**TestNG Interview Questions:**

1. How to skip the test case
2. Difference between invocation count and Thread count?
3. How to execute failure test cases in TestNG
4. What is the listener used to execute failure test cases?
5. What is the report format you have used in your project?

Selenium Interveiw Questions

1. Suppose there are n frames in a webpage and there is a button in one of the inner frames. How to click that button.
2. How to take full screenshot of a webpage
3. Listener used to generate report?
4. How to run the test cases in mutliple browser by making the browser instance as unique
5. Every application has an Authentication username and password. How to pass these username and password in the testng.xml at test level?
6. We have grid that having scroll bars. Normally it has 10 records. If we move forward it has another 10 records. But now I want to get all the record count in that grid instead of getting the record count of each grid?
7. There is a button in an application. What are different options we have to click that button?
8. Can we click on the button with HTML DOM operations?
9. If we click on any button it will open new window or popup. So instead of open a new window or popup I have to open new tab?
10. Is there any way to perform mouse operations?
11. If I want to generate custom popups where I want to send text in that popup. What is the approach ou follow?

Java Interview Questions.

1. Purpose of static in java.
2. How can we call static method or variable?
3. How to call non-static method from static method? And calling the static method from non-static method.

Answer: We cannot call non-static method from static method because static method belongs to class and not the instance.

1. What is the difference between static and non-static method?
2. Suppose there is one class having two static method and one non-static method. What is the difference between calling a static method from static method and calling a static method from a non-static method?
3. We are having 3 classes. Class A , Class B and Class C. Class A is Top level Parent . Class B is the root level child and Class C is inheriting Class B and Class B is inheriting Class A . There is method in all 3 classes which is overridden . (Multi level inheritance)
4. Have your ever worked on Static constructors in java
5. How will you iterate list of Hashmaps?
6. How to work with filters in selenium like I have page where when I click on Name button the list will be sorted. i.e it will display in either in Ascending or descending. Now I have to check whether it is in sorted order or not?
7. Do we have any default collection object which will sort the elements in ascending order?
8. Have you worked on the linked hash map?

BDD Cucumber:

1. How to execute one test case with multiple sets of data. What are the different approaches?
2. What are the options we have in Test runner class?
3. Tags: I have an excel where I have used tags for each test case and will provide the run flag to execute the tag. The requirement is by taking the tags which has run flag as “Y” and it has to create one maven command automatically?

UFT Questions:

1. How to work with file operations in UFT?
2. I have 1000 lines of data in a file so if I want to print all the lines one by one?
3. I have an excel sheet. In that I want to enter one value in 14th row 15 column
4. How to switch between multiple sheets in an excel shhets. i.e. perform some operations on first and then switch the first sheet from 3rd sheet. And 3rd sheet from 1st sheet?
5. How to work with Data base in UFT?
6. I am retrieving 2 rows and I want to get the value from 2nd row and 1st column
7. How to perform operations on ALM with VBScript? If I want to run the test case from HP ALM Test Plan or Test Lab.
8. How will you differentiate your test case whether it is in Test plan or Test lab in the VBS file?
9. With out clicking on the link how to check the link is broken or not?
10. How to modify tool settings?
11. I am clicking on a link. Once I click on a link a popup will appear like loading text is appearing . when the browser is not in synch status the browser will disappear. And when the browser is synch mode then only the opup will appear. Can we capture that particular text from the popu during run time?
12. I have 40 rows and 50 columns. In each cell there are 3 radio buttons. Now I want to click on second radio of 40th row and 10 column?
13. How to get the rowcount and column of a webtable?
14. I have data table . How can you get the row count of a data table?
15. How to get the 3rd row , 2nd column value in a web table?
16. I am having 100 links in an application . How can I print all the names of the links.
17. What are the different operations we have click on a link?
18. How can we perform mouse operations in UFT? – Analog recording, device replay, fire event
19. How to get the color of a particular cell in a webtable?
20. Difference between by val and by ref?

Inheritance: used to add additional functionalities to an existing class. Inheritance is used to extend the present class by adding some more properties to it. Inheritance is used to reuse the present tried and tested code so that you may not have to write them and compile them again.

