**`What is Java ?**

Java is high level Programming language developed by sun microsystems. It is platform independent , so any device equipped with JVM can run the java code.

**Main Features of Java**

* Platform – independent
* Object Oriented
* Multithreading
* Robust and secure

**What is Wrapper Class in Java ?**

Wrapper Class will convert our Primitive data type into objects. Like Integer , Float Boolean double etc..

**Difference between Primitive data type and refrence data type ?**

Primitive data type will store the value as it is

Reference data type will store the memory address of that value.

**Difference between throw and throws ?**

Throw will used to manually throw an expection inside the method.

Throws it just declares the expection that method might throw

**What is Exception ?**

Exception is Error that will break our program flow that can cause program to crash , We have to handle this exception otherwise our execution will be stoped , We will handle this in try / catch concept or throws keyword

Try{

}catch(){  
}

Finally{

}

**Type of Exception**

Checked Exception => Its Predicitable Error that can occur in our code ,Usually this will check by the complier itself and it will throw complication error if we are not handling in the code.

IOExecption,Interrupted Execption , ParseExecption

Unchecked Expection : It’s a Error that will occur during execution of our program

Arthimetic expection , ArrayIndexOutOfBound Expection , NullPointerExpection

**Final Keyword**

We will use final keyword in three place , Which is Variable , Method ,class

Variable : it wont allow reassignment

Method : it wont allow child class to Override the Method

Class : it wont allow other class to inherit this class

**Finalize**

Finalize is protected Method in the Object class , basically garbage collector will call this method before removing the un refrenced object in the memory

**What is package**

A package is set of class, subclass , interface that are similar

* Bulit in package
* User defined package

**What is OOPS in Java ?**

Object Oriented Programming language in java is programming models that used Object to represent and manipulate data. It based on four priniciple

Encapsulation

Abstraction

Inheritance

Polymorphism

**What is Class ?**

A class is blueprint or template for the object

**What is Object ?**

An Object is the instance of a Class.

**What is Encapsulation ?**

Encapsulation means restricting direct access to data by making Attribute private and it can be accessed through getter and setter method

**What is Abstraction ?**

Abstraction hides implementation details and expose only essential features of an object. It can achieved using Abstract class or interfaces

**What is Abstract Class ?**

A abstract class in java is a class that cannot be instantiated. It serve as blueprint for other classes by providing abstract Method and concerte method

* When classes share common behaviour , But some methods needs to be implemented differently, then we can use abstract

**What is Interface?**

An interface in java is blueprint of a class that contains only Abstract Method , But After Java 8 it can also contains default and static method.

We can use interface to achieve abstraction & Multiple Inheritance

**Difference between Abstract and interface ?**

| **Feature** | **Interface** | **Abstract Class** |
| --- | --- | --- |
| Methods | Only **abstract methods** (before Java 8); **default & static methods** (from Java 8) | Can have both **abstract and concrete methods** |
| Fields | Only **public static final** (constants) | Can have **instance variables** |
| Constructor | Cannot have constructors | Can have constructors |
| Access Modifiers | Methods are **public** by default | Methods can be **private, protected, or public** |
| Multiple Inheritance | Supports multiple inheritance | Supports single inheritance |

**What is Polymorphism ?**

Polymorphism enables methods to behave differently based on the object that call them

Compile – time Polymorphism or Method Overloading

Multiple methods in the same class with same name but with different parameter

Run-time polymorphism or Method Overriding

A subclass provides specific implementation of a method that is already defined in its super-class

**Differences Between Method Overloading & Overriding**

| **Feature** | **Method Overloading** | **Method Overriding** |  |
| --- | --- | --- | --- |
| **Definition** | Multiple methods with the same name but different parameters in the same class | A subclass provides a new implementation of a method from the parent class |  |
| **Determined at** | **Compile-time** | **Runtime** |  |
| **Return Type** | Can be the same or different | Must be the **same or covariant** |  |
| **Parameters** | Must be **different** (number, type, or order) | Must be **exactly the same** |  |
| **Access Modifier** | No restrictions | Cannot reduce the visibility of the method |  |
| **Static Methods** | Can be overloaded | Cannot be overridden |  |

**What is Inhertaince ?**

Inhertanice is OOPS concept where one class inherties the properties and behaviours of another class. It promotes code reusability, hierarchical releationship

**Types of Inheritance in Java**

| **Type** | **Description** |
| --- | --- |
| **Single Inheritance** | A subclass inherits from one superclass. |
| **Multilevel Inheritance** | A subclass inherits from another subclass. |
| **Hierarchical Inheritance** | Multiple subclasses inherit from the same superclass. |
| **Multiple Inheritance (via Interfaces)** | A class implements multiple interfaces (Java does not support multiple class inheritance). |

## ****Why Java Does Not Support Multiple Class Inheritance?****

* **Java does not allow a class to inherit from multiple classes (class A extends B, C).**  
  **Reason:** To avoid the **Diamond Problem**, where the compiler cannot decide which parent method to inherit if both have the same method name.

