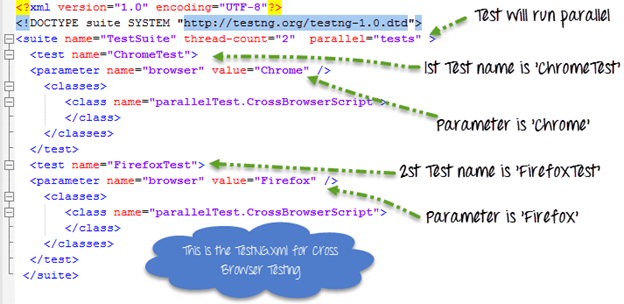
**Mock interview no:1**

**How to perform Cross Browser Testing**

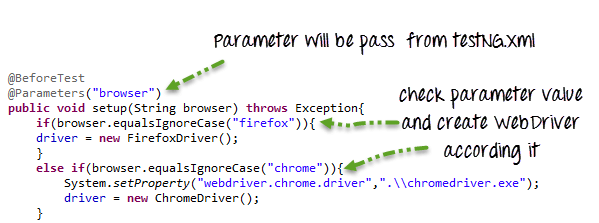
If we are using Selenium WebDriver, we can automate test cases using Internet Explorer, FireFox, Chrome, Safari browsers.

To execute test cases with different browsers in the same machine at the same time we can integrate[Testng](https://www.guru99.com/all-about-testng-and-selenium.html)framework with Selenium WebDriver.

Your testing.xml will look like that,



This testing.xml will map with the[Test Case](https://www.guru99.com/test-case.html)which will look like that



Here because the testing.xml has two Test tags (‘ChromeTest’,’FirefoxTest’),this test case will execute two times for 2 different browsers.

First Test ‘ChromeTest’ will pass the value of parameter ‘browser’ as ‘chrome’ so ChromeDriver will be executed. This test case will run on Chrome browser.

Second Test ‘FirefoxTest’ will pass the value of parameter ‘browser’ as ‘Firefox’ so FirefoxDriver will be executed. This test case will run on FireFox browser.

Complete Code:

**Guru99CrossBrowserScript.java**

package parallelTest;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.edge.EdgeDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Parameters;

import org.testng.annotations.Test;

public class CrossBrowserScript {

WebDriver driver;

/\*\*

\* This function will execute before each Test tag in testng.xml

\* @param browser

\* @throws Exception

\*/

@BeforeTest

@Parameters("browser")

public void setup(String browser) throws Exception{

//Check if parameter passed from TestNG is 'firefox'

if(browser.equalsIgnoreCase("firefox")){

//create firefox instance

System.setProperty("webdriver.gecko.driver", ".\\geckodriver.exe");

driver = new FirefoxDriver();

}

//Check if parameter passed as 'chrome'

else if(browser.equalsIgnoreCase("chrome")){

//set path to chromedriver.exe

System.setProperty("webdriver.chrome.driver",".\\chromedriver.exe");

//create chrome instance

driver = new ChromeDriver();

}

//Check if parameter passed as 'Edge'

else if(browser.equalsIgnoreCase("Edge")){

//set path to Edge.exe

System.setProperty("webdriver.edge.driver",".\\MicrosoftWebDriver.exe");

//create Edge instance

driver = new EdgeDriver();

}

else{

//If no browser passed throw exception

throw new Exception("Browser is not correct");

}

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

}

@Test

public void testParameterWithXML() throws InterruptedException{

driver.get("http://demo.guru99.com/V4/");

//Find user name

WebElement userName = driver.findElement(By.name("uid"));

//Fill user name

userName.sendKeys("guru99");

//Find password

WebElement password = driver.findElement(By.name("password"));

//Fill password

password.sendKeys("guru99");

}

}

**testing.xml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="TestSuite" thread-count="2" parallel="tests" >

<test name="ChromeTest">

<parameter name="browser" value="Chrome" />

<classes>

<class name="parallelTest.CrossBrowserScript">

</class>

</classes>

</test>

<test name="FirefoxTest">

<parameter name="browser" value="Firefox" />

<classes>

<class name="parallelTest.CrossBrowserScript">

</class>

</classes>

</test>

<test name="EdgeTest">

<parameter name="browser" value="Edge" />

<classes>

<class name="parallelTest.CrossBrowserScript">

</class>

</classes>

</test>

</suite>

package TestNG;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.ie.InternetExplorerDriver;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.\*;

import org.testng.annotations.\*;

public class TestNGClass {

private WebDriver driver;

private String URL = "http://www.calculator.net";

@Parameters("browser")

@BeforeTest

public void launchapp(String browser) {

if (browser.equalsIgnoreCase("firefox")) {

System.out.println(" Executing on FireFox");

driver = new FirefoxDriver();

driver.get(URL);

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

driver.manage() wait.window().maximize();

} else if (browser.equalsIgnoreCase("chrome")) {

System.out.println(" Executing on CHROME");

System.out.println("Executing on IE");

System.setProperty("webdriver.chrome.driver", "D:\\chromedriver.exe");

driver = new ChromeDriver();

driver.get(URL);

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

driver.manage().window().maximize();

} else if (browser.equalsIgnoreCase("ie")) {

System.out.println("Executing on IE");

System.setProperty("webdriver.ie.driver", "D:\\IEDriverServer.exe");

driver = new InternetExplorerDriver();

driver.get(URL);

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

driver.manage().window().maximize();

} else {

throw new IllegalArgumentException("The Browser Type is Undefined");

}

}

@Test

public void calculatepercent() {

// Click on Math Calculators

driver.findElement(By.xpath(".//\*[@id = 'menu']/div[3]/a")).click();

// Click on Percent Calculators

driver.findElement(By.xpath(".//\*[@id = 'menu']/div[4]/div[3]/a")).click();

// Enter value 10 in the first number of the percent Calculator

driver.findElement(By.id("cpar1")).sendKeys("10");

// Enter value 50 in the second number of the percent Calculator

driver.findElement(By.id("cpar2")).sendKeys("50");

// Click Calculate Button

driver.findElement(By.xpath(".//\*[@id = 'content']/table/tbody/tr/td[2]/input")).click();

// Get the Result Text based on its xpath

String result =

driver.findElement(By.xpath(".//\*[@id = 'content']/p[2]/span/font/b")).getText();

// Print a Log In message to the screen

System.out.println(" The Result is " + result);

if(result.equals("5")) {

System.out.println(" The Result is Pass");

} else {

System.out.println(" The Result is Fail");

}

}

@AfterTest

public void closeBrowser() {

driver.close();

}

}

### **Handling Multiple Windows in Selenium**

The user scenario being automated here is: Open a new tab and then switch back to the last window to complete the other pending activities. In such scenarios, Selenium helps to handle multiple windows through window handlers and javascript executors.

### **What is a window handle?**

It is a unique identifier that holds the address of all the windows. Think of it as a pointer to a window, which returns the string value. It is assumed that each browser will have a unique window handle. This window handle function helps to retrieve the handles of all windows.

**Syntax**

1. **get.windowhandle()**: This method helps to get the window handle of the current window
2. **get.windowhandles()**: This method helps to get the handles of all the windows opened
3. **set**: This method helps to set the window handles in the form of a string. set<string> set= driver.get.windowhandles()
4. **switch to:** This method helps to switch between the windows
5. **action**: This method helps to perform certain actions on the windows

These are some of the methods that will be used to handle multiple windows in Selenium.

**People also read:**[How to handle Alerts and Popups in Selenium?](https://www.browserstack.com/guide/alerts-and-popups-in-selenium)

### **Example of handling multiple windows**

**Scenario**: Navigate to the [Browserstack home page](https://www.browserstack.com/" \o "BrowserStack Home Page). This is the parent window. From the parent window, let’s see how to handle the child windows and then again navigate back to the parent windows.

Steps to execute:

1. Get the handle of the parent window using the command: String parentWindowHandle = driver.getWindowHandle();
2. Print the window handle of the parent window.
3. Find the element on the web page using an ID which is an element locator.
4. Open multiple child windows.
5. Iterate through child windows.
6. Get the handles of all the windows that are currently open using the command: Set<String> allWindowHandles = driver.getWindowHandles(); which returns the set of handles.
7. Use the **SwitchTo** command to switch to the desired window and also pass the URL of the web page.

Refer to the complete program below.

Before running the code, one should do a quick check on 6 things to avoid while running selenium scrips. [**Check it out**](https://www.browserstack.com/guide/things-to-avoid-in-selenium-test-scripts).

import java.util.Iterator;

import java.util.Set;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

public class WindowHandle\_Demo {

public static void main(String[] args) throws Exception {

System.setProperty("webdriver.chrome.driver","Path to the driver");

WebDriver driver = new ChromeDriver();

driver.manage().window().maximize();

// Load the website

driver.get("http://www.naukri.com/");

// It will return the parent window name as a String

String parent=driver.getWindowHandle();

Set<String>s=driver.getWindowHandles();

// Now iterate using Iterator

Iterator<String> I1= s.iterator();

while(I1.hasNext())

{

String child\_window=I1.next();

if(!parent.equals(child\_window))

{

driver.switchTo().window(child\_window);

System.out.println(driver.switchTo().window(child\_window).getTitle());

driver.close();

}

}

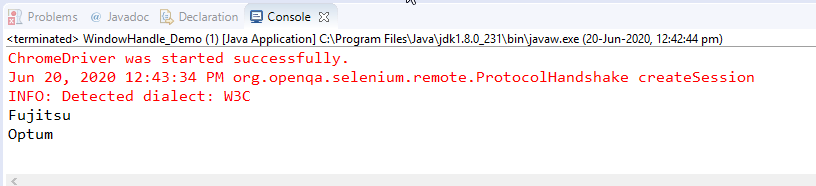
//switch to the parent window

driver.switchTo().window(parent);

}

}

**Output:**



On executing the parent window handle, it will open multiple child windows and then navigate back to the final window handle.

[Run Selenium Tests on Real Device Cloud for Free](https://www.browserstack.com/users/sign_up?ref=guide-handling-multiple-windows-in-selenium-mid&product=automate)

Now let’s perform some actions on the Browserstack website.

**Pro Tip:** Want to dive deeper into Selenium implementation on BrowserStack with free interactive courses and lab exercises? [Visit Test University](http://browserstack.com/test-university?ref=guide)

**Code Snippet**

import java.util.Set;

import org.openqa.selenium.By;

import org.openqa.selenium.JavascriptExecutor;

import org.openqa.selenium.Keys;

import org.openqa.selenium.Point;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.interactions.Actions;

public class selenium { public static void main(String[] args) throws Exception

{

System.setProperty("webdriver.chrome.driver", "D:\\Selenium\\chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.get("https://www.browserstack.com/");

String title = driver.getTitle(); System.out.println(title);

JavascriptExecutor js = (JavascriptExecutor) driver;

driver.findElement(By.xpath("//span[contains(text(),'Solutions')]")).click();

driver.findElement(By.xpath("//a[contains(text(),'Geolocation Testing')]")).click();

js.executeScript("window.scrollBy(0,40)");

WebElement link = driver.findElement(By.xpath("//a[@id='product-menu-toggle']//span[@class='account-down-caret']//\*[local-name()='svg']"));

Actions newwin = new Actions(driver);

newwin.keyDown(Keys.SHIFT).click(link).keyUp(Keys.SHIFT).build().perform();

//Thread.sleep(2000);

//js.executeScript("window.scrollBy(0,400)");

Thread.sleep(3000);

Set<String> windows = driver.getWindowHandles();

System.out.println(windows);

System.out.println("a1");

for (String window : windows)

{

driver.switchTo().window(window);

if (driver.getTitle().contains("Most Reliable App & Cross Browser Testing Platform | Browserstack"))

{

System.out.println("a2");

js.executeScript("window.scrollBy(0,1000)");

System.out.println("b1");

driver.findElement(By.xpath("//a[@id='logo']//\*[local-name()='svg']")).click();

driver.findElement(By.xpath("//a[@id='signupModalButton']")).click();

driver.manage().window().setPosition(new Point(2000, 0));

}

}

Thread.sleep(3000);

Set<String> windows1 = driver.getWindowHandles();

System.out.println(windows1);

System.out.println("a3");

for (String window : windows1)

{

driver.switchTo().window(window);

System.out.println("a4");

js.executeScript("window.scrollBy(0,400)");

}

}

}

# Selenium Wait – Implicit, Explicit and Fluent Waits

In Selenium, “Waits” play an important role in executing tests. In this tutorial, you will learn various aspects and difference between Implicit and Explicit wait in Selenium.