**Scenario1: Child class reference and Child class object** – This will allow to access all the methods of base class(parent class) and child class.

Example: ChildClass obj1=new ChildClass();

**Scenario2: Base Class reference and Base class object** – This will allow to access all the methods of base class only.

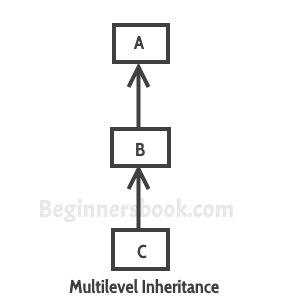
Example: Baseclass obj1=new Baseclass();

**Scenario3: Base class reference and child class object** – This will allow to accesss all the methods of base class and not child class.

Example: Baseclass obj1=new ChildClass();

**Scenario 4: Child class reference and BaseClass Object** – This will be invalid scenario because child cannot hold parent

**Multi level Inheritance:** When a class extends a class, which extends another class then this is called **multilevel inheritance**. For example class C extends class B and class B extends class A then this [type of inheritance](https://beginnersbook.com/2013/05/java-inheritance-types/) is known as multilevel inheritance.



It’s pretty clear with the diagram that in Multilevel inheritance there is a concept of grand parent class. If we take the example of this diagram, then class C inherits class B and class B inherits class A which means B is a parent class of C and A is a parent class of B. So in this case class C is implicitly inheriting the properties and methods of class A along with class B that’s what is called multilevel inheritance.

Multilevel Inheritance Example

In this example we have three classes –  Car, Maruti and Maruti800. We have done a setup – class Maruti extends Car and class Maruti800 extends Maruti. With the help of this Multilevel hierarchy setup our Maruti800 class is able to use the methods of both the classes (Car and Maruti).

class Car{

public Car()

{

System.out.println("Class Car");

}

public void vehicleType()

{

System.out.println("Vehicle Type: Car");

}

}

class Maruti extends Car{

public Maruti()

{

System.out.println("Class Maruti");

}

public void brand()

{

System.out.println("Brand: Maruti");

}

public void speed()

{

System.out.println("Max: 90Kmph");

}

}

public class Maruti800 extends Maruti{

public Maruti800()

{

System.out.println("Maruti Model: 800");

}

public void speed()

{

System.out.println("Max: 80Kmph");

}

public static void main(String args[])

{

Maruti800 obj=new Maruti800();

obj.vehicleType();

obj.brand();

obj.speed();

}

}

Output:

Class Car

Class Maruti

Maruti Model: 800

Vehicle Type: Car

Brand: Maruti

Max: 80Kmph

**Method Overriding in Java:** When a class extends its super class, all or some members of super class are inherited to sub class. When a inherited super class in modified in sub class, then we call its method as method over rided.  Through method overriding, we can modify super class method according to requirements of sub class.

Method Overriding in java is most useful features of java. Through inheritance we can reuse already existed code and through method overriding we can modify that reused code according to our requirements. This can be best explained with example.

class SuperClass

{

    void methodOfSuperClass()

    {

        System.out.println("From Super Class");

    }

}

class SubClass extends SuperClass

{

    void methodOfSuperClass()

    {

**//SuperClass method is overrided**

        //We can keep any tasks here according to our requirements.

        System.out.println("From Sub Class");

    }

}

public class MethodOverriding

{

    public static void main(String[] args)

    {

        SuperClass superclass = new SuperClass();

        superclass.methodOfSuperClass();         //Output : From Super Class

        SubClass subclass = new SubClass();

        subclass.methodOfSuperClass();          //Output : From Sub Class

    }

}

**Method overriding rules:**

**1)Name of the overrided method:** must be same as in the super class. You can’t change name of the method in subclass.

**2)Return type of overrided method:** The return type of the overrided method must be compatible with super class method. If super class method has primitive data type as its return type, then overrided method must have same return type in sub class also. If super class method has derived or user defined data type as its return type, then return type of sub class method must be of same type or its sub class. For example,

class SuperClass

{

    void firstMethodOfSuperClass()

    {

        System.out.println("From Super Class");

    }

    double secondMethodOfSuperClass()

    {

        return 0.0;