**What is Super keyword ?**

Super is used to the refer the immediate parent class

We can use super keyword to call parent class constructor , Methods and attributes

**What is this keyword ?**

This is the reference variable that used to refer current object.

**What is Access Modifier in Java ?**

In Java access modifier is used to control the visibility and accessibility of classes, Variables and Methods.

**Types of Access Modifiers in Java**

| **Modifier** | **Class** | **Same Package** | **Different Package (Subclass)** | **Different Package (Non-Subclass)** |
| --- | --- | --- | --- | --- |
| public | ✅ Yes | ✅ Yes | ✅ Yes | ✅ Yes |
| protected | ✅ Yes | ✅ Yes | ✅ Yes | ❌ No (only via inheritance) |
| default *(no modifier)* | ✅ Yes | ✅ Yes | ❌ No | ❌ No |
| private | ✅ Yes | ❌ No | ❌ No | ❌ No |

**Selenium**

**What is Selenium ?**

🔹 Selenium is an **open-source automation tool** used for **web application testing**. It supports multiple languages (Java, Python, C#) and browsers (Chrome, Firefox, Edge).  
🔹 **Components of Selenium:**

* **Selenium WebDriver** – Automates browser actions.
* **Selenium Grid** – Runs tests in parallel across multiple machines.
* **Selenium IDE** – A record and playback tool.

**What is Selenium architecture** ?

Selenium follows client – server architecture and its made up of four major components

**Selenium client libarays** : Its provides support to multiple libraries such as Ruby, Python, Java, etc as language bindings

**JSON Wire Protocal** : Selenium Webdriver communicates with browser using JSON Wire protocal

**Browser Drivers** : Each browser has its own brower-specific driver like chromedriver edgedriver and its used to communicate with browser

**Browser** : We have chrome , Firefox ,edge , Safari brower so our web driver sends commands to the brower via browser driver

**What is RemoteWebDriver ?**

**RemoteWebDriver** is class in selenium that enables test execution on a **remote machine** instead of local machine

It is used for **cross-browser, cross-platform, and parallel execution** by communicating with a **Selenium Grid Hub**

**What is DesiredCapabilities ?**

DesiredCapabilities is a **class in Selenium** (deprecated in Selenium 4) that allows us to **set properties** for the WebDriver session before starting the browser.  
 It is used to define the **browser name, version, platform, and other settings** when working with **RemoteWebDriver** (Selenium Grid, cloud services).

**What is ChromeOptions ?**

ChromeOptions is a **class in Selenium 4** used to configure **Google Chrome** settings before launching the browser. It **replaces** DesiredCapabilities (which is deprecated) for setting browser preferences, enabling headless mode, browser size and opening in maximize mode

op.addArguments("--start-maximized");  
op.addArguments("--window-size=1300,1300"); options.addArguments("--headless")

**What is Webdriver ?**

WebDriver is an **interface** in Selenium that provides methods to interact with web browsers.

**What is difference between get and navigate().to() ?**

Both **get(String URL)** and **navigate().to(String URL)** are used to open a web page in Selenium WebDriver, but there are some key differences.

* Get will wait for entire page to load
* Navigate to wont wait for entire page to load and also its supports forward and backword operation as well

**What are the different locators in Selenium?**

**Locators in Selenium** help identify web elements:

* **ID**: driver.findElement(By.id("username"))
* **Name**: driver.findElement(By.name("password"))
* **Class Name**: driver.findElement(By.className("btn-login"))
* **Tag Name**: driver.findElement(By.tagName("button"))
* **Link Text**: driver.findElement(By.linkText("Sign Up"))
* **Partial Link Text**: driver.findElement(By.partialLinkText("Sign"))
* **CSS Selector**: driver.findElement(By.cssSelector("#loginButton"))
* **XPath**: driver.findElement(By.xpath("//input[@name='email']"))

**What is Xpath ?**

**Xpath** is used to identify the web elements in the DOM (It is used to represent the webpage as tree structure of nodes)

XPath is of two types:

| **Type** | **Description** | **Example** |
| --- | --- | --- |
| **1. Absolute Xpath** | Starts from the root (/html) and follows a direct path | /html/body/div[1]/div[2]/button |
| **2. Relative Xpath** | Finds elements anywhere in the DOM, making it more flexible | //button[@id=’login’] |

**What is Dyanmic Xpath ?**

* **Dynamic Xpath** is used to locate elements whose attributes (such as id, class, name, etc.) **change dynamically** (i.e., change every time the page reloads).
* It helps find elements that **do not have fixed attributes** by using **functions like contains(), starts-with(), text(), and Xpath axes**.