In this tutorial, you will learn about different types of waits in Selenium:

* [Why Do We Need Waits In Selenium?](https://www.guru99.com/implicit-explicit-waits-selenium.html#why-do-we-need-waits-in-selenium)
* [Implicit Wait](https://www.guru99.com/implicit-explicit-waits-selenium.html#implicit-wait)
* [Explicit Wait](https://www.guru99.com/implicit-explicit-waits-selenium.html#explicit-wait)
* [Fluent Wait](https://www.guru99.com/implicit-explicit-waits-selenium.html#fluent-wait)
* [Difference Between Implicit Wait Vs Explicit Wait](https://www.guru99.com/implicit-explicit-waits-selenium.html#difference-between-implicit-Wait-vs-explicit-wait)

## Why Do We Need Waits In Selenium?

Most of the web applications are developed using [Ajax](https://www.guru99.com/php-ajax.html) and [Javascript](https://www.guru99.com/interactive-javascript-tutorials.html). When a page is loaded by the browser the elements which we want to interact with may load at different time intervals.

Not only it makes this difficult to identify the element but also if the element is not located it will throw an “**ElementNotVisibleException**” exception. Using Selenium Waits, we can resolve this problem.

Let’s consider a scenario where we have to use both implicit and explicit waits in our test. Assume that implicit wait time is set to 20 seconds and explicit wait time is set to 10 seconds.

Suppose we are trying to find an element which has some **“ExpectedConditions** “(Explicit Wait), If the element is not located within the time frame defined by the Explicit wait(10 Seconds), It will use the time frame defined by implicit wait(20 seconds) before throwing an “**ElementNotVisibleException**“.

**Selenium Web Driver Waits**

1. Implicit Wait
2. Explicit Wait

## Implicit Wait in Selenium

The **Implicit Wait in Selenium** is used to tell the web driver to wait for a certain amount of time before it throws a “No Such Element Exception”. The default setting is 0. Once we set the time, the web driver will wait for the element for that time before throwing an exception.

Selenium Web Driver has borrowed the idea of implicit waits from Watir.

In the below example we have declared an implicit wait with the time frame of 10 seconds. It means that if the element is not located on the web page within that time frame, it will throw an exception.

To declare implicit wait in Selenium WebDriver:

### Implicit Wait syntax:

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

package guru.test99;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.testng.annotations.Test;

public class AppTest {

protected WebDriver driver;

@Test

public void guru99tutorials() throws InterruptedException

{

System.setProperty ("webdriver.chrome.driver",".\\chromedriver.exe" );

driver = new ChromeDriver();

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

String eTitle = "Demo Guru99 Page";

String aTitle = "" ;

// launch Chrome and redirect it to the Base URL

driver.get("http://demo.guru99.com/test/guru99home/" );

//Maximizes the browser window

driver.manage().window().maximize() ;

//get the actual value of the title

aTitle = driver.getTitle();

//compare the actual title with the expected title

if (aTitle.equals(eTitle))

{

System.out.println( "Test Passed") ;

}

else {

System.out.println( "Test Failed" );

}

//close browser

driver.close();

}

}

**Explanation of Code**

In the above example,

**Consider Following Code:**

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

Implicit wait will accept 2 parameters, the first parameter will accept the time as an integer value and the second parameter will accept the time measurement in terms of SECONDS, MINUTES, MILISECOND, MICROSECONDS, NANOSECONDS, DAYS, HOURS, etc.

## Explicit Wait in Selenium

The **Explicit Wait in Selenium** is used to tell the Web Driver to wait for certain conditions (Expected Conditions) or maximum time exceeded before throwing “ElementNotVisibleException” exception. It is an intelligent kind of wait, but it can be applied only for specified elements. It gives better options than implicit wait as it waits for dynamically loaded Ajax elements.

Once we declare explicit wait we have to use “**ExpectedConditions**” or we can configure how frequently we want to check the condition using **Fluent Wait**. These days while implementing we are using **Thread.Sleep()**generally it is not recommended to use

In the below example, we are creating reference wait for “**WebDriverWait**” class and instantiating using “**WebDriver**” reference, and we are giving a maximum time frame of 20 seconds.

### Explicit Wait syntax:

WebDriverWait wait = new WebDriverWait(WebDriverRefrence,TimeOut);

package guru.test99;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.support.ui.ExpectedConditions;

import org.openqa.selenium.support.ui.WebDriverWait;

import org.testng.annotations.Test;

public class AppTest2 {

protected WebDriver driver;

@Test

public void guru99tutorials() throws InterruptedException

{

System.setProperty ("webdriver.chrome.driver",".\\chromedriver.exe" );

driver = new ChromeDriver();

WebDriverWait wait=new WebDriverWait(driver, 20);

String eTitle = "Demo Guru99 Page";

String aTitle = "" ;

// launch Chrome and redirect it to the Base URL

driver.get("http://demo.guru99.com/test/guru99home/" );

//Maximizes the browser window

driver.manage().window().maximize() ;

//get the actual value of the title

aTitle = driver.getTitle();

//compare the actual title with the expected title

if (aTitle.contentEquals(eTitle))

{

System.out.println( "Test Passed") ;

}

else {

System.out.println( "Test Failed" );

}

WebElement guru99seleniumlink;

guru99seleniumlink= wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath( "/html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/div/div[2]/div[2]/div/div/div/div/div[1]/div/div/a/i")));

guru99seleniumlink.click();

}

}

**Explanation of Code**

**Consider Following Code:**

WebElement guru99seleniumlink;

guru99seleniumlink = wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath("/html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/div/div[2]/div[2]/div/div/div/div/div[1]/div/div/a/i")));

guru99seleniumlink.click();

In this WebDriver wait example, wait for the amount of time defined in the “**WebDriverWait**” class or the “**ExpectedConditions**” to occur whichever occurs first.

The above[Java](https://www.guru99.com/java-tutorial.html)code states that we are waiting for an element for the time frame of 20 seconds as defined in the “**WebDriverWait**” class on the webpage until the “**ExpectedConditions**” are met and the condition is “**visibilityofElementLocated**“.

The following are the Expected Conditions that can be used in Selenium Explicit Wait

1. alertIsPresent()
2. elementSelectionStateToBe()
3. elementToBeClickable()
4. elementToBeSelected()
5. frameToBeAvaliableAndSwitchToIt()
6. invisibilityOfTheElementLocated()
7. invisibilityOfElementWithText()
8. presenceOfAllElementsLocatedBy()
9. presenceOfElementLocated()
10. textToBePresentInElement()
11. textToBePresentInElementLocated()
12. textToBePresentInElementValue()
13. titleIs()
14. titleContains()
15. visibilityOf()
16. visibilityOfAllElements()
17. visibilityOfAllElementsLocatedBy()
18. visibilityOfElementLocated()

## Fluent Wait in Selenium

The **Fluent Wait in Selenium** is used to define maximum time for the web driver to wait for a condition, as well as the frequency with which we want to check the condition before throwing an “ElementNotVisibleException” exception. It checks for the web element at regular intervals until the object is found or timeout happens.

**Frequency:**Setting up a repeat cycle with the time frame to verify/check the condition at the regular interval of time

Let’s consider a scenario where an element is loaded at different intervals of time. The element might load within 10 seconds, 20 seconds or even more then that if we declare an explicit wait of 20 seconds. It will wait till the specified time before throwing an exception. In such scenarios, the fluent wait is the ideal wait to use as this will try to find the element at different frequency until it finds it or the final timer runs out.

### Fluent Wait syntax:

Wait wait = new FluentWait(WebDriver reference)

.withTimeout(timeout, SECONDS)

.pollingEvery(timeout, SECONDS)

.ignoring(Exception.class);

Above code is deprecated in Selenium v3.11 and above. You need to use

Wait wait = new FluentWait(WebDriver reference)

.withTimeout(Duration.ofSeconds(SECONDS))

.pollingEvery(Duration.ofSeconds(SECONDS))

.ignoring(Exception.class);

package guru.test99;

import org.testng.annotations.Test;

import java.util.NoSuchElementException;

import java.util.concurrent.TimeUnit;

import java.util.function.Function;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.support.ui.ExpectedConditions;

import org.openqa.selenium.support.ui.FluentWait;

import org.openqa.selenium.support.ui.Wait;

import org.openqa.selenium.support.ui.WebDriverWait;

import org.testng.annotations.Test;

public class AppTest3 {

protected WebDriver driver;

@Test

public void guru99tutorials() throws InterruptedException

{

System.setProperty ("webdriver.chrome.driver",".\\chromedriver.exe" );

String eTitle = "Demo Guru99 Page";

String aTitle = "" ;

driver = new ChromeDriver();

// launch Chrome and redirect it to the Base URL

driver.get("http://demo.guru99.com/test/guru99home/" );

//Maximizes the browser window

driver.manage().window().maximize() ;

//get the actual value of the title

aTitle = driver.getTitle();

//compare the actual title with the expected title

if (aTitle.contentEquals(eTitle))

{

System.out.println( "Test Passed") ;

}

else {

System.out.println( "Test Failed" );

}

Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)

.withTimeout(30, TimeUnit.SECONDS)

.pollingEvery(5, TimeUnit.SECONDS)

.ignoring(NoSuchElementException.class);

WebElement clickseleniumlink = wait.until(new Function<WebDriver, WebElement>(){

public WebElement apply(WebDriver driver ) {

return driver.findElement(By.xpath("/html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/div/div[2]/div[2]/div/div/div/div/div[1]/div/div/a/i"));

}

});

//click on the selenium link

clickseleniumlink.click();

//close~ browser

driver.close() ;

}

}

**Explanation of Code**

**Consider Following Code:**

Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)

.withTimeout(30, TimeUnit.SECONDS)

.pollingEvery(5, TimeUnit.SECONDS)

.ignoring(NoSuchElementException.class);

In the above example, we are declaring a fluent wait with the timeout of 30 seconds and the frequency is set to 5 seconds by ignoring “**NoSuchElementException**”

**Consider Following Code:**

public WebElement apply(WebDriver driver) {

return driver.findElement(By.xpath("/html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/div/div[2]/div[2]/div/div/div/div/div[1]/div/div/a/i"));

We have created a new function to identify the Web Element on the page. (Ex: Here Web Element is nothing but the Selenium link on the webpage).

Frequency is set to 5 seconds and the maximum time is set to 30 seconds. Thus this means that it will check for the element on the web page at every 5 seconds for the maximum time of 30 seconds. If the element is located within this time frame it will perform the operations else it will throw an “**ElementNotVisibleException**”

**Also Check:-**[Selenium IDE Tutorial for Beginners](https://www.guru99.com/introduction-selenuim-ide.html)

## Difference Between Implicit Wait Vs Explicit Wait

Following is the main difference between implicit wait and explicit wait in Selenium:

|  |  |
| --- | --- |
| **Implicit Wait** | **Explicit Wait** |
| * Implicit Wait time is applied to all the elements in the script | * Explicit Wait time is applied only to those elements which are intended by us |
| * In Implicit Wait, we need **not** specify “ExpectedConditions” on the element to be located | * In Explicit Wait, we need to specify “ExpectedConditions” on the element to be located |
| * It is recommended how to take to use when the elements are located with the time frame specified in Selenium implicit wait | * It is recommended to use when the elements are taking long time to load and also for verifying the property of the element like(visibilityOfElementLocated, elementToBeClickable,elementToBeSelected) |

## Conclusion:

Implicit, Explicit and Fluent Wait are the different waits used in Selenium. Usage of these waits are totally based on the elements which are loaded at different intervals of time. It is always not recommended to use Thread.Sleep() while[Testing](https://www.guru99.com/software-testing.html)our application or building our framework.

# How to Handle iFrames in Selenium Webdriver: switchTo()

## iFrame in Selenium Webdriver

**iFrame in Selenium Webdriver** is a web page or an inline frame which is embedded in another web page or an HTML document embedded inside another HTML document. The iframe is often used to add content from other sources like an advertisement into a web page. The iframe is defined with the <**iframe**> tag.

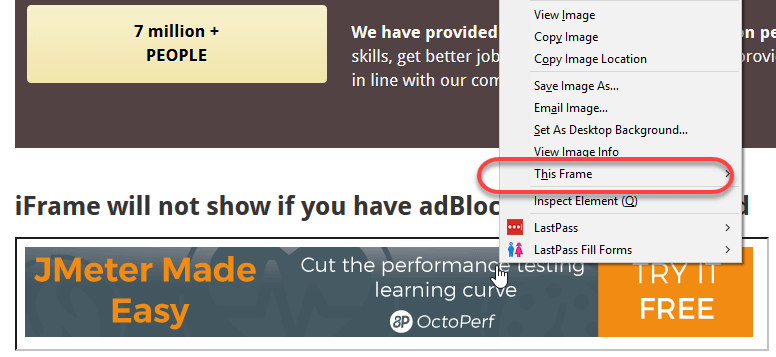
In this tutorial, you will learn –

1. [How to identify the iframe:](https://www.guru99.com/handling-iframes-selenium.html#1)
2. [How do switch over the elements in iframes using Web Driver commands:](https://www.guru99.com/handling-iframes-selenium.html#2)
3. [Concept of Nested Frames (Frames inside Frames):](https://www.guru99.com/handling-iframes-selenium.html#3)

## How to identify the iframe:

We cannot detect the frames by just seeing the page or by inspecting Firebug.