    }

    Object thirdMethodOfSuperClass()

    {

        return new Object();

    }

}

class SubClass extends SuperClass

{

    int firstMethodOfSuperClass()

    {

**//Compile time error, return type must be void not int**

    }

    void secondMethodOfSuperClass()

    {

**//Complie time error, return type must be double not void**

    }

    String thirdMethodOfSuperClass()

    {

**//No Compile time error,**

        //return type is compatible with super class method, because

        //String is sub class of Object class

        return new String();

    }

}

**Visibility of Overrided Method:** You can keep same visibility or increase the visibility of overrided method but you can’t reduce the visibility of overrided methods in the subclass. For example, default method can be overided as default or protected or public method but not as private.For example,

class SuperClass

{

    protected void methodOfSuperClass()

    {

        System.out.println("From Super Class");

    }

}

class SubClass extends SuperClass

{

    private void methodOfSuperClass()

    {

        //Compile time error, can't reduce visibility of overrided method

        //here, visibility must be protected or public but not private or default

    }

}

**Arguments of Overrided method:** For method to be properly overrided, You must not change arguments of method in subclass. If you change the number of arguments or types of arguments of overrided method in the subclass, then method will be overloaded not overrided.

class SuperClass

{

    void methodOfSuperClass()

    {

        System.out.println("From Super Class");

    }

}

class SubClass extends SuperClass

{

    //This class will have two methodOfSuperClass() methods.

    //one is from super class which takes no argument

    //and one is below method which takes one argument

    void methodOfSuperClass(int i)

    {

        System.out.println(i);

    }

}

public class MethodOverloading

{

    public static void main(String[] args)

    {

        SuperClass superclass = new SuperClass();

        superclass.methodOfSuperClass();         //Output : From Super Class

        SubClass subclass = new SubClass();

        subclass.methodOfSuperClass();          //Output : From Super Class

        subclass.methodOfSuperClass(10);       // Output : 10

    }

}

**Method Overloading in Java:**

When a class has more than one method with same name, then we call that method is overloaded. The overloaded methods will have different number of arguments or different types of arguments, but name of the methods remains same.

Compiler checks **method signature** for duplicate methods or for method overloading. method signature consist of three things, **1) Method Name   2) Number Of Arguments   3) Types of arguments.**

If these three things are same for any two methods in a class, then compiler gives **duplicate method error.**

Compiler first checks method name. If it is same, then it checks number of arguments. If methods differs in number of arguments, then it does not check types of argument. It treats as methods are overloaded. If number of arguments are same then compiler checks types of arguments. If types of arguments are also same, then compiler will give duplicate method error. If types of arguments are not same, then compiler will treat them as methods are overloaded.

Example:

public class MethodOverloading

{

    void methodOverloaded()

    {

**//No argument method**

    }

    void methodOverloaded(int i)

    {

**//One argument is passed**

    }

    void methodOverloaded(double d)

    {

**//One argument is passed but type of argument is different**

    }

    void methodOverloaded(int i, double d)

    {

**//Two argument method**

**//Method signature of this method is methodOverloaded(int, double)**

    }

    void methodOverloaded(double d, int i)

    {

**//It is also two argument method but type of arguments changes**

**//Method signature of this method is methodOverloaded(double, int)**

    }

    void methodOverloaded(double d1, int i1)

    {

**//It has same method signature methodOverloaded(double, int) as of above method**

**//So, it is a Duplicate method, You will get compile time error here**

    }

    void differentMethod()

    {

        //Different method

    }

}

Overloaded methods may have same return types or different return types. It does not effect method overloading.

public class MethodOverloading

{

    void methodOverloaded()

    {

        //No argument method, return type is void

    }

    int methodOverloaded(int i)

    {

        //Returns int type

        return i;

    }

    int methodOverloaded(double d)

    {

        //Same return type as of above method

        return 0;

    }

    void methodOverloaded(double d)

    {

        //Duplicate method because it has same method signature as of above method

    }

}

**Important Note :**

If two methods have same signature and different return types, then those methods will not be treated as two different methods or methods overloaded. For duplication, compiler checks only method signature not return types. If method signature is same, straight away it gives duplicate method error.