//input[contains(@id, ‘username’)]

//button[starts-with(@id, ‘submit’)]

**List of Xpath Axes in Selenium**

| **Axis** | **Description** | **Example** |
| --- | --- | --- |
| **ancestor** | Selects all ancestor (parent, grandparent, etc.) elements | //button[@id=’submit’]/ancestor::div |
| **ancestor-or-self** | Selects the element itself and all its ancestors | //button[@id=’submit’]/ancestor-or-self::div |
| **child** | Selects direct child elements of a node | //div[@class=’container’]/child::input |
| **descendant** | Selects all children and their sub-elements | //div[@class=’form’]/descendant::input |
| **descendant-or-self** | Selects the element itself and all its descendants | //div[@class=’box’]/descendant-or-self::p |
| **following** | Selects everything after the current node in the DOM | //h2[text()=’Welcome’]/following::p |
| **following-sibling** | Selects only siblings after the current node | //h2[text()=’Welcome’]/following-sibling::p |
| **preceding** | Selects everything before the current node in the DOM | //p[@id=’description’]/preceding::h3 |
| **preceding-sibling** | Selects only siblings before the current node | //p[@id=’description’]/preceding-sibling::h3 |
| **parent** | Selects the parent of the current node | //span[@id=’child’]/parent::div |
| **self** | Selects the current node itself | //h1[@class=’title’]/self::h1 |
|  |  |  |

**What is implicit wait ?**

Implicit wait is a timeout setting in the selenium that will instruct the Webdriver to wait for certain amount of time before throwing a NoSuchElementExpection , When trying to locate the element.

We will declare this globally and it will applies for all the webelements in the script

driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));

**What is Explicit Wait ?**

**Explicit Wait** is a wait mechanism in Selenium WebDriver that waits **until a specific condition is met** before proceeding with execution.

Unlike **Implicit Wait**, which applies globally, **Explicit Wait** is used for **specific elements** that take time to appear, become clickable, or become visible.

**WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));**

**WebElement element = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("login")));**

**What is Fluent Wait ?**

Fluent wait will wait for Specific condition to met before proceeding execution and We have to set polling frequency so that it will check the web elements at interval and also we can ignore specific expection.

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

.withTimeout(Duration.ofSeconds(20)) // Max wait time

.pollingEvery(Duration.ofSeconds(2)) // Check every 2 seconds

.ignoring(NoSuchElementException.class); // Ignore NoSuchElementException

WebElement element = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("login")));

wait.until(Function)

FluentWait<WebDriver> wait = new FluentWait<WebDriver>(driver).  
 withTimeout(Duration.*ofSeconds*(10)).  
 pollingEvery(Duration.*ofSeconds*(2)).  
 ignoring(StaleElementReferenceException.class);  
Function<WebDriver, WebElement> WebElement = (driver->driver.findElement(By.*id*("name")));  
wait.until(WebElement);

**Differecne between FindElement and FindElements ?**

Find Element will return first Matching Web Element , if there is no match it will throw NoSuchElement Expection.

FindElements will return all matching Web Element in List<WebElement>. If there is no match it will return empty list

**How will handle different Windows in Selenium ?**

Each browser window has a **unique identifier (Window Handle)** assigned to it. Selenium uses:

* driver.getWindowHandle() → Returns **current window handle** (String).
* driver .getWindowHandles() → Returns **set of all open window handles** (Set<String>).
* driver .switchTo().window(handle) → Switches control to the specified window.

**How will handle different Frames in selenium ?**

First we have find to the iframe webelement then we can switch to that frame

driver.switchTo().frame(int index)

driver.switchTo().frame(String id or name)

driver.switchTo().frame(WebElement ele)

driver.switchTo().parentFrame() => It will switch to first parent frame

driver.switchTo().defaultContent() => It will swich to Main document

**How do you handle alert ?**

Alert alert = driver.switchTo().alert();

alert.getText();

alert.accept();

alert.dismiss();

ExpectedConditions.alertIsPresent()

NoAlertPresentException

## ****Actions Class in Selenium****

The Actions class in Selenium is used to perform advanced user interactions like:  
✅ Mouse hover  
✅ Right-click (context click)  
✅ Double-click  
✅ Drag and drop  
✅ Click and hold  
✅ Keyboard actions

Actions act = new Actions(driver);

act.moveToElement(WebElement).perform();

act.sendKeys(WebElement , CharSequece).perform();

actions.doubleClick(element).perform();

actions.dragAndDrop(source, target).perform();

actions.clickAndHold(element).release().perform();

actions.sendKeys(Keys.ENTER).perform();

actions.contextClick(WebElement).perform();

actions.keyDown(Keys.CONTROL).sendKeys("a").keyUp(Keys.CONTROL).perform(); // Press CTRL + A

actions.keyDown(Keys.SHIFT).sendKeys("selenium").keyUp(Keys.SHIFT).perform(); // Types "SELENIUM"

**What is JavaScriptExecutor ?**

JavaScriptExecutor is an interface in Selenium that allows executing **JavaScript code** within the browser. It is useful when Selenium WebDriver **fails to interact** with certain elements (e.g., hidden elements, scrolling issues, or dynamic elements).

js.executeScript("arguments[0].click();",driver.findElement(By.*xpath*("//a[text()='Inputs']")));  
js.executeScript("arguments[0].value = '12';",driver.findElement(By.*tagName*("input")));  
js.executeScript("arguments[0].scrollIntoView(true);",driver.findElement(By.*id*("page-footer")));

js.executeScript("this.scrollTo(0,this.outerHeight);");

js.executeScript("history.go(0);");  
js.executeScript("history.back();");  
js.executeScript("history.forward();");

js.executeScript("return document.title;")

js.executeScript("window.resizeTo(1024, 768);");

js.executeScript("alert('Hello from Selenium!');");

**getCssValue() in Selenium WebDriver ?**

The getCssValue() method in Selenium is used to **retrieve the CSS property value** of a web element.