Observe the below image, Advertisement being displayed is an Iframe, we cannot locate or recognize that by just inspecting using Firebug. So the question is how can you identify the iframe?

How to identify the iframe using Selenium WebDriver

We can identify the frames in Selenium using methods given below:

* Right click on the element, If you find the option like ‘This Frame’ then it is an iframe.(Please refer the above diagram)
* Right click on the page and click ‘View Page Source’ and Search with the ‘iframe’, if you can find any tag name with the ‘iframe’ then it is meaning to say the page consisting an iframe.

In above diagram, you can see that ‘**This Frame**‘ option is available upon right clicking, so we are now sure that it is an iframe.

We can even identify total number of iframes by using below snippet.

Int size = driver.findElements(By.tagName("iframe")).size();

## How to switch over the elements in iframes using Web Driver commands:

Basically, we can switch over the elements and handle frames in Selenium using 3 ways.

* **By Index**
* **By Name or Id**
* **By Web Element**

**Switch to the frame by index:**

Index is one of the attributes for frame handling in Selenium through which we can switch to it.

Index of the iframe starts with ‘0’.

Suppose if there are 100 frames in page, we can switch to frame in Selenium by using index.

* driver.switchTo().frame(0);
* driver.switchTo().frame(1);

**Switch to the frame by Name or ID:**

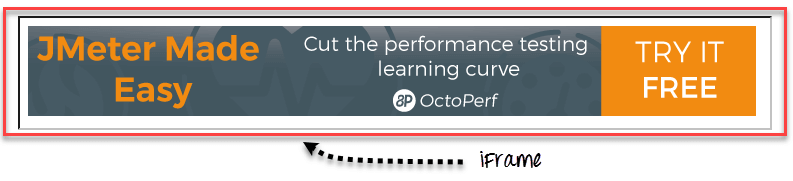
Name and ID are attributes for handling frames in Selenium through which we can switch to the iframe.

* driver.switchTo().frame(“iframe1”);
* driver.switchTo().frame(“id of the element”);

**Example of Switching to iframe through ID:**

Let’s take an example to switch frame in Selenium displayed in the below image. Our requirement is to click the iframe.

We can access this iframe through this below URL: http://demo.guru99.com/test/guru99home/



It is impossible to click iframe directly through[XPath](https://www.guru99.com/xpath-selenium.html)since it is an iframe. First we have to switch to the frame and then we can click using xpath.

**Step 1)**

WebDriver driver = new FirefoxDriver();

driver.get("http://demo.guru99.com/test/guru99home/");

driver.manage().window().maximize();

* We initialise the Firefox driver.
* Navigate to the “guru99” site which consist the iframe.
* Maximized the window.

**Step 2)**

driver.switchTo().frame("a077aa5e");

* In this step we need to find out the id of the iframe by inspecting through Firebug.
* Then switch to the iframe through ID.

**Step 3)**

driver.findElement(By.xpath("html/body/a/img")).click();

* Here we need to find out the xpath of the element to be clicked.
* Click the element using web driver command shown above.

Here is the complete code:

public class SwitchToFrame\_ID {

public static void main(String[] args) {

WebDriver driver = new FirefoxDriver(); //navigates to the Browser

driver.get("http://demo.guru99.com/test/guru99home/");

// navigates to the page consisting an iframe

driver.manage().window().maximize();

driver.switchTo().frame("a077aa5e"); //switching the frame by ID

System.out.println("\*\*\*\*\*\*\*\*We are switch to the iframe\*\*\*\*\*\*\*");

driver.findElement(By.xpath("html/body/a/img")).click();

//Clicks the iframe

System.out.println("\*\*\*\*\*\*\*\*\*We are done\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

}

**Output:**

Browser navigates to the page consisting the above iframe and clicks on the iframe.

**Switch to the frame by Web Element:**

We can even switch to the iframe using web element .

* driver.switchTo().frame(WebElement);

**How to switch back to the Main Frame**

We have to come out of the iframe.

To move back to the parent frame, you can either use switchTo().parentFrame() or if you want to get back to the main (or most parent) frame, you can use switchTo().defaultContent();

driver.switchTo().parentFrame();

driver.switchTo().defaultContent();

**How to switch over the frame, if we CANNOT switch using ID or Web Element:**

Suppose if there are 100 frames in the page, and there is no ID available, in this case, we just don’t know from which iframe required element is being loaded (It is the case when we do not know the index of the frame also).

The solution for the above concern is, we must find the index of the iframe through which the element is being loaded and then we need to switch to the iframe through the index.

Below are the steps for finding the index of the Frame by which the element is being loaded by using below snippet

**Step 1)**

WebDriver driver = new FirefoxDriver();

driver.get("http://demo.guru99.com/test/guru99home/");

driver.manage().window().maximize();

* Initialise the Firefox driver.
* Navigate to the “guru99” site which consisting the iframe.
* Maximized the window.

**Step 2)**

int size = driver.findElements(By.tagName("iframe")).size();

* The above code finds the total number of iframes present inside the page using the tagname ‘iframe’.

**Step 3)**

**Objective for**this step would be finding out the index of iframe.

for(int i=0; i<=size; i++){

driver.switchTo().frame(i);

int total=driver.findElements(By.xpath("html/body/a/img")).size();

System.out.println(total);

driver.switchTo().defaultContent();}

Above “forloop” iterates all the iframes in the page and it prints ‘1’ if our required iframe was found else returns ‘0’.

**Here is the complete code till step 3:**

public class IndexOfIframe {

public static void main(String[] args) {

WebDriver driver = new FirefoxDriver();

driver.get("http://demo.guru99.com/test/guru99home/");

driver.manage().window().maximize();

//driver.manage().timeouts().implicitlyWait(100, TimeUnit.SECONDS);

int size = driver.findElements(By.tagName("iframe")).size();

for(int i=0; i<=size; i++){

driver.switchTo().frame(i);

int total=driver.findElements(By.xpath("html/body/a/img")).size();

System.out.println(total);

driver.switchTo().defaultContent();}}}

**Execute this program and output would be like below:**

**Output:**

1

0

0

0

0

0

Verify the output, you can find the series of 0’s and 1’s.

* Wherever you find the ‘1’ in output that is the index of Frame by which the element is being loaded.
* Since the index of the iframe starts with ‘0’ if you find the 1 in the 1stplace, then the index is 0.
* If you find 1 in 3rd place, the index is 2.

We can comment out the for loop, once we found the index.  
**Step 4)**

driver.switchTo().frame(0);

* Once you find the index of the element, you can switch over the frame using above command.
* driver.switchTo().frame(index found from the Step 3);

**Step 5)**

driver.findElement(By.xpath("html/body/a/img")).click();

* The above code will clicks the iframe or element in the iframe.

So the complete code would be like below:

public class SwitchToframe {

public static void main(String[] args) throws NoSuchElementException{

WebDriver driver = new FirefoxDriver();

driver.get("http://demo.guru99.com/test/guru99home/");

driver.manage().window().maximize();

//int size = driver.findElements(By.tagName("iframe")).size();

/\*for(int i=0; i<=size; i++){

driver.switchTo().frame(i);

int total=driver.findElements(By.xpath("html/body/a/img")).size();

System.out.println(total);

driver.switchTo().defaultContent(); //switching back from the iframe

}\*/

//Commented the code for finding the index of the element

driver.switchTo().frame(0); //Switching to the frame

System.out.println("\*\*\*\*\*\*\*\*We are switched to the iframe\*\*\*\*\*\*\*");

driver.findElement(By.xpath("html/body/a/img")).click();

//Clicking the element in line with Advertisement

System.out.println("\*\*\*\*\*\*\*\*\*We are done\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

}

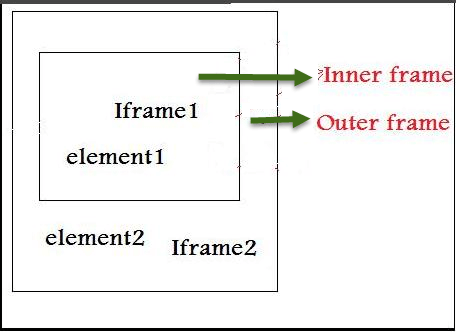
**Output:**  
Browser navigates to the page consisting the above iframe and clicks on the iframe.

## Concept of Nested Frames(Frames inside Frames):

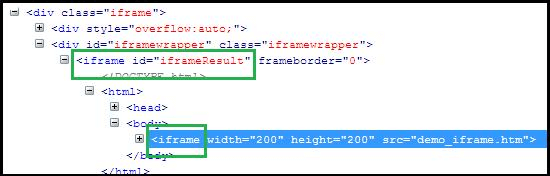
Let’s assume that there are two frames one inside other like shown in below image and our requirement is printing the text in the outer frame and inner frame.  
In the case of nested frames,

* At first we must switch to the outer frame by either Index or ID of the iframe
* Once we switch to the outer frame we can find the total number of iframes inside the outer frame, and
* We can switch to the inner frame by any of the known methods.

While exiting out of the frame, we must exit out in the same order as we entered into it from the inner frame first and then outer frame.

Nested iFrames in Selenium WebDriver

The Html code for the above nested frame is as shown below.



The above HTML code clearly explains the iframe tag (highlighted in green) within another iframe tag, indicating presence of nested iframes.

Below are the steps for switching to outer frame and printing the text on outer frames:

**Step 1)**

WebDriver driver=new FirefoxDriver();

driver.get("Url");

driver.manage().window().maximize();

driver.manage().timeouts().implicitlyWait(2, TimeUnit.SECONDS);

int size = driver.findElements(By.tagName("iframe")).size();

System.out.println("Total Frames --" + size);

// prints the total number of frames

driver.switchTo().frame(0); // Switching the Outer Frame

System.out.println (driver.findElement(By.xpath("xpath of the outer element ")).getText());

* Switch to the outer Frame.
* Prints the text on outer frame.

Once we switch to the outer frame, we should know whether any inner frame present inside the outer frame

**Step 2)**

size = driver.findElements(By.tagName("iframe")).size();

// prints the total number of frames inside outer frame

System.out.println("Total Frames --" + size);

* Finds the total number of iframes inside outer frame.
* If size was found ‘0’ then there is no inner frame inside the frame.

**Step 3)**

driver.switchTo().frame(0); // Switching to innerframe

System.out.println(driver.findElement(By.xpath("xpath of the inner element ")).getText());

* Switch to the inner frame
* Prints the text on the inner frame.

**Here is the complete code:**

public class FramesInsideFrames {

public static void main(String[] args) {

WebDriver driver=new FirefoxDriver();

driver.get("Url");

driver.manage().window().maximize();

driver.manage().timeouts().implicitlyWait(2, TimeUnit.SECONDS);

int size = driver.findElements(By.tagName("iframe")).size();

System.out.println("Total Frames --" + size);

// prints the total number of frames

driver.switchTo().frame(0); // Switching the Outer Frame

System.out.println (driver.findElement(By.xpath("xpath of the outer element ")).getText());

//Printing the text in outer frame

size = driver.findElements(By.tagName("iframe")).size();

// prints the total number of frames inside outer frame

System.out.println("Total Frames --" + size);

driver.switchTo().frame(0); // Switching to innerframe

System.out.println(driver.findElement(By.xpath("xpath of the inner element ")).getText());

//Printing the text in inner frame

driver.switchTo().defaultContent();

}

}

**Output**:  
The output of the above code would print the text in the Inner frame and Outer frame.

# How to Take Screenshot in Selenium WebDriver

## What is Ashot API?

Ashot is a third party utility by Yandex supported by Selenium WebDriver to capture the Screenshots. It takes a screenshot of an individual WebElement as well as a full-page screenshot of a page, which is more significant than screen size.

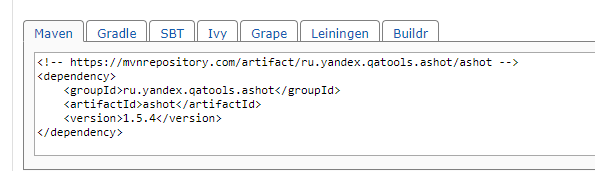
## How to download and configure Ashot API?

There are two methods to configure Ashot API

* 1.Using Maven
* 2.Manually without using any tool

### To configure through Maven:

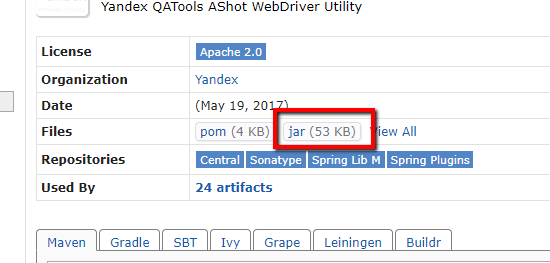
* Go to <https://mvnrepository.com/artifact/ru.yandex.qatools.ashot/ashot>
* Click on the latest version, for now. It is 1.5.4
* Copy the Dependency code and add to your pom.xml file



* Save the file, and Maven will add the jar to your build path
* And now you are ready!!!

