Java - Introduction to Programming
Lecture 11

Polymorphism Concepts in Selenium Automation Framework

1. Method Overloading: We Use Implicit wait in selenium. Implicit wait is an example of method overloading. In implicit wait, we use different time stamps such as SECONDS, MINUTES, and HOURS etc.

e.g.

driver.manage().timeouts().implicitlyWait(10,TimeUnit.SECONDS) ;

2. **Method Overriding:** 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.

```java
package myPackage;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.edge.EdgeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class BaseTest {
    WebDriver driver;

    // Method to be overridden
    public void launchBrowser() {
        System.out.println("Launching default browser...");
        driver = new ChromeDriver(); // Default browser
    }

    public class ChromeTest extends BaseTest {
        @Override
        public void launchBrowser() {
            System.out.println("Launching Chrome browser...");

            driver = new ChromeDriver();
        }
    }

    public class FirefoxTest extends BaseTest {
        @Override
        public void launchBrowser() {
            System.out.println("Launching Firefox browser...");
            driver = new FirefoxDriver();
        }
    }

    public class EdgeTest extends BaseTest {
        @Override
        public void launchBrowser() {
            System.out.println("Launching Edge browser...");
            driver = new EdgeDriver();
        }
    }
}
```

Package in Java

Package is a group of similar types of classes, interfaces and sub-packages. Packages can be built-in or user defined.

Built-in packages - java, util, io etc.

import java.util.*;

Here, **util** is a sub-package created inside the **java** package.

In Java, we can import classes from a package using either of the following methods:

1. Import a specific class:

import java.util.Vector;

This imports only the Vector class from the java.util package.

2. Import all classes from a package:

import java.util.*;

This imports all classes and interfaces from the java.util package but does not include sub-packages.

Types of Java Packages:

- Built-in Packages
- User-defined Packages

1. Built-in Packages

These packages consist of a large number of classes, which are a part of Java. Some of the commonly used built-in packages are:

- java.lang: Contains language support classes (e.g classes that defines primitive data types, math operations). This package is automatically imported.

- java.io: Contains classes for supporting input / output operations.

- java.util: Contains utility classes, which implement data structures like Linked List, Dictionary and support; for Date / Time operations.

- java.applet: Contains classes for creating Applets.
- java.awt: Contain classes for implementing the components for graphical user interfaces (like button, menus etc). 6)

- java.net: Contain classes for supporting networking operations.

2. **User-defined Packages**

These are the packages that are defined by the user.

1. Create the Package:
First we create a directory myPackage (name should be same as the name of the package). Then create the MyClass inside the directory with the first statement being the package names.

```java
package myPackage;

public class MyClass {
    public void getNames(String s)
    {
        System.out.println(s);
    }

    }
```

```java
import myPackage.*;

public class Myclass2 {
    public static void main(String args[]) {

        // Initializing the String variable
        // with a value
        String s = "piyushkeshari";

        // Creating an instance of class MyClass in
        // the package.
        MyClass myc = new MyClass();
        myc.getNames(s);
     }

}
```

Access Modifiers in Java

- **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

```
// error while using class from different package with
// private access modifier
package myPackage;

// Class A
class A {
     private void display() {
        System.out.println("GeeksforGeeks");
    }
}

// Class B
class B {
    public static void main(String args[]) {
        A obj = new A();

        // Trying to access private method
        // of another class
        obj.display();
    }
}
```

- **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

```
package myPackage;
public class Account{
     void display(){
        System.out.println("Hello World");
     }
}
```

```
package myPackage2;
```

```java
import myPackage.*;
public class Myclass2 {
    public static void main(String args[]) {
     Account acc =  new Account();
        acc.display();
     }

}
```

- **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

```java
package myPackage;

public class A {
    protected void display() {
        System.out.println("piyush keshari");
    }
}
```

```java
package myPackage2;
import myPackage.*;

// Class B is subclass of A
class B extends A {
    public static void main(String args[]) {
        B obj = new B();
        obj.display();
    }
}
```

- **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

```java
package myPackage;
public class Account{
    public void display(){
        System.out.println("Hello World");
     }
}
```

```java
package myPackage2;

import myPackage.*;
public class Myclass2 {
    public static void main(String args[]) {
     Account acc =  new Account();
        acc.display();
     }

}
```

Encapsulation

Encapsulation is the process of combining data and functions into a single unit called class. In Encapsulation, the data is not accessed directly; it is accessed through the functions present inside the class. In simpler words, attributes of the class are kept private and public getter and setter methods are provided to manipulate these attributes. Thus, encapsulation makes the concept of data hiding possible.(**Data hiding**: a language feature to restrict access to members of an object, reducing the negative effect due to dependencies. e.g. "protected", "private" feature in Java).

- In encapsulation, the variables or data of a class are hidden from any other class and can be accessed only through any member function of its own class.
- A private class can hide its members or methods from the end user, using abstraction to hide implementation details, by combining data hiding and abstraction.
- Encapsulation can be achieved by declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.
- It is more defined with the setter and getter method.

```java

package myPackage2;


class Person {
    // Encapsulating the name and age
    // only approachable and used using
    // methods defined
    private String name;
    private int age;

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public int getAge()
    {
        return age;
    }

    public void setAge(int age)
    {
        this.age = age;
    }
}

// Driver Class
public class Encapsulation {
// main function
    public static void main(String[] args)
    {
        // person object created
        Person p = new Person();
        p.setName("John");
        p.setAge(30);
        // Using methods to get the values from the
        // variables
        System.out.println("Name: " + p.getName());
        System.out.println("Age: " + p.getAge());
    }
}
```