String cssValue = element.getCssValue("propertyName");

 **element** → The WebElement whose CSS property you want to retrieve.

 **propertyName** → The name of the CSS property (e.g., "color", "font-size", "background-color").

 **Returns:** A String representing the value of the CSS property.

WebElement element = driver.findElement(By.id("myElement"));

String color = element.getCssValue("color");

Color.*fromString*(colour).asHex()

String bgColor = element.getCssValue("background-color");  
String fontSize = element.getCssValue("font-size");  
String fontFamily = element.getCssValue("font-family");  
String border = element.getCssValue("border");

**How to Handle dropdown in selenium ?**

Using Select class we can handle dropdowns

WebElement dropdown = driver.findElement(By.id("country"));

Select select = new Select(dropdown);

| **Method** | **Description** |
| --- | --- |
| selectByVisibleText("Option") | Selects an option by its visible text. |
| selectByValue("value") | Selects an option by its value attribute. |
| selectByIndex(index) | Selects an option based on its index (starting from 0). |
| getFirstSelectedOption() | Gets the currently selected option. |
| getOptions() | Gets all options available in the dropdown. |
| getAllSelectedOptions() | Gets all selected options (for multi-select dropdowns). |
| deselectAll() | Deselects all selected options (for multi-select dropdowns). |
| deselectByVisibleText("Option") | Deselects an option by its visible text. |

**How to Handle dynamic dropdowns in selenium ?**

So Dyanmic dropdowns usually wont have <select> tag , instead of that it will have div , ul or li tags..First we have to find how to trigger the dynamic event and then we have to use Explicit wait to wait for the element to visible & clickable and then we have to select

WebDriver driver = new ChromeDriver();

WebElement searchBox = driver.findElement(By.id("city-search"));

searchBox.sendKeys("New"); // Typing to trigger dynamic results

WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));

List<WebElement> options = wait.until(ExpectedConditions.visibilityOfAllElementsLocatedBy(By.xpath("//ul[@id='city-list']/li")));

**Difference Between quit() and close() in Selenium WebDriver**

| **Method** | **Description** | **Use Case** |
| --- | --- | --- |
| **driver.quit();** | Closes **all** browser windows opened by Selenium and **terminates the WebDriver session**. | Used at the **end of a test** to clean up and free resources. |
| **driver.close();** | Closes **only the current active window** but keeps the WebDriver session active if multiple windows/tabs are open. | Used when working with **multiple windows** to close a specific window but keep the session alive. |

**How to get ScreenShot ?**

String timestamp = new SimpleDateFormat("yyyyMMdd\_HHmmss").format(new Date());

File screenshot = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE); FileUtils.copyFile(screenshot, new File("C:\\Screenshots\\screenshot\_" + timestamp + ".png"));

**How to handle broken links ?**

WebDriver driver;  
WebDriverWait wait;  
int broken\_links = 0;  
int vaild\_links = 0;  
int invalid\_links = 0;  
public void y(){  
 ChromeOptions op = new ChromeOptions();  
 op.addArguments("--start-maximized");  
 driver = new ChromeDriver(op);  
 driver.manage().timeouts().implicitlyWait(Duration.*ofSeconds*(10));  
 wait = new WebDriverWait(driver,Duration.*ofSeconds*(10));  
 driver.get("https://www.amazon.in/");  
 long start = System.*currentTimeMillis*();  
 driver.findElements(By.*tagName*("a")).parallelStream().forEach(w->checkURL(w.getAttribute("href")));  
 long end = System.*currentTimeMillis*();  
 System.*out*.println(end-start);  
 System.*out*.println("Vaild Links "+vaild\_links);  
 System.*out*.println("InVaild Links "+invalid\_links);  
 System.*out*.println("Broken Links "+broken\_links);  
   
}  
public void checkURL(String link){  
 try {  
 URL url = new URL(link);  
 HttpURLConnection hp = (HttpURLConnection) url.openConnection();  
 hp.setConnectTimeout(5000);  
 int rp = hp.getResponseCode();  
 if(rp>=400) {  
 System.*out*.println(link + "is Broken Link With response code "+rp+" "+hp.getResponseMessage());  
 broken\_links++;  
 }  
 else {  
 System.*out*.println(link + " is vaild and response is code "+hp.getResponseCode());  
 vaild\_links++;  
 }  
 }  
 catch (Exception e){  
 invalid\_links++;  
 }  
}

**TestNG**

**What is TestNG ?**

**TestNG** (Test Next Generation) is a **testing framework for Java** that is widely used for **unit, integration, and functional testing**. It provides powerful **annotations, parallel execution, and reporting** capabilities.