### To configure manually without any dependency tool

1. Go to <https://mvnrepository.com/artifact/ru.yandex.qatools.ashot/ashot>
2. Click on the latest version, for now. It is 1.5.4
3. Click on the jar, download and save it on your machine



1. Add the jar file in your build path:
2. In Eclipse, right-click on the project -> go to properties -> Build Path -> Libraries -> Add External jars
3. Select the jar file
4. Apply and Close

## Capture Full Page Screenshot with AShot API

**Step 1)** Create an Ashot object and call takeScreenshot() method if you just want the screenshot for the screen size page.

Screenshot screenshot = new Ashot().takeScreenshot(driver);

But if you want a screenshot of the page bigger then the screen size, call the shootingStrategy() method before calling takeScreenshot() method to set up the policy. Then call a method takeScreenshot() passing the webdriver, for example,

Screenshot screenshot = new AShot().shootingStrategy(ShootingStrategies.viewportPasting(1000)).takeScreenshot(driver);

Here 1000 is scrolled out time in milliseconds, so for taking a screenshot, the program will scroll for each 1000 msec.

**Step 2):** Now, get the image from the screenshot and write it to the file. You can provide the file type as jpg, png, etc.

ImageIO.write(screenshot.getImage(), "jpg", new File(".\\screenshot\\fullimage.jpg"));

Taking a full-page screenshot of a page which is bigger than screen size.

**Example:** Here is the example of capturing a full-page screenshot of http://demo.guru99.com/test/guru99home/ and save to file “screenshot.jpg.”

Due to using the ShootingStrategy class of Ashot API, we will be able to capture a full image of a page bigger than the screen size. Here is the program:

package Guru99;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import org.openqa.selenium.By;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import ru.yandex.qatools.ashot.AShot;

import ru.yandex.qatools.ashot.Screenshot;

import ru.yandex.qatools.ashot.shooting.ShootingStrategies;

public class TestScreenshotUsingAshot {

public static void main(String[] args) throws IOException {

System.setProperty("webdriver.chrome.driver", "c:\\chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.get("http://demo.guru99.com/test/guru99home/");

driver.manage().window().maximize();

Screenshot = new AShot().shootingStrategy(ShootingStrategies.viewportPasting(1000)).takeScreenshot(driver);

ImageIO.write(screenshot.getImage(), "jpg", new File("c:\\ElementScreenshot.jpg"));

}

}

## Taking a screenshot of a particular element of the page

**Example:** Here is the example of capturing element screenshot of Guru 99 logo on http://demo.guru99.com/test/guru99home/ page and save to file “ElementScreenshot.jpg”. Here is the code:

package Guru99;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import org.openqa.selenium.By;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import ru.yandex.qatools.ashot.AShot;

import ru.yandex.qatools.ashot.Screenshot;

import ru.yandex.qatools.ashot.shooting.ShootingStrategies;

public class TestElementScreenshotUsingAshot {

public static void main(String[] args) throws IOException {

System.setProperty("webdriver.chrome.driver", "c:\\chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.get("http://demo.guru99.com/test/guru99home/");

driver.manage().window().maximize();

// Find the element to take a screenshot

WebElement element = driver.findElement(By.xpath ("//\*[@id=\"site-name\"]/a[1]/img"));

// Along with driver pass element also in takeScreenshot() method.

Screenshot = new AShot().shootingStrategy(ShootingStrategies.viewportPasting(1000)).takeScreenshot(driver,element);

ImageIO.write(screenshot.getImage(), "jpg", new File("c:\\ElementScreenshot.jpg"));

}

}

## Image Comparison using AShot

package Guru99;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import ru.yandex.qatools.ashot.AShot;

import ru.yandex.qatools.ashot.Screenshot;

import ru.yandex.qatools.ashot.comparison.ImageDiff;

import ru.yandex.qatools.ashot.comparison.ImageDiffer;

public class TestImageComaprison {

public static void main(String[] args) throws IOException {

System.setProperty("webdriver.chrome.driver", "C:\\chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.get("http://demo.guru99.com/test/guru99home/");

// Find the element and take a screenshot

WebElement logoElement = driver.findElement(By.xpath("//\*[@id=\"site-name\"]/a[1]/img"));

Screenshot logoElementScreenshot = new AShot().takeScreenshot(driver, logoElemnent);

// read the image to compare

BufferedImage expectedImage = ImageIO.read(new File("C:\\Guru99logo.png"));

BufferedImage actualImage = logoElementScreenshot.getImage();

// Create ImageDiffer object and call method makeDiff()

ImageDiffer imgDiff = new ImageDiffer();

ImageDiff diff = imgDiff.makeDiff(actualImage, expectedImage);

if (diff.hasDiff() == true) {

System.out.println("Images are same");

} else {

System.out.println("Images are different");

}

driver.quit();

}

}

## Summary

* Ashot API is a freeware from Yandex.
* It is a utility for taking a screenshot in Selenium.
* It helps you to take a screenshot of an individual WebElement on different platforms like desktop browsers, iOS Simulator Mobile Safari, Android Emulator Browser.
* It can take a page screenshot of a page bigger than screen size.
* This feature has been removed in selenium version 3, so Ashot API is a good option.
* It can decorate the screenshots.
* It provides a screenshot comparison.

# **How to get the current invocation count in TestNG?**

TestNG supports multi-invocation of a test method, i.e., a @Test method can be invoked multiple times sequentially or in parallel. If we want to run single @Test 10 times at a single thread, then **invocationCount** can be used.

To invoke a method multiple times, the keyword **invocationCount** = **<int>** is required. For example −

@Test(invocationCount = 10)

In this example, the @Test method will execute for 10 times each on a single thread.

In this article, we will illustrate how to get the current invocation count.

### **Approach/Algorithm to solve this problem**

* **Step 1** − Create a TestNG class, **NewTestngClass**.
* **Step 2** − Write two @Test methods in the class **NewTestngClass** as shown in the programming code section below. Add **invocationCount=10**
* **Step 3** − Create the **testNG.xml** as given below to run the TestNG classes.
* **Step 4** − Now, run the **testNG.xml** or directly testNG class in IDE or compile and run it using command line.
* **Step 5** − In the output, the user can see a total of 1 thread running sequentially for all invocations of @Test.

## Example

Use the following code for the commonTestNG class, **NewTestngClass** −

### **src/ NewTestngClass.java**

import org.testng.ITestContext;

import org.testng.annotations.\*;

public class NewTestngClass {

@Test(invocationCount = 10)

public void testcase1(ITestContext testContext){

System.out.println("Running testcase 1");

int currentCount = testContext.getAllTestMethods()[0].getCurrentInvocationCount();

System.out.println("Executing count: " + currentCount);

}

@Test(invocationCount = 10)

public void testcase2(ITestContext testContext){

System.out.println("Running testcase 2");

int currentCount = testContext.getAllTestMethods()[1].getCurrentInvocationCount();

System.out.println("Executing count: " + currentCount);

}

}

### **testng.xml**

This is a configuration file that is used to organize and run the TestNG test cases. It is very handy when limited tests are needed to execute rather than the full suite.

<?xml version = "1.0" encoding = "UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name = "Suite1">

<test name = "test1">

<classes>

<class name = "NewTestngClass"/>

</classes>

</test>

</suite>

### **Output**

Running testcase 1

Executing count: 0

Running testcase 1

Executing count: 1

Running testcase 1

Executing count: 2

Running testcase 1

Executing count: 3

Running testcase 1

Executing count: 4

Running testcase 1

Executing count: 5

Running testcase 1

Executing count: 6

Running testcase 1

Executing count: 7

Running testcase 1

Executing count: 8

Running testcase 1

Executing count: 9

Running testcase 2

Executing count: 0

Running testcase 2

Executing count: 1

Running testcase 2

Executing count: 2

Running testcase 2

Executing count: 3

Running testcase 2

Executing count: 4

Running testcase 2

Executing count: 5

Running testcase 2

Executing count: 6

Running testcase 2

Executing count: 7

Running testcase 2

Executing count: 8

Running testcase 2

Executing count: 9

===============================================

Suite1

Total tests run: 20, Passes: 20, Failures: 0, Skips: 0

==================================================

# Dataprovider & TestNG XML: Parameterization in Selenium(Example)

As we create software, we always wish it should work differently with a different set of data. When it comes to[Testing](https://www.guru99.com/software-testing.html)the same piece of software, we can’t be unfair to test it with just one set of data. Here again, we need to verify that our system is taking all set of combinations which are expected to support. For that, we need to parameterize our test scipts. Here comes Parameterization in the picture.

## Parameterization in Selenium

**Parameterization in Selenium** is a process to parameterize the test scripts in order to pass multiple data to the application at runtime. It is a strategy of execution which automatically runs test cases multiple times using different values. The concept achieved by parameterizing the test scripts is called **Data Driven Testing**.

In this tutorial, you will learn-

* [Type of Parameterization in TestNG-](https://www.guru99.com/parameterization-using-xml-and-dataproviders-selenium.html#1)
* [Parameters annotation with Testng.xml](https://www.guru99.com/parameterization-using-xml-and-dataproviders-selenium.html#2)
* [Troubleshooting](https://www.guru99.com/parameterization-using-xml-and-dataproviders-selenium.html#3)
* [Parameters using Dataprovider](https://www.guru99.com/parameterization-using-xml-and-dataproviders-selenium.html#4)
* [Invoke DataProvider from different class](https://www.guru99.com/parameterization-using-xml-and-dataproviders-selenium.html#5)
* [Types of Parameters in Dataprovider](https://www.guru99.com/parameterization-using-xml-and-dataproviders-selenium.html#6)

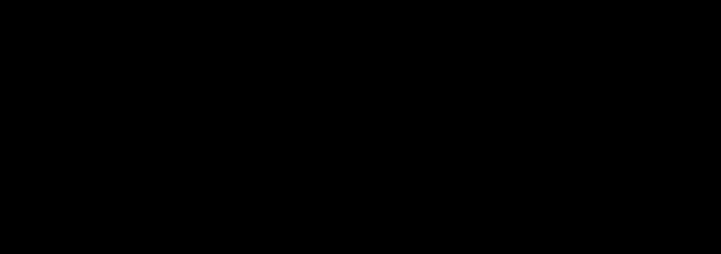
## Type of Parameterization in TestNG-

To make parameterization more clear, we will go through the parameterization options in one the most popular framework for Selenium Webdriver – **TestNG**.

There are **two ways** by which we can achieve parameterization in TestNG

1. With the help of **Parameters** **annotation** and **TestNG XML** file.

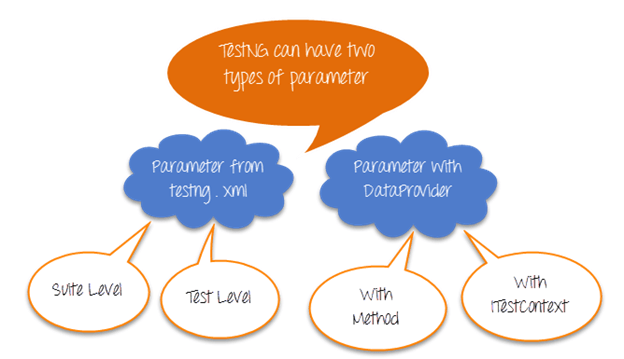
TestNG: Parameterization using XML & DataProvider in Selenium

1. 

00:0835:04EXPLORE MORELearn Java Programming with Beginners Tutorial08:32Linux Tutorial for Beginners: Introduction to Linux Operating...01:35What is Integration Testing Software Testing Tutorial03:04What is JVM (Java Virtual Machine) with Architecture JAVA...02:24How to write a TEST CASE Software Testing Tutorial01:08Seven Testing Principles Software Testing05:01Linux File Permissions Commands with Examples13:29How to use Text tool in Photoshop CC Tutorial08:32What is NoSQL Database Tutorial02:00Important Linux Commands for Beginners Linux Tutorial15:03

With the help of **DataProvider** annotation.

TestNG: Parameterization using XML & DataProvider in Selenium



Parameters from Testng.xml can be suite or test level

Parameter from DataProvider can take Method and ITestContext as the parameter.

Let’s study them in detail –

## Parameters Annotation in TestNG

**Parameters Annotation in TestNG** is a method used to pass values to the test methods as arguments using .xml file. Users may be required to pass the values to the test methods during run time. The @Parameters annotation method can be used in any method having @Test, @Before, @After or @Factory annotation.

## Parameters annotation with Testng.xml

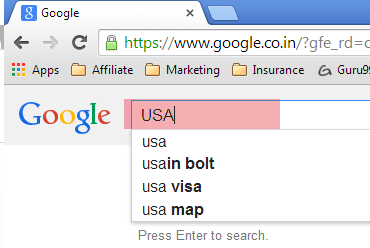
Select parameterization using annotations when you do want to deal with complexity & the number of input combinations are less.

