out.println("Console: " + message);
    }
}

public class FileLogger implements Logger {
    public void logMessage(String message) {
        // Log to a file
    }
}
```

Polymorphism: Leverage polymorphism to allow different classes or methods to be used interchangeably.
For instance, you might have a generic method for clicking elements that works with various types of UI elements.

```
public class ElementClicker {
    public void clickElement(WebElement element) {
        element.click();
    }
}

// Usage
ElementClicker clicker = new ElementClicker();
clicker.clickElement(loginPage.getLoginButton());
```

- **Inheritance:** Use inheritance to create a hierarchy of classes.
- A base class can contain common methods and properties shared among different components, and derived classes can inherit and extend this functionality.

```
public class BasePage {
    protected WebDriver driver;

    public BasePage(WebDriver driver) {
        this.driver = driver;
    }
}

public class LoginPage extends BasePage {
    private WebElement usernameField;
    private WebElement passwordField;
    private WebElement loginButton;

    public LoginPage(WebDriver driver) {
        super(driver);
        // Additional properties and methods specific to LoginPage
    }
}
```

- Encapsulation: Encapsulate the interaction logic for each page or component within respective classes.
- Hide the internal details of how elements are located and manipulated, exposing only essential methods.
- 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.

```
public class HomePage {
    private WebDriver driver;
    private WebElement welcomeMessage;

    public HomePage(WebDriver driver) {
        this.driver = driver;
        this.welcomeMessage = driver.findElement(By.id("welcome-mes
    }

    public String getWelcomeMessage() {
        return welcomeMessage.getText();
    }
}
```

- **Method Overloading**

```
import org.openqa.selenium.By;
public class SeleniumExample {
   private WebDriver driver;
   public SeleniumExample(WebDriver driver) {
      this.driver = driver;
   }
   // Enter text into an input field using its By locator
   public void enterText(By locator, String text) {
      WebElement element = driver.findElement(locator);
      element.sendKeys(text);
   }
   // Enter text into an input field using its WebElement reference
   public void enterText(WebElement element, String text) {
      element.sendKeys(text);
   }
   public static void main(String[] args) {
      // Assuming 'driver' is instantiated and the web page is open
      SeleniumExample seleniumExample = new SeleniumExample(driver);
Example 1: Entering text using By locator
      seleniumExample.enterText(By.name("username"), "user123");
Example 2: Entering text using WebElement reference
      WebElement usernameField = driver.findElement(By.name("username"));
      seleniumExample.enterText(usernameField, "user123");
   }
}
```

In the example, the enterText methods are overloaded. There are two versions of each method—one that takes a By locator and another that takes a WebElement reference. This allows flexibility when interacting with web elements, as you can choose to locate elements using different strategies or reuse a previously located element.

- **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 .

- **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

Encapsulation

Data Protection

Encapsulation safeguards the data within an object, preventing unauthorized access and modification.

Data Hiding

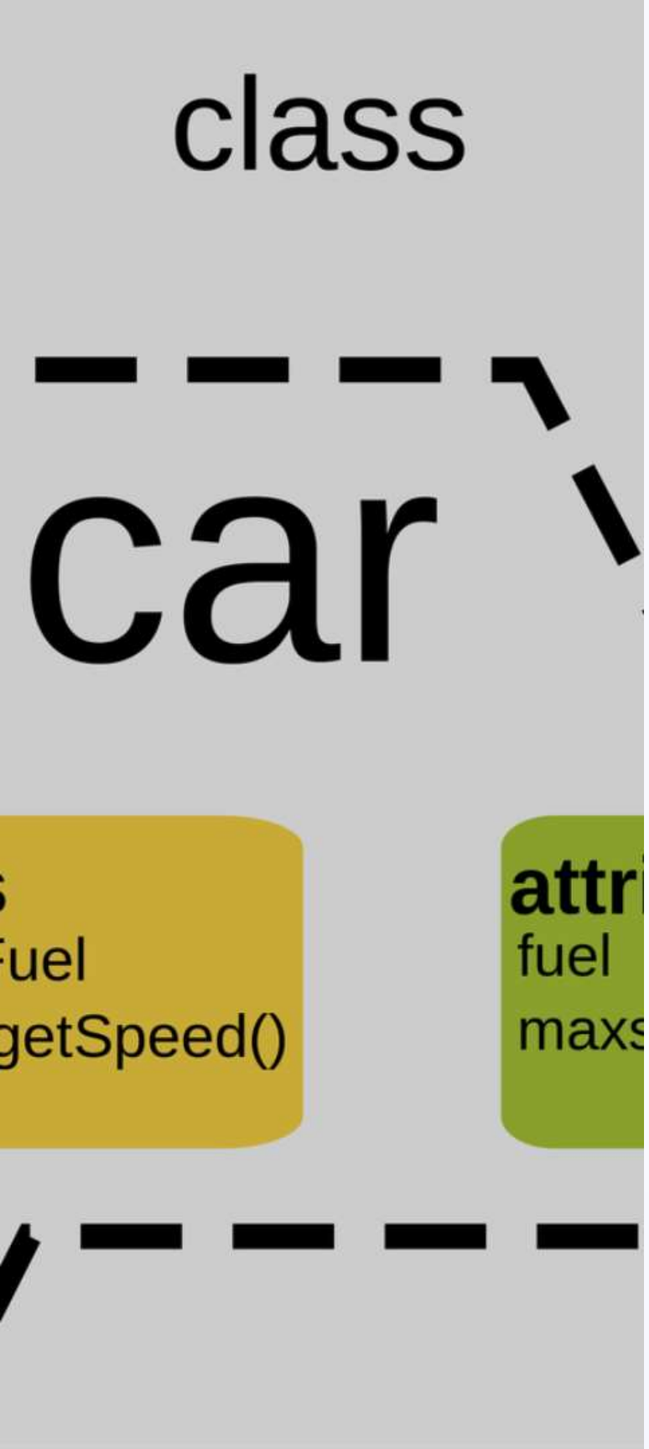
It conceals the internal state of an object, reducing the likelihood of unintended interference and ensuring security.

Inheritance

Parent Class	Child Class
Provides the base characteristics	Inherits features from the parent
Supports code reusability	Allows adding new features

POLYMORPHISM

- Polymorphism can be utilized when interacting with different web elements on a web page.
- For instance, you can define a common interface or base class for web elements like buttons, input fields, and dropdowns.
- Each specific web element class can then implement this interface or extend the base class, allowing you to interact with them using a common set of methods regardless of their specific type.



Objects

1

Instance of a Class

Represents real-world entities or concepts, possessing unique attributes and behaviors.

2

Data and Methods

Contains data (attributes) and behavior (methods), encapsulating state and functionality.

3

Dynamic Engagement

Interacts with other objects and systems to accomplish specific tasks.

Methods

1 **Functionality**

Methods encapsulate specific functionalities, promoting reusability and modularity.

2 **Polymorphic Behavior**

Supports polymorphism, allowing a method to behave differently based on the object invoking it.

3 **Access Modifiers**

Encapsulates methods and controls access to them, ensuring proper interaction and usage.

Conclusion

The foundation of Oops concepts in Selenium Automation is vital for creating maintainable and scalable automation frameworks. Understanding these principles enables efficient test case development and enhances the robustness of automated testing suites.