**What are advantage of TestNG ?**

* TestNG Provides Parallel execution of test methods
* It allows to define dependency of one test method over other method
* It allows to assign priority to test methods
* It allows grouping of test methods into test groups
* It has support for parameterizing test cases using @Parameter annotations
* It allows data driven testing using @DataProvider annotation
* It has different assertions that helps in checking the expected and actual result
* Detailed (HTML) reports

**Annotations in TestNG**

**1. List of TestNG Annotations**

| **Annotation** | **Purpose** |
| --- | --- |
| @Test | Marks a method as a test case. |
| @BeforeMethod | Runs **before each** test method. |
| @AfterMethod | Runs **after each** test method. |
| @BeforeClass | Runs **once before** the first test in a class. |
| @AfterClass | Runs **once after** the last test in a class. |
| @BeforeTest | Runs before tests in a <test> tag in testng.xml. |
| @AfterTest | Runs after tests in a <test> tag in testng.xml. |
| @BeforeSuite | Runs once **before all tests** in a suite. |
| @AfterSuite | Runs once **after all tests** in a suite. |
| @BeforeGroups | Runs before a specified test group starts execution. |
| @AfterGroups | Runs after a specified test group completes execution. |
| @DataProvider | Provides **data-driven testing** capabilities. |
| @Parameters | Passes **parameters from testng.xml** to test methods. |
| @Listeners | Listens to test execution events (used for logging & reporting). |
| @Factory | Used for **dynamic test case generation**. |

**TestNG.xml Tag in parent to child order**

<suite>

<test>

<classes>

<class>

<method>

<?xml version="1.0" encoding="UTF-8"?> // TestNG scehme  
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">  
<suite name="All Test Suite">  
 <test name="MaseterTest">  
 <classes>  
 <class name="testCases.TC001\_AccountRegistrationTest"></class>  
 <class name="testCases.TC002\_LoginTest"></class>  
 </classes>  
 </test>  
</suite>

**Why do we need a testng.xml file in the TestNG framework? How does it help in managing test execution?**

In TestNG framework, we need to create testng.xml file to create and handle multiple test classes. We do configure our test run, set test dependency, include or exclude any test, method, class or package and set priority etc in the xml file.

**What is the importance of testng.xml file?**

In a Selenium TestNG project, we use testng.xml file to configure the complete test suite in a single file. Some of the features are as follows.

• testng.xml file allows to include or exclude the execution of test methods and test groups

• It allows to pass parameters to the test cases

• Allows to add group dependencies

• Allows to add priorities to the test cases

• Allows to configure parallel execution of test cases

• Allows to parameterize the test cases

**How to pass parameter through testng.xml file to a test case?**

We could define the parameters in the testng.xml file and then reference those parameters in the source files.

public class Practise {  
 @Test  
 @Parameters({"browser","os"})  
 public void setup(String broswer,String os){  
 System.*out*.println(os + " "+broswer);  
 }  
}

<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">  
<suite name="All Test Suite">  
 <test name="MaseterTest">  
 <parameter name="os" value="Windows"></parameter>  
 <parameter name="browser" value="chrome"></parameter>  
 <classes>  
 <class name="testCases.Practise"></class>  
 </classes>  
 </test>  
</suite>

**What is TestNG Assert and list out common TestNG Assertions?**

TestNG Asserts help us to verify the condition of the test in the middle of the test run. Based on the TestNG Assertions, we will consider a successful test only if it is completed the test run without throwing any exception.