Let see how this works

Test Scenario

Step 1) Launch browser & go to Google.com

Step 2) Enter a search keyword



Step 3) Verify the inputted value is same as that provided by our test data

Step 4) Repeat 2 & 3 until all values are inputted

|  |  |
| --- | --- |
| **Test Author** | **SearchKey** |
| Guru99 | India |
| Krishna | USA |
| Bhupesh | China |

Here is an example of how to do it WITHOUT parameters

package parameters;

import org.testng.annotations.Test;

import org.testng.AssertJUnit;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

public class NoParameterWithTestNGXML {

String driverPath = "C:\\geckodriver.exe";

WebDriver driver;

@Test

public void testNoParameter() throws InterruptedException{

String author = "guru99";

String searchKey = "india";

System.setProperty("webdriver.gecko.driver", driverPath);

driver= new FirefoxDriver();

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

driver.get("https://google.com");

WebElement searchText = driver.findElement(By.name("q"));

//Searching text in google text box

searchText.sendKeys(searchKey);

System.out.println("Welcome ->"+author+" Your search key is->"+searchKey);

System.out.println("Thread will sleep now");

Thread.sleep(3000);

System.out.println("Value in Google Search Box = "+searchText.getAttribute("value") +" ::: Value given by input = "+searchKey);

//verifying the value in google search box

AssertJUnit.assertTrue(searchText.getAttribute("value").equalsIgnoreCase(searchKey));

}

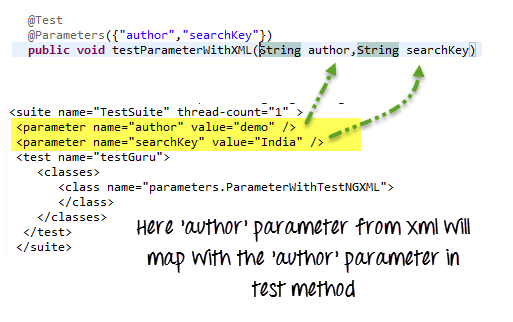
}

A Study, the above example. Just imagine how complex the code will become when we do this for 3 input combinations

Now, let’s parameterize this using TestNG

To do so, you will need to

* Create an XML file which will store the parameters
* In the test, add annotation @Parameters



Here is the complete code

**Test Level TestNG.xml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="TestSuite" thread-count="3" >

<parameter name="author" value="Guru99" />

<parameter name="searchKey" value="India" />

<test name="testGuru">

<parameter name="searchKey" value="UK" />

<classes>

<class name="parameters.ParameterWithTestNGXML">

</class>

</classes>

</test>

</suite>

**ParameterWithTestNGXML.java File**

package parameters;

import org.testng.AssertJUnit;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.annotations.Optional;

import org.testng.annotations.Parameters;

import org.testng.annotations.Test;

public class ParameterWithTestNGXML {

String driverPath = "C:\\geckodriver.exe";

WebDriver driver;

@Test

@Parameters({"author","searchKey"})

public void testParameterWithXML( @Optional("Abc") String author,String searchKey) throws InterruptedException{

System.setProperty("webdriver.gecko.driver", driverPath);

driver = new FirefoxDriver();

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

driver.get("https://google.com");

WebElement searchText = driver.findElement(By.name("q"));

//Searching text in google text box

searchText.sendKeys(searchKey);

System.out.println("Welcome ->"+author+" Your search key is->"+searchKey);

System.out.println("Thread will sleep now");

Thread.sleep(3000);

System.out.println("Value in Google Search Box = "+searchText.getAttribute("value") +" ::: Value given by input = "+searchKey);

//verifying the value in google search box

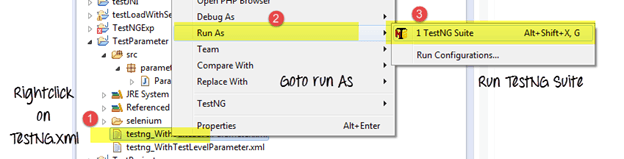
AssertJUnit.assertTrue(searchText.getAttribute("value").equalsIgnoreCase(searchKey));

}

}

Instructions to run the script, select the XML file and Run as Test NG Suite

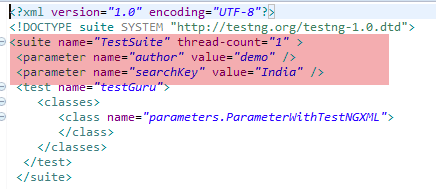
**Right Click on .xml file -> Run as ->**[**Testng**](https://www.guru99.com/all-about-testng-and-selenium.html)**Suite (Note : Suite)**



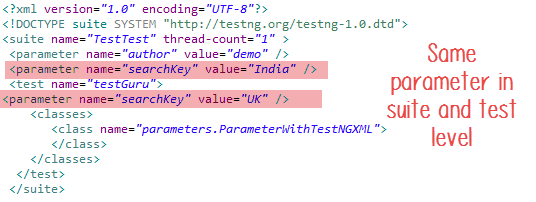
Now, parameters can be defined at 2 levels

1. Suite level – The parameters inside the <suite> tag of TestNG XML file will be a suite level parameter.
2. Test Level — The parameters inside the <Test> tag of testing XML file will be a Test level parameter.

Here is the same test with suite level parameters



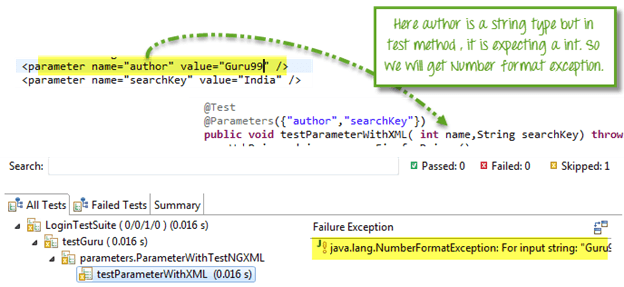
**NOTE:**In case if the parameter name is same in suite level and test level then test level parameter will get preference over suite level. So, in that case, all the classes inside that test level will share the overridden parameter, and other classes which are outside the test level will share suite level parameter.



### Troubleshooting

**Issue # 1**Parameter value in testng.xml cannot be typecasted to the corresponding test method’s parameter it will throw an error.

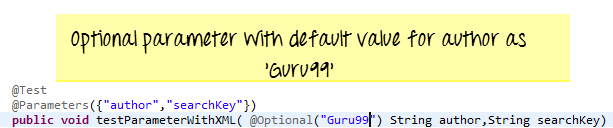
Consider the following example



Here, ‘author’ attribute is equal to ‘Guru99’ which is a string and in corresponding test method its expecting an integer value, so we will get an exception here.

**Issue # 2**Your @Parameters do not have a corresponding value in testing.xml.

You can solve this situation by adding **@optional** **annotation** in the corresponding parameter in the test method.



**Issue # 3:**You want to test multiple values of the same parameter using Testng.xml

The Simple answer is this can not be done! You can have multiple different parameters, but each parameter can only have a single value. This helps prevent hardcoding values into the script. This makes code reusable. Think of it as config files for your script. If you want to use multiple values for a parameter use DataProviders

## Data Provider in TestNG

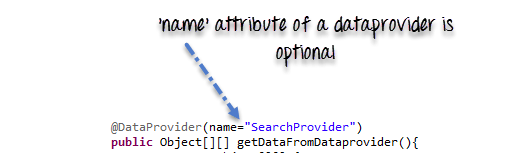
**Data Provider in TestNG** is a method used when a user needs to pass complex parameters. Complex Parameters need to be created from Java such as complex objects, objects from property files or from a database can be passed by the data provider method. The method is annotated by @DataProvider and it returns an array of objects.

## Parameters using Dataprovider

@Parameters annotation is easy but to test with multiple sets of data we need to use Data Provider.

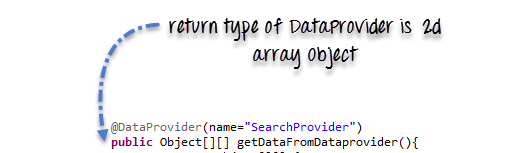
To fill thousand’s of web forms using our testing framework we need a different methodology which can give us a very large dataset in a single execution flow.

This data driven concept is achieved by **@DataProvider**annotation in TestNG.

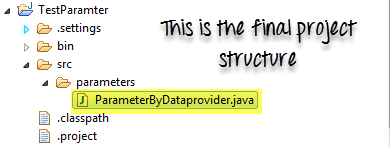


It has only one **attribute ‘name’**. If you do not specify the name attribute then the DataProvider’s name will be same as the corresponding method name.

Data provider returns **a two-dimensional JAVA object** to the test method and the test method, will invoke M times in a M\*N type of object array. For example, if the DataProvider returns an array of 2\*3 objects, the corresponding testcase will be invoked 2 times with 3 parameters each time.



**Complete Example**



package parameters;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.Assert;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

public class ParameterByDataprovider {

WebDriver driver;

String driverPath = "C:\\geckodriver.exe";

@BeforeTest

public void setup(){

//Create firefox driver object

System.setProperty("webdriver.gecko.driver", driverPath);

driver = new FirefoxDriver();

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

driver.get("https://google.com");

}

/\*\* Test case to verify google search box

\* @param author

\* @param searchKey

\* @throws InterruptedException

\*/

@Test(dataProvider="SearchProvider")

public void testMethod(String author,String searchKey) throws InterruptedException{

{

WebElement searchText = driver.findElement(By.name("q"));

//search value in google searchbox

searchText.sendKeys(searchKey);

System.out.println("Welcome ->"+author+" Your search key is->"+searchKey);

Thread.sleep(3000);

String testValue = searchText.getAttribute("value");

System.out.println(testValue +"::::"+searchKey);

searchText.clear();

//Verify if the value in google search box is correct

Assert.assertTrue(testValue.equalsIgnoreCase(searchKey));

}

}

/\*\*

\* @return Object[][] where first column contains 'author'

\* and second column contains 'searchKey'

\*/

@DataProvider(name="SearchProvider")

public Object[][] getDataFromDataprovider(){

return new Object[][]

{

{ "Guru99", "India" },

{ "Krishna", "UK" },

{ "Bhupesh", "USA" }

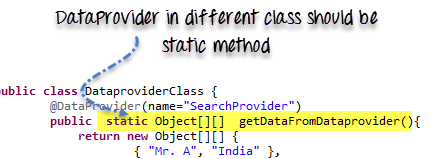
};

}

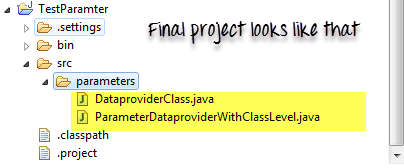
}

### Invoke DataProvider from different class

By default, DataProvider resides in the same class where test method is or its base class. To put it in some other class we need to make data provider method as static and in test method we need to add an attribute **dataProviderClass** in**@Test** annotation.



**Code Example**



**TestClass ParameterDataproviderWithClassLevel.java**

package parameters;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.Assert;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;

public class ParameterDataproviderWithClassLevel {

WebDriver driver;

String driverPath = "C:\\geckodriver.exe";

@BeforeTest

public void setup(){

System.setProperty("webdriver.gecko.driver", driverPath);

driver = new FirefoxDriver();

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

driver.get("https://google.com");

}

@Test(dataProvider="SearchProvider",dataProviderClass=DataproviderClass.class)

public void testMethod(String author,String searchKey) throws InterruptedException{

WebElement searchText = driver.findElement(By.name("q"));

//Search text in google text box

searchText.sendKeys(searchKey);

System.out.println("Welcome ->"+author+" Your search key is->"+searchKey);

Thread.sleep(3000);

//get text from search box

String testValue = searchText.getAttribute("value");

System.out.println(testValue +"::::"+searchKey);

searchText.clear();

//verify if search box has correct value

Assert.assertTrue(testValue.equalsIgnoreCase(searchKey));

}

}

**DataproviderClass.java**

package parameters;

import org.testng.annotations.DataProvider;

public class DataproviderClass {

@DataProvider(name="SearchProvider")

public static Object[][] getDataFromDataprovider(){

return new Object[][] {

{ "Guru99", "India" },

{ "Krishna", "UK" },

{ "Bhupesh", "USA" }

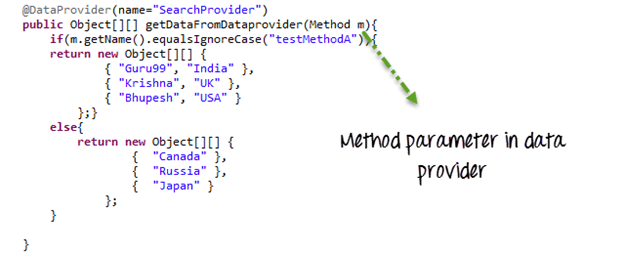
};

}}

### Types of Parameters in Dataprovider

There are two type of parameters supported by DataProvider method.