Some of the common assertions supported by TestNG are

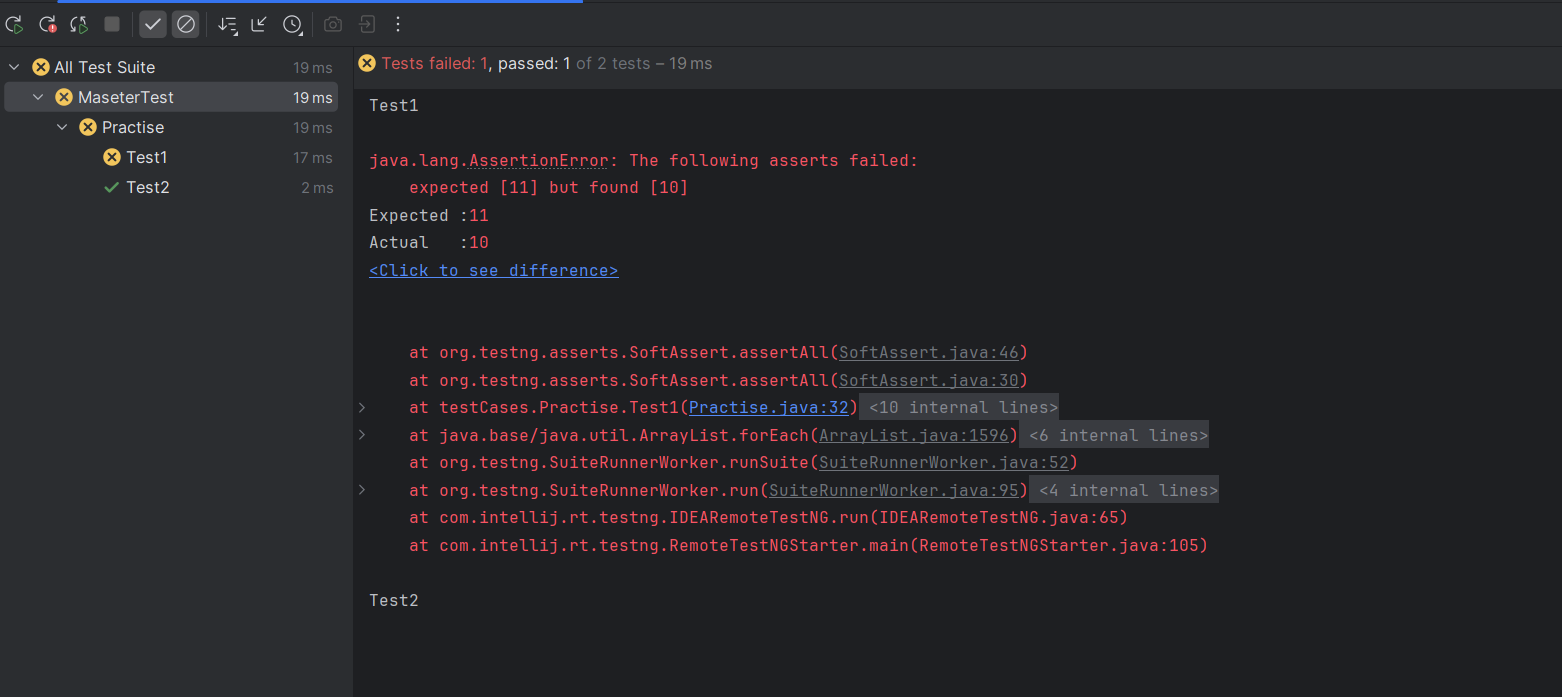
* assertEqual(String actual, String expected)
* assertEqual(String actual, String expected, String message)
* assertEquals(boolean actual, boolean expected)
* assertTrue(condition)
* assertTrue(condition, message)
* assertFalse(condition)
* assertFalse(condition, message)

**What is Soft Assert in TestNG?**

Soft Assert collects errors during @Test. Soft Assert does not throw an exception when an assert fails and would continue with the next step after the assert statement.

If there is any exception and you want to throw it then you need to use assertAll() method as a last statement in the @Test and test suite again continue with next @Test as it is.

@Test  
public void Test1(){  
 SoftAssert softAssert = new SoftAssert();  
 softAssert.assertEquals(10,11);  
 System.*out*.println("Test1");  
 softAssert.assertAll();  
}  
  
@Test  
public void Test2(){  
 System.*out*.println("Test2");  
}



**What is Hard Assert in TestNG?**

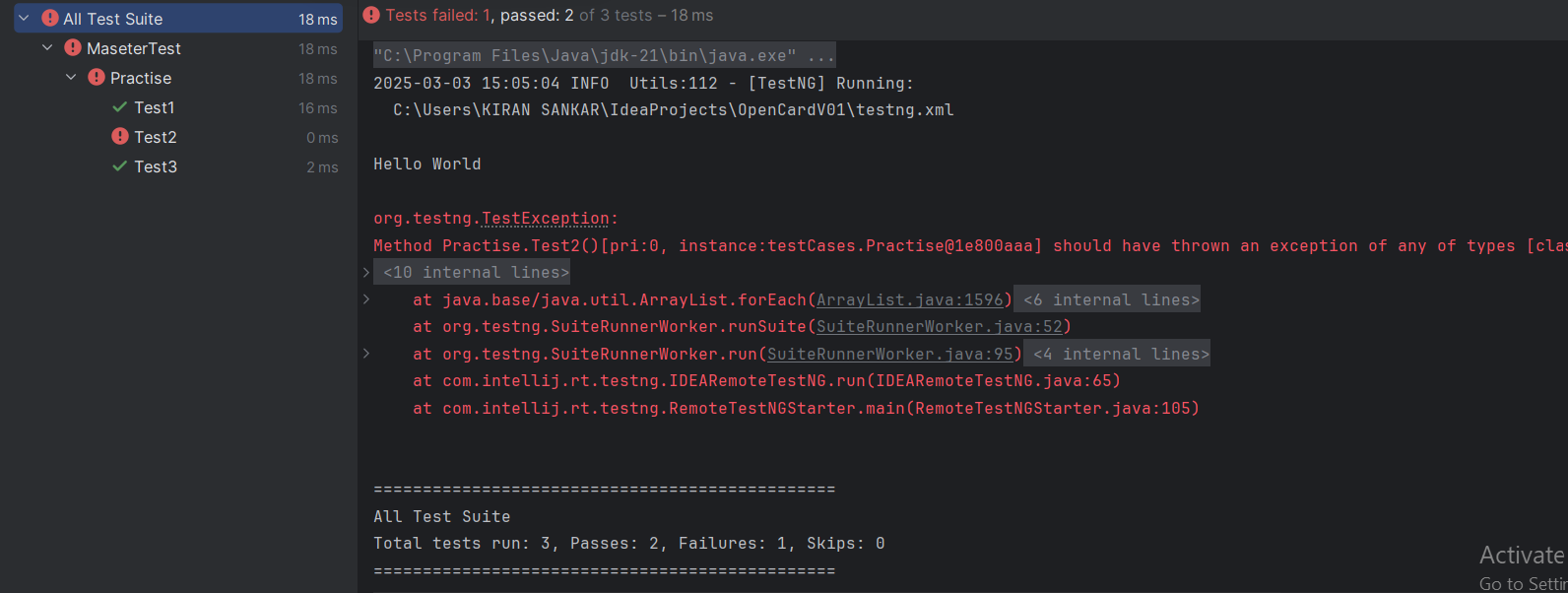
Hard Assert throws an AssertException immediately when an assert statement fails and test suite continues with next @Test