**Method**– If the **SAME** DataProvider should behave differently with different test method , use Method parameter.



In the following example ,

* We check if method name is testMethodA.
* If yes return one set of value
* Else return another set of value

package parameters;

import java.lang.reflect.Method;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.Assert;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

public class ParameterByMethodInDataprovider{

WebDriver driver;

String driverPath = "C:\\geckodriver.exe";

@BeforeTest

public void setup(){

System.setProperty("webdriver.gecko.driver", driverPath);

driver = new FirefoxDriver();

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

driver.get("https://google.com");

}

@Test(dataProvider="SearchProvider")

public void testMethodA(String author,String searchKey) throws InterruptedException{

WebElement searchText = driver.findElement(By.name("q"));

//Search text in search box

searchText.sendKeys(searchKey);

//Print author and search string

System.out.println("Welcome ->"+author+" Your search key is->"+searchKey);

Thread.sleep(3000);

String testValue = searchText.getAttribute("value");

System.out.println(testValue +"::::"+searchKey);

searchText.clear();

//Verify if google text box is showing correct value

Assert.assertTrue(testValue.equalsIgnoreCase(searchKey));

}

@Test(dataProvider="SearchProvider")

public void testMethodB(String searchKey) throws InterruptedException{

{

WebElement searchText = driver.findElement(By.name("q"));

//Search text in search box

searchText.sendKeys(searchKey);

//Print only search string

System.out.println("Welcome ->Unknown user Your search key is->"+searchKey);

Thread.sleep(3000);

String testValue = searchText.getAttribute("value");

System.out.println(testValue +"::::"+searchKey);

searchText.clear();

//Verify if google text box is showing correct value

Assert.assertTrue(testValue.equalsIgnoreCase(searchKey));

}

}

/\*\*

\* Here DataProvider returning value on the basis of test method name

\* @param m

\* @return

\*\*/

@DataProvider(name="SearchProvider")

public Object[][] getDataFromDataprovider(Method m){

if(m.getName().equalsIgnoreCase("testMethodA")){

return new Object[][] {

{ "Guru99", "India" },

{ "Krishna", "UK" },

{ "Bhupesh", "USA" }

};}

else{

return new Object[][] {

{ "Canada" },

{ "Russia" },

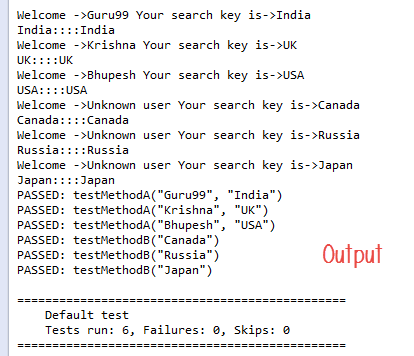
{ "Japan" }

};}

}

}

Here is the output



**ITestContext**– It can use to create different parameters for test cases based on groups.

In real-life, you can use ITestContext to vary parameter values based on Test Methods, hosts, configurations of the test.



In the following code example

* We have 2 groups A & B
* Each test method is assigned to a group
* If value of group is A, a particular data set is returned
* If value of group is B, another data set is returned

package parameters;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.Assert;

import org.testng.ITestContext;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

public class ParameterByITestContextInDataprovider {

WebDriver driver;

String driverPath = "C:\\geckodriver.exe";

@BeforeTest(groups={"A","B"})

public void setup(){

System.setProperty("webdriver.gecko.driver", driverPath);

driver = new FirefoxDriver();

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

driver.get("https://google.com");

}

@Test(dataProvider="SearchProvider",groups="A")

public void testMethodA(String author,String searchKey) throws InterruptedException{

{

//search google textbox

WebElement searchText = driver.findElement(By.name("q"));

//search a value on it

searchText.sendKeys(searchKey);

System.out.println("Welcome ->"+author+" Your search key is->"+searchKey);

Thread.sleep(3000);

String testValue = searchText.getAttribute("value");

System.out.println(testValue +"::::"+searchKey);

searchText.clear();

//verify correct value in searchbox

Assert.assertTrue(testValue.equalsIgnoreCase(searchKey));

}

}

@Test(dataProvider="SearchProvider",groups="B")

public void testMethodB(String searchKey) throws InterruptedException{

{

//find google search box

WebElement searchText = driver.findElement(By.name("q"));

//search a value on it

searchText.sendKeys(searchKey);

System.out.println("Welcome ->Unknown user Your search key is->"+searchKey);

Thread.sleep(3000);

String testValue = searchText.getAttribute("value");

System.out.println(testValue +"::::"+searchKey);

searchText.clear();

//verify correct value in searchbox

Assert.assertTrue(testValue.equalsIgnoreCase(searchKey));

}

}

/\*\*

\* Here the DAtaProvider will provide Object array on the basis on ITestContext

\* @param c

\* @return

\*/

@DataProvider(name="SearchProvider")

public Object[][] getDataFromDataprovider(ITestContext c){

Object[][] groupArray = null;

for (String group : c.getIncludedGroups()) {

if(group.equalsIgnoreCase("A")){

groupArray = new Object[][] {

{ "Guru99", "India" },

{ "Krishna", "UK" },

{ "Bhupesh", "USA" }

};

break;

}

else if(group.equalsIgnoreCase("B"))

{

groupArray = new Object[][] {

{ "Canada" },

{ "Russia" },

{ "Japan" }

};

}

break;

}

return groupArray;

}

}

Note: If you directly run your testng class, it will first call dataprovider which can’t get groups information as groups are not available. But instead if you call this class via testng.xml, it will have groups info available with ITestContext. Use the following XML to call the test

<!DOCTYPE suite SYSTEM "http://beust.com/testng/testng-1.0.dtd" >

<suite name="test-parameter">

<test name="example1">

<groups>

<run>

<include name="A" />

</run>

</groups>

<classes>

<class

name="parameters.ParameterByITestContextInDataprovider" />

</classes>

</test>

<test name="example2">

<groups>

<run>

<include name="B" />

</run>

</groups>

<classes>

<class

name="parameters.ParameterByITestContextInDataprovider" />

</classes>

</test>

</suite>

## Summary

* **Parameterization** is require to create **Data Driven Testing**.
* TestNG support two kinds of parameterization, using **@Parameter+TestNG.xml** and using**@DataProvider**
* In**@Parameter+TestNG.xml**parameters can be placed in suite level and test level. If

The Same parameter name is declared in both places; test level parameter will get preference over suit level parameter.

* using @Parameter+TestNG.xml only one value can be set at a time, but @DataProvider return **an 2d array of Object**.
* If DataProvider is present in the different class then the class where the test method resides,**DataProvider** should be **static method**.
* There are two parameters supported by **DataProvider** are **Method** and **ITestContext.**

# Different ways of Reading a text file in Java

There are multiple ways of writing and reading a text file. this is required while dealing with many applications. There are several ways to read a plain text file in Java e.g. you can use [FileReader](https://www.geeksforgeeks.org/file-handling-java-using-filewriter-filereader/), [BufferedReader](https://www.geeksforgeeks.org/java-io-bufferedreader-class-java/), or [Scanner](https://www.geeksforgeeks.org/scanner-class-in-java/) to read a text file. Every utility provides something special e.g. BufferedReader provides buffering of data for fast reading, and Scanner provides parsing ability.

**Methods:**

1. Using BufferedReader class
2. Using Scanner class
3. Using File Reader class
4. Reading the whole file in a List
5. Read a text file as String

*We can also use both BufferReader and Scanner to read a text file line by line in Java. Then Java SE 8 introduces another Stream class****java.util.stream.Stream****which provides a lazy and more efficient way to read a file.*

***Tip Note:****Practices of writing good code like flushing/closing streams, Exception-Handling etc, have been avoided for better understanding of codes by beginners as well.*

Let us discuss each of the above methods to a deeper depth and most importantly by implementing them via a clean java program.

**Method 1:** [Using BufferedReader class](https://www.geeksforgeeks.org/java-io-bufferedreader-class-java/)

This method reads text from a character-input stream. It does buffer for efficient reading of characters, arrays, and lines. The buffer size may be specified, or the default size may be used. The default is large enough for most purposes. In general, each read request made of a Reader causes a corresponding read request to be made of the underlying character or byte stream. It is therefore advisable to wrap a BufferedReader around any Reader whose read() operations may be costly, such as FileReaders and InputStreamReaders as shown below as follows:

BufferedReader in = new BufferedReader(Reader in, int size);

**Example:**

* Java

|  |
| --- |
| // Java Program to illustrate Reading from FileReader  // using BufferedReader Class    // Importing input output classes  **import** java.io.\*;    // Main class  **public** **class** GFG {        // main driver method  **public** **static** **void** main(String[] args) **throws** Exception      {            // File path is passed as parameter          File file = **new** File(              "C:\\Users\\pankaj\\Desktop\\test.txt");            // Note:  Double backquote is to avoid compiler          // interpret words          // like \test as \t (ie. as a escape sequence)            // Creating an object of BufferedReader class          BufferedReader br              = **new** BufferedReader(**new** FileReader(file));            // Declaring a string variable          String st;          // Condition holds true till          // there is character in a string  **while** ((st = br.readLine()) != **null**)                // Print the string              System.out.println(st);      }  } |

**Output:**

If you want to code refer to GeeksforGeeks

**Method 2:**[Using FileReader class](https://www.geeksforgeeks.org/difference-between-bufferedreader-and-filereader-in-java/)

Convenience class for reading character files. The constructors of this class assume that the default character encoding and the default byte-buffer size are appropriate.

Constructors defined in this class are as follows:

1. **FileReader(File file):** Creates a new FileReader, given the File to read from
2. **FileReader(FileDescriptor fd):** Creates a new FileReader, given the FileDescriptor to read from
3. **FileReader(String fileName):** Creates a new FileReader, given the name of the file to read from

**Example:**

* Java

|  |
| --- |
| // Java Program to Illustrate reading from  // FileReader using FileReader class    // Importing input output classes  **import** java.io.\*;    // Main class  // ReadingFromFile  **public** **class** GFG {        // Main driver method  **public** **static** **void** main(String[] args) **throws** Exception      {            // Passing the path to the file as a parameter          FileReader fr = **new** FileReader(              "C:\\Users\\pankaj\\Desktop\\test.txt");            // Declaring loop variable  **int** i;          // Holds true till there is nothing to read  **while** ((i = fr.read()) != -1)                // Print all the content of a file              System.out.print((**char**)i);      }  } |

**Output:**

If you want to code refer to GeeksforGeeks

**Method 3:**[Using Scanner class](https://www.geeksforgeeks.org/scanner-class-in-java/)

A simple text scanner that can parse primitive types and strings using regular expressions. A Scanner breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values of different types using the various next methods.

**Example 1:**With using loops

* Java

|  |
| --- |
| // Java Program to illustrate  // reading from Text File  // using Scanner Class  **import** java.io.File;  **import** java.util.Scanner;  **public** **class** ReadFromFileUsingScanner  {  **public** **static** **void** main(String[] args) **throws** Exception    {      // pass the path to the file as a parameter      File file = **new** File("C:\\Users\\pankaj\\Desktop\\test.txt");      Scanner sc = **new** Scanner(file);    **while** (sc.hasNextLine())        System.out.println(sc.nextLine());    }  } |

**Output:**

If you want to code refer to GeeksforGeeks

**Example 2:**Without using loops

* Java

|  |
| --- |
| // Java Program to illustrate reading from FileReader  // using Scanner Class reading entire File  // without using loop  **import** java.io.File;  **import** java.io.FileNotFoundException;  **import** java.util.Scanner;    **public** **class** ReadingEntireFileWithoutLoop  {  **public** **static** **void** main(String[] args)  **throws** FileNotFoundException    {      File file = **new** File("C:\\Users\\pankaj\\Desktop\\test.txt");      Scanner sc = **new** Scanner(file);        // we just need to use \\Z as delimiter      sc.useDelimiter("\\Z");        System.out.println(sc.next());    }  } |

**Output:**

If you want to code refer to GeeksforGeeks

**Method 4:**Reading the whole file in a List

Read all lines from a file. This method ensures that the file is closed when all bytes have been read or an I/O error, or other runtime exception, is thrown. Bytes from the file are decoded into characters using the specified charset.

**Syntax:**

public static List readAllLines(Path path,Charset cs)throws IOException

This method recognizes the following as line terminators:

\u000D followed by \u000A, CARRIAGE RETURN followed by LINE FEED

\u000A, LINE FEED

\u000D, CARRIAGE RETURN

**Example**

* Java

|  |
| --- |
| // Java program to illustrate reading data from file  // using nio.File  **import** java.util.\*;  **import** java.nio.charset.StandardCharsets;  **import** java.nio.file.\*;  **import** java.io.\*;  **public** **class** ReadFileIntoList  {  **public** **static** List<String> readFileInList(String fileName)    {        List<String> lines = Collections.emptyList();  **try**      {        lines =         Files.readAllLines(Paths.get(fileName), StandardCharsets.UTF\_8);      }    **catch** (IOException e)      {          // do something        e.printStackTrace();      }  **return** lines;    }  **public** **static** **void** main(String[] args)    {      List l = readFileInList("C:\\Users\\pankaj\\Desktop\\test.java");        Iterator<String> itr = l.iterator();  **while** (itr.hasNext())        System.out.println(itr.next());    }  } |

**Output:**

If you want to code refer to GeeksforGeeks

**Method 5:**Read a text file as String

**Example**

* Java

|  |
| --- |
| // Java Program to illustrate  // reading from text file  // as string in Java  **package** io;    **import** java.nio.file.\*;;    **public** **class** ReadTextAsString {    **public** **static** String readFileAsString(String fileName)**throws** Exception    {      String data = "";      data = **new** String(Files.readAllBytes(Paths.get(fileName)));  **return** data;    }    **public** **static** **void** main(String[] args) **throws** Exception    {      String data = readFileAsString("C:\\Users\\pankaj\\Desktop\\test.java");      System.out.println(data);    }  } |

**Output:**

If you want to code refer to GeeksforGeeks

## What are Hooks in Cucumber?

Cucumber supports ***hooks***, which are blocks of code that run ***before*** or ***after*** each scenario. You can define them anywhere in your project or step definition layers, using the methods ***@Before*** and ***@After***. ***Cucumber Hooks*** allows us to better manage the code workflow and helps us to reduce the code redundancy. We can say that it is an unseen step, which allows us to perform our scenarios or tests.

## Why Cucumber Hooks?

In the world of testing, you must have encountered the situations where you need to perform the prerequisite steps before testing any test scenario. This prerequisite can be anything from:

* Starting a webdriver
* Setting up DB connections
* Setting up test data
* Setting up browser cookies
* Navigating to certain page
* or anything before the test

In the same way, there are always after steps as well of the tests like:

* Killing the webdriver
* Closing DB connections
* Clearing the test data
* Clearing browser cookies
* Logging out from the application
* Printing reports or logs
* Taking screenshots on error
* or anything after the test

To handle these kinds of situations, cucumber hooks are the best choice to use. Unlike [***TestNG Annotations***](https://toolsqa.com/testng/testng-annotations/), cucumber supports only two hooks (Before & After) which works at the start and the end of the test scenario. As the name suggests, @before hook gets executed well before any other test scenario, and @after hook gets executed after executing the scenario.

## How to implement Hooks in Cucumber Test

Let's do some easy and small examples of Cucumber Hooks just to understand the concept. I will bring the intelligent usage of Hooks in my later tutorial series of ***Designing Framework with Cucumber***.

### ***Test Hooks with Single Scenario***

***Feature File***

Feature: Test Hooks

Scenario: This scenario is to test hooks functionality

Given this is the first step

When this is the second step

Then this is the third step

***Step Definitions***

package stepDefinition;

import cucumber.api.java.en.Given;

import cucumber.api.java.en.Then;

import cucumber.api.java.en.When;

public class Hooks\_Steps {

@Given("^this is the first step$")

public void This\_Is\_The\_First\_Step(){

System.out.println("This is the first step");

}

@When("^this is the second step$")

public void This\_Is\_The\_Second\_Step(){

System.out.println("This is the second step");

}

@Then("^this is the third step$")

public void This\_Is\_The\_Third\_Step(){

System.out.println("This is the third step");

}

}

***Note***: There is no logic used in the step definitions. Just printing the step summary log.

***Hooks***

package utilities;

import cucumber.api.java.After;

import cucumber.api.java.Before;

public class Hooks {

@Before

public void beforeScenario(){

System.out.println("This will run before the Scenario");

}

@After

public void afterScenario(){

System.out.println("This will run after the Scenario");

}

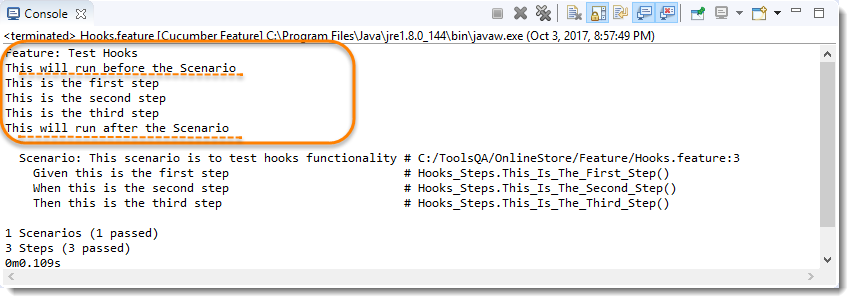
}

***Things to note***

* An important thing to note about the after hook is that even in case of test fail, after hook will execute for sure.
* Method name can be anything, need not to be beforeScenario() or afterScenario(). can also be named as setUp() and tearDown().
* \*Make sure that the package import statement should be ***import cucumber.api.java.After; & import cucumber.api.java.Before;***

Often people mistaken and import Junit Annotations, so be careful with this.

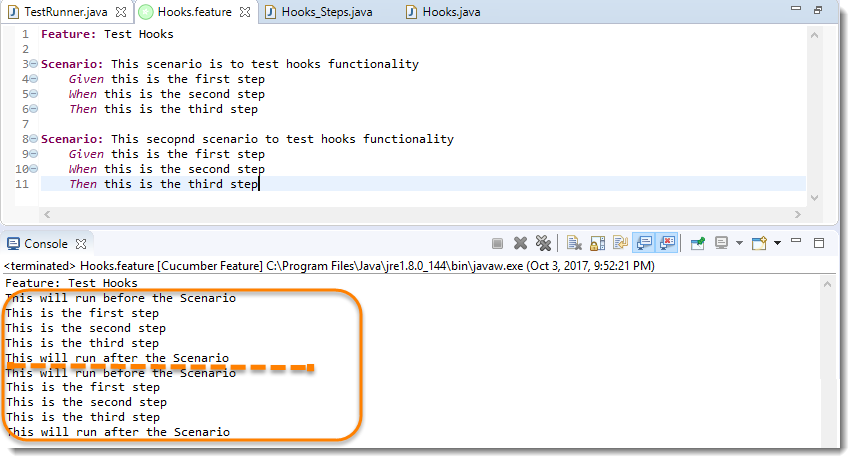
***Output***



No need for explanation, it is self-explanatory :)

### ***Test Hooks with Multiple Scenarios***

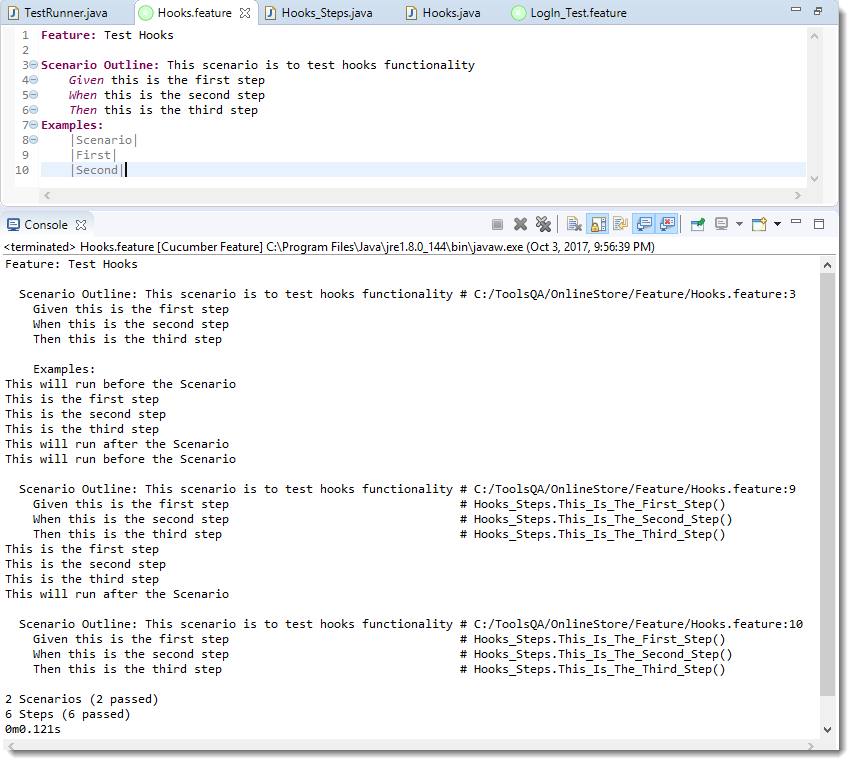
I just wanted to show you the reaction of Hooks with the multiple scenarios. Let's just add one more Test Scenario in the feature file and run the feature again.



***Note***: Scenario Hooks execute before and after every scenario. In the above example, executed two times for two scenarios.

### ***Test Hooks with Example Scenarios***

Lets take a look when we have Scenario Outline with Examples.



***Note***: Again, in cucumber, every example is considered as a separate scenario. So the output is the same as the second example above.

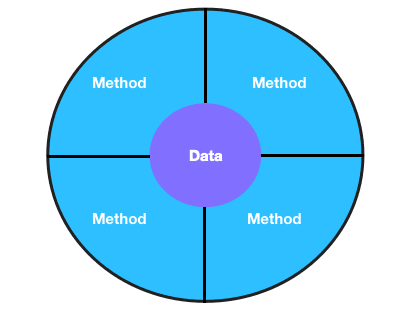
## Definition of OOPS Concepts in Java

So we can define OOP programming as:

“Object-oriented programming is an approach that modularizes programs by creating a partitioned memory area for both functions and data that can be used as templates for creating copies of such modules on demand.”

## OOPS Paradigm

The primary objective of the object-oriented approach is to eliminate some of the pitfalls that exist in the [procedural approach](https://hackr.io/blog/procedural-programming). OOP treats data as an element in the program, not allowing it to flow around the system freely. It ties data closely to the functions that operate on it and protects it from unintentional modification by other existing functions. OOPS allows decomposing a problem into several entities called Objects and then build data and functions from these entities. The combination of the data makes up an object.



Object = Method + Data

The data of an object is accessed by the methods associated with that object. However, the methods of an object can access methods of other objects.

## Features of OOPS

Some features of object-oriented programming in java are:

* Emphasis is on data than procedures
* Programs are divided into objects
* Data Structures are designed to characterize objects.
* Methods operating on the data of an object are tied together in the data structure.
* Data is hidden, and external functions cannot access it.
* Objects communicate with each other through methods
* New methods and data can be easily added whenever necessary
* Follows the bottom-up approach in program design

## List of OOPS Concepts in Java with Examples

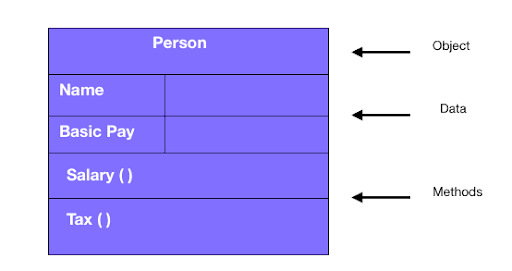
General OOPS concepts in Java are:

### **Objects and Classes**

Objects are runtime entities in an object-oriented system. An object can represent a person, a bank account, a place, a table of data. It may also represent user-defined data types like lists and vectors. Any programming problem is analyzed based on objects and how they communicate amongst themselves. The objects interact with each other by sending messages to one another when a program is executed. For Example, ‘customer’ and ‘account’ are two objects that may send a message to the account object requesting for the balance. Each object contains code and data to manipulate the data. Objects can even interact without knowing the details of each other’s code or data.

The entire set of code and data of an object can be made user-defined data type using the concept of the class. A class is a ‘data-type’ and an object as a ‘variable’ of that type. Any number of objects can be created after a class is created.

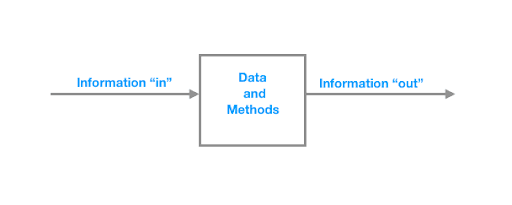
The collection of objects of similar types is termed as a class. For Example, apple, orange, and mango are the objects of the class Fruit. Classes behave like built-in data types of a [programming language](https://hackr.io/blog/what-is-programming-language) but are user-defined data types.



Representation of an Object

### **Data Abstraction and Encapsulation**

The wrapping up of the data and methods into the single unit is known as encapsulation. The data is accessible only to those methods, which are wrapped in the class, and not to the outside world. This insulation of data from the direct access of the program is called data hiding. Encapsulation of the object makes it possible for the objects to be treated like ‘black boxes’ that perform a specific task without any concern for internal implementation.