public class Practise {  
 @Test  
 public void Test1(){  
 System.*out*.println("Test1");  
 Assert.*assertEquals*(10,11);  
 }  
 @Test  
 public void Test2(){  
 System.*out*.println("Test2");  
 }  
}

**expectedExceptions Attribute in TestNG**

The expectedExceptions attribute in TestNG is used to verify if a test method throws an **expected exception** during execution. If the specified exception is thrown, the test **passes**; otherwise, it **fails**.

public class Practise {  
 @Test(expectedExceptions = ArithmeticException.class)  
 public void Test1(){  
 System.*out*.println(1/0);  
 }  
  
 @Test(expectedExceptions = {NoSuchElementException.class, NullPointerException.class})  
 public void Test2(){  
 System.*out*.println("Hello World");  
 }  
  
 @Test(expectedExceptions = {NoSuchElementException.class, NullPointerException.class},expectedExceptionsMessageRegExp = "No Element Found.\*")  
 public void Test3(){  
 throw new NoSuchElementException("No Element Found");  
 }  
}



**How to set test case priority in TestNG?**

We use priority attribute to the @Test annotations. In case priority is not set then the test scripts execute in alphabetical order.

public class Practise {  
 @Test(priority = 1)  
 public void Test1(){  
 System.*out*.println("Test 1");  
 }  
 @Test(priority = 3)  
 public void Test2(){  
 System.*out*.println("Test 2");  
 }  
 @Test(priority = 2)  
 public void Test3(){  
 System.*out*.println("Test 3");  
 }  
 @Test(priority = 3)  
 public void TestA(){  
 System.*out*.println("Test 4A");  
 }  
 @Test(priority = 3)  
 public void TestB(){  
 System.*out*.println("Test 4B");  
 }  
}

Output:

Test 1

Test 3

Test 2

Test 4A

Test 4B

**What is Parameterized testing in TestNG?**

Parameterized tests allow developers to run the same test over and over again using different values.

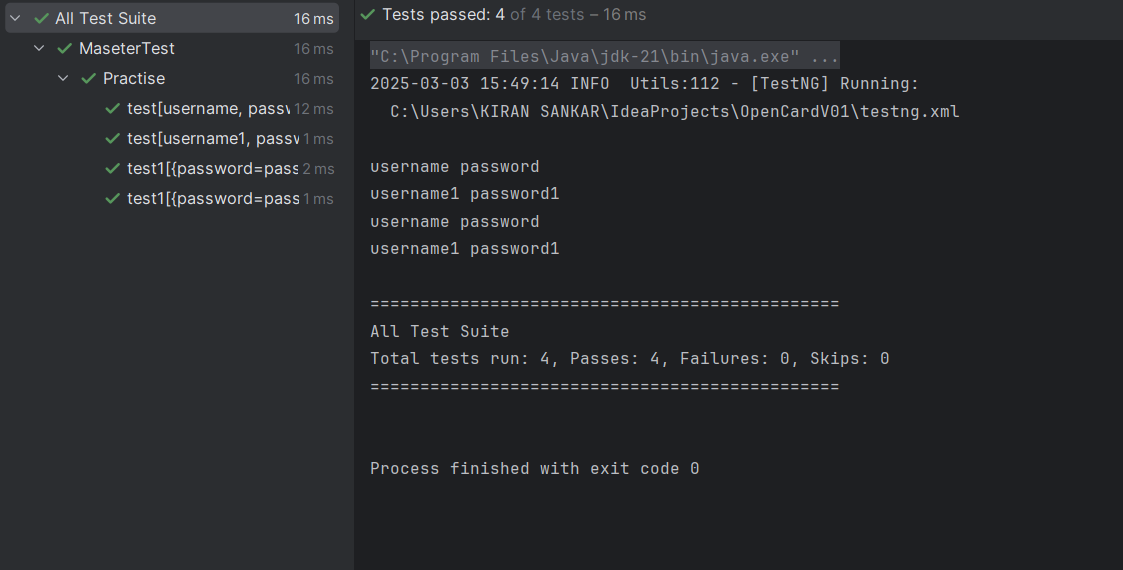
There are two ways to set these parameters:

* + using testng.xml
  + using Data Providers

What is dataprovider in TestNG ?

By using dataProvider annotation we can create data driven framework

public class Practise {  
 @DataProvider(name = "getData")  
 public Object[][] getData(){  
 return new Object[][]{  
 {"username","password"},  
 {"username1","password1"}  
 };  
 }  
 @DataProvider(name = "getData1")  
 public Object[] getData1(){  
 HashMap<String,String> map = new HashMap<String,String>();  
 map.put("username","username");  
 map.put("password","password");  
  
 HashMap<String,String> map1 = new HashMap<String,String>();  
 map1.put("username","username1");  
 map1.put("password","password1");  
  
 return new Object[]{map,map1};  
 }  
 @Test(dataProvider = "getData")  
 public void test(String username,String password){  
 System.*out*.println(username+" "+password);  
 }  
 @Test(dataProvider = "getData1")  
 public void test1(HashMap<String,String> map){  
 System.*out*.println(map.get("username")+" "+map.get("password"));  
 }  
}



**What is Parellel testing in TestNG ?**

Parallel testing in TestNG allows multiple test cases, test methods, or test classes to run simultaneously, reducing execution time and improving efficiency.

TestNG provides **four levels** of parallel execution:

| **Parallel Mode** | **Description** |
| --- | --- |
| methods | Runs **each test method** in a separate thread. |
| classes | Runs **each test class** in a separate thread. |
| instances | Runs **each instance of a test class** in a separate thread. |
| tests | Runs **each test inside <test> tag in XML** in a separate thread. |

<suite name="Suite" parallel="methods" thread-count="2">

<test name="TestA">

<classes>

<class name="A"/>

</classes>

</test>

</suite>

**What is groups in TestNG ?**

**Groups in TestNG** allow you to categorize test methods and run specific sets of tests together. This is useful for organizing large test suites, selectively executing tests, and managing dependencies.

So we can mention groups in Test annotation and then we can also control which group to execute in the XML file as include and exclude tag..

This scenario:

public class Practise2 {  
  
 @Test(groups = "smoke")  
 public void Test1(){  
 System.*out*.println("Test 1");  
 }  
  
 @Test(groups = "regression")  
 public void Test2(){  
 System.*out*.println("Test 2");  
 }  
  
 @Test(groups = "sanity",dependsOnGroups = {"smoke","regression"})  
 public void Test3(){  
 System.*out*.println("Test3");  
 }  
}

<suite name="All Test Suite" thread-count="2">  
 <groups>  
 <run>  
 <include name="smoke"></include>  
 <include name="sanity"></include>  
 <exclude name="regression"></exclude>  
 </run>  
 </groups>  
 <test name="test1">  
 <classes>  
 <class name="testCases.Practise2"></class>  
 </classes>  
 </test>  
</suite>

Here all three groups will be executed because we our sanity group depends on both smoke and regression and TestNG will forcefully run this.

**When do we use always-run = true ?**

Always run true means that method will always run no matter what happened to its dependonmethods method or dependsongroup method

@Test(dependsOnMethods = "t2",alwaysRun = true)  
 public void t1(){  
 System.*out*.println("Hello");  
}

Note : If you are running groups include xml file then the group method will only execute before and after method wont execute in that case you can alwasyRun = true

**invocationCount in TestNG**

In **TestNG**, the invocationCount attribute is used to execute a test method **multiple times** in a single test execution. It helps in **load testing, reliability testing, and performance testing** of an application.

public class Practise {  
 @Test(invocationCount = 5, threadPoolSize = 2)  
 public void parallelTest() {  
 System.*out*.println("Running in Thread: " + Thread.*currentThread*().getId());  
 }  
}

**Listeners in TestNG**

* Listeners in TestNG is used to customize the behaviour of the test execution and gather information about the test execution lifecycle.
* TestNG listeners allow we to perform various actions before, during, and after test

public class Practise implements ITestListener {  
  
 @Override  
 public void onTestStart(ITestResult result) {  
 System.*out*.println("Test Started: " + result.getName());  
 }  
  
 @Override  
 public void onTestSuccess(ITestResult result) {  
 System.*out*.println("Test Passed: " + result.getName());  
 }  
  
 @Override  
 public void onTestFailure(ITestResult result) {  
 System.*out*.println("Test Failed: " + result.getName());  
 // You can add screenshot capture logic here for failed tests  
 }  
  
 @Override  
 public void onTestSkipped(ITestResult result) {  
 System.*out*.println("Test Skipped: " + result.getName());  
 }  
}