Encapsulation- Objects as “black-boxes”

Abstraction is the act of reducing programming complexity by representing essential features without including the background explanations or details. Classes are the concept of abstraction and are defined as the list of abstract attributes such as size, weight, cost, and methods that operate on these attributes. Classes wrap or encapsulate all the essential properties of the objects that are to be created.

#### **Abstract classes and Abstract methods:**

1. An abstract class is a class with an abstract keyword.
2. An abstract method is a method declared without a method body.
3. An abstract class may not have all the abstract methods. Some methods are concrete.
4. A method defined abstract must have its implementation in the derived class, thus making method overriding compulsory. Making a subclass abstract avoids overriding.
5. Classes that contain abstract method(s) must be declared with abstract keyword.
6. The object for an abstract class cannot be instantiated with the new operator.
7. An abstract class can have parameterized constructors.

#### **Ways to achieve abstraction:**

* Using abstract keyword
* Using interfaces

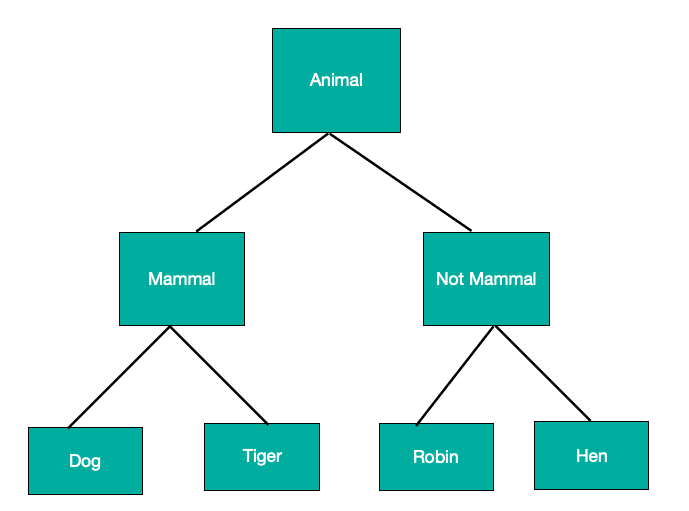
#### **Example Code:**

The code below shows an example of abstraction.

**abstract** **class** **Car**  
{  
 Car()  
 {  
  System.**out**.println("Car is built. ");  
 }  
 **abstract** **void** **drive**();  
 **void** **gearChange**()  
 {  
  System.**out**.println("Gearchanged!!");  
 }  
}   
  
**class** **Tesla** **extends** **Car**  
 {  
  **void** **drive**()  
  {  
   System.**out**.println("Drive Safely");  
  }  
 }  
  
**class** **Abstraction**   
 {  
  **public** **static** **void** **main** (String args[])  
  {  
   Car obj = **new** Tesla();  
   obj.drive();  
   obj. gearChange();  
  }  
 }

### **Inheritance**

Inheritance is the process by which objects of one class acquire some properties of objects of another class. Inheritance supports the concept of hierarchical classification. For Example, a bird Robin is part of the class, not a mammal, which is again a part of the class Animal. The principle behind this division is that each subclass shares common characteristics from the class from its parent class.



Properties of Inheritance

In OOP, the idea of inheritance provides the concept of reusability. It means that we can add additional features to parent class without modification; this is possible by deriving a new class from the parent class. The new class consists of the combined features from both the classes. In Java, the derived class is also known as the subclass.

#### **Types of Inheritance**

* Single
* Multiple
* Multilevel
* Hybrid

#### **Example Code:**

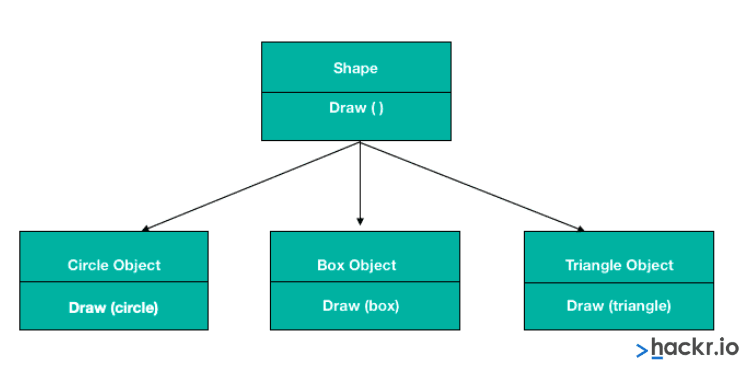
The code below illustrates an example of Inheritance.

**class** **Animal**   
{  
 **void** **eat**()  
 {  
  System.out.println("I am a omnivorous!! ");  
 }  
}  
  
**class** **Mammal** **extends** **Animal**   
{  
 **void** **nature**()  
 {  
  System.out.println("I am a mammal!! ");  
 }  
}  
  
**class** **Dog** **extends** **Mammal**   
{  
 **void** **sound**()  
 {  
  System.out.println("I bark!! ");  
 }  
}  
  
**class** **Inheritance**   
{  
 **public** **static** **void** **main**(String args[])  
 {  
  Dog d = **new** Dog();  
  d.eat();  
  d.nature();  
  d.sound();  
 }  
}

### **Polymorphism**

Polymorphism is an important OOP concept; it means the ability to take many forms. For Example, an operation exhibits different behavior in different situations. The behavior depends on the type of data used in operation. For Example, in the operation of addition, the operation generates a sum for two numbers. If the operands are strings, then a third-string is produced by the operation by concatenation.

The figure below demonstrates that a single function name can be used to handle the different numbers and different types of arguments.



In polymorphism, objects having different internal structures can share the same external interface; it means that a class of operation may be accessed in the same manner even though actions with each operation may differ. Inheritance extensively uses the concept of polymorphism.

Polymorphism can be achieved in two ways:

#### **Method Overloading**

It is possible to create methods with the same name but different parameter lists and different definitions. This is called method overloading. Method overloading is required when objects are required to perform similar tasks but using different input parameters. When we call a method in an object, Java matches up the method name first and then the number and type of parameters to decide which definition to execute.

Method overloading is achieved in three ways:

|  |  |
| --- | --- |
| **On the Basis of** | **Example** |
| **Number of Parameters** | **times**(**int**, **int**) **times**(**int**, **int**, **int**) |
| **Data Types of Parameters** | **times**(**int**, **int**) **times**(**int**, float) |
| **The Sequence of Data Types of Parameters** | times(int, float) times(float, int) |

#### **Example Code:**

The code below demonstrates the concept of method overloading.

**class** **CircleArea**   
{  
 **double** **area**(**double** x)  
 {  
  **return** 3.14 \* x;  
 }  
}  
  
**class** **SquareArea**   
{  
 **int** **area**(**int** x)  
 {  
  **return** x \* x;  
 }  
}  
  
**class** **RectangleArea**   
{  
 **int** **area**(**int** x, **int** y)  
 {  
  **return** x \* y;  
 }  
}  
  
**class** **TriangleArea**   
{  
 **int** **area**(**int** y, **int** x)  
 {  
  **return** (y \* x)/2;  
 }  
}  
  
**class** **Overloading**   
{  
 **public** **static** **void** **main**(String args[])  
 {  
  CircleArea ca = **new** CircleArea();  
  SquareArea sa = **new** SquareArea();  
  RectangleArea ra = **new** RectangleArea();  
  TriangleArea ta = **new** TriangleArea();  
  
  System.**out**.println("Circle area = "+ ca.area(1));  
  System.**out**.println("Square area = "+ sa.area(2));  
  System.**out**.println("Rectangle area = "+ ra.area(3,4));  
  System.**out**.println("Triangle area = "+ ta.area(6,3));  
 }  
}

#### **Method Overriding**

A method defined in the superclass is inherited by its subclass and is used by the objects created by the subclass. However, there may be occasions when an object should respond to the same method but behave differently when that method is called, which means a method defined in the superclass is overridden. Overriding is achieved by defining a method in the subclass that has the same name, the same arguments, and the same return type as a method in the superclass. So, when the method is called, the method defined in the subclass invoked and executed instead of the one in the superclass.

#### **Example Code:**

The code below demonstrates the concept of method overriding.

**class** **Shape**   
{  
 **void** **draw**()  
 {  
  System.**out**.println("Mention shape here");  
 }  
  
 **void**  **numberOfSides**()  
 {  
  System.**out**.println("side = 0");  
 }  
}  
  
**class** **Circle** **extends** **Shape**   
{  
 **void** **draw**()  
 {  
  System.**out**.println("CIRCLE ");  
 }  
  
 **void** **numberOfSides**()  
 {  
  System.**out**.println("side = 0 ");   
 }  
}  
  
**class** **Box** **extends** **Shape**   
{  
 **void** **draw**()  
 {  
  System.**out**.println("BOX ");  
 }  
  
 **void** **numberOfSides**()  
 {  
  System.**out**.println("side= 6");   
 }  
}  
  
**class** **Triangle** **extends** **Shape**   
{  
 **void** **draw**()  
 {  
  System.**out**.println("TRIANGLE ");  
 }  
  
 **void** **numberOfSides**()  
 {  
  System.**out**.println("side = 3 ");  
 }  
}  
  
**class** **Overriding**   
{  
 **public** **static** **void** **main** (String args[])  
 {  
  Circle c = **new** Circle();  
  c.draw();  
  c.numberOfSides();  
  
  Box b = **new** Box();  
  b.draw();  
  b.numberOfSides();  
  
  Triangle t = **new** Triangle();  
  t.draw();  
  t.numberOfSides();  
 }  
}

### **Dynamic Binding**

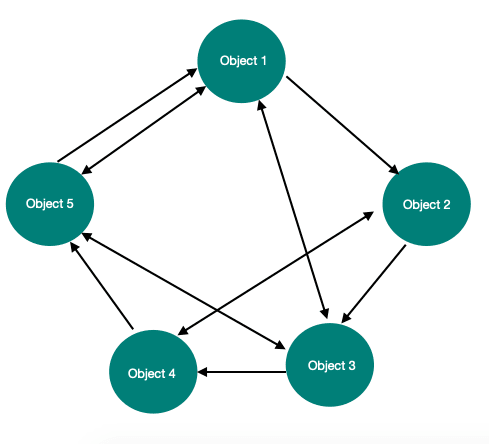
Binding is the process of linking a procedure call to the code to be executed in response to the call. It means that the code associated with the given procedure call is not known until the time of the call at runtime. It is associated with inheritance and polymorphism.

**Message Communication**

Objects communicate with each other in OOPs The process of programming in case of OOP consists of the following:

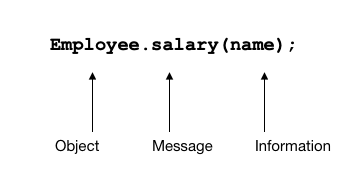
* Creating classes defining objects and their behavior.
* Creating objects
* Establishing communication between objects.

The network of Objects Communicating with Each Other



Specifying the object name, the method name, and the information to be sent is involved in message passing.

**For Example, consider the statement.**

****

Objects can be created or destroyed as they have a life cycle. It allows communication between the objects until the objects are alive.

## ****Benefits of OOPs Concept in Java****

* Inheritance eliminates redundant code and enables reusability.
* As Message passing allows communication with objects, this presents writing code from scratch every time. It is thus saving development time and higher productivity.
* Partitions work in a project based on classes and objects.
* Systems up-gradation is easy.

## ****Applications of OOPs Concept in Java****

* Real-time systems
* Simulation and modeling
* Object-oriented databases
* Hypertext and Hypermedia
* AI and expert systems
* Neural networks and parallel programming
* Automation systems

## ****Summary****

Java is a robust and scalable object-oriented programming language that is based on the concept of objects and classes. It offers features like inheritance, abstraction, encapsulation, and polymorphism for developing an efficient and reliable code.

**People are also reading:**

* [Best Java Courses](https://hackr.io/blog/best-java-courses)
* [Top 10 Java Certifications](https://hackr.io/blog/java-certification-courses)
* [Best Java Books](https://hackr.io/blog/best-java-books-for-beginners-and-advanced-programmers)
* [Best Java Projects](https://hackr.io/blog/java-projects)
* [Top Java Programming Interview Questions](https://hackr.io/blog/java-interview-questions)
* [Core Java Cheatsheet - Introduction to Programming in Java](https://hackr.io/blog/java-cheat-sheet)
* [Difference between Java vs Javascript](https://hackr.io/blog/java-vs-javascript)
* [Top 10 Java Frameworks](https://hackr.io/blog/java-frameworks)
* [Best Way to Learn Java](https://hackr.io/blog/best-way-to-learn-java)
* [Constructor in java](https://hackr.io/blog/java-constructor)
* [Prime Number Program in Java](https://hackr.io/blog/prime-number-program-in-java)
* [What is Java?](https://hackr.io/blog/what-is-java)
* [Features of Java](https://hackr.io/blog/features-of-java)