Overloaded methods may have same access modifiers or different access modifiers. It also does not effect method overloading

public class MethodOverloading

{

    private void methodOverloaded()

    {

        //No argument, private method

    }

    private int methodOverloaded(int i)

    {

        //One argument private method

        return i;

    }

    protected int methodOverloaded(double d)

    {

        //Protected Method

        return 0;

    }

    public void methodOverloaded(int i, double d)

    {

        //Public Method

    }

Overloaded methods may be static or non-static. This also does not effect method overloading.

public class MethodOverloading

{

    private static void methodOverloaded()

    {

        //No argument, private static method

    }

    private int methodOverloaded(int i)

    {

        //One argument private non-static method

        return i;

    }

    static int methodOverloaded(double d)

    {

        //static Method

        return 0;

    }

    public void methodOverloaded(int i, double d)

    {

        //Public non-static Method

    }

}

From the above examples, it is clear that compiler will check only method signature for method overloading or for duplicate methods. It does not check return types, access modifiers and static or non-static.

Difference between Method Overloading and Overriding in Java

|  |  |  |
| --- | --- | --- |
|  | Method Overloading | Method Overriding |
| **Definition** | When a class has more than one method with same name but with different different arguments then we call it as method overloading | When a super class method is modified in the subclass , then we call this as method overriding. |
| Method Signature | Overloaded methods must have different method signatures. That means they should differ at least in any one of these three things – Number of arguments, Types of arguments and order of arguments. But, they must have same name. | Overridden methods must have same method signature. I.e. you must not change the method name, types of arguments, number of arguments and order of arguments while overriding a super class method. |
| **Return Types** | Overloaded methods can have same or different return types. | The return type of the overridden method must be compatible with that of super class method. That means if super class method has primitive type as its return type, then it must be overridden with same return type. If super class method has derived type as its return type then it must be overridden with same type or its sub class type. |
| **Visibility**(private, public, protected and default) | Overloaded methods can have same visibility or different visibility. | While overriding a super class method either you can keep the same visibility or you can increase the visibility. But you can’t reduce it. |
| **Static Context** | Overloaded methods can be static or not static. It does not affect the method overloading. | You can’t override a static method. |
| **Binding** | Binding between method call and method definition happens at compile time (Static Binding). | Binding between method call and method definition happens at run time (Dynamic Binding). |
| **Polymorphism** | It shows static polymorphism. | It shows dynamic polymorphism. |
| **Private methods** | Private methods can be overloaded. | Private methods can’t be overridden. |
| **Final Methods** | Final methods can be overloaded | Final methods can’t be overridden. |
| **Class Requirement** | For method overloading, only one class is required. I.e. Method overloading happens within a class. | For method overriding, two classes are required – super class and sub class. That means method overriding happens between two classes. |

**Constructor Overloading Key Points:**

**1)**Constructor name should be same as class name. if you give another name it will give compile time error.  If you give another name, it is neither a method because of no return type, nor constructor because name is different from class name.

class A

{

     A()

     {

         // Constructor of Class A

     }

     A1()

     {

**// Compile time error, It is neither a constructor nor a method**

     }

}

**2)Constructor must not have return type. If you keep return type for the constructor , it will be treated as another method. But compiler gives you warning that this method has a constructor name.**

class A

{

     A()

     {

         // Constructor of Class A, not having any return type.

     }

     void A()

     {

**// constructor having a return type, It will be treated as method but with a warning.**

     }

}

**3)Every class should have at least one constructor. If you don’t write constructor for your class, compiler will give default constructor. Default constructor is always public and it has no arguments (No-Arg Constructor).**

class A

{

     // No Constructors written

}

Compiler will treat the above code as,

class A

{

    public A()

    {

        //   Constructor provided by the compiler.

    }

}

**4)**Constructor can be declared as private. If you declare constructor as private, you can’t use it outside that class.

class A

{

     private A()

     {

          // Private Constructor

     }

     void methodOne()

     {

          //You can use private constructor inside the class

          A a1 = new A();

     }

}

class MainClass

{

     public static void main(String[] args)

     {

          //You can't use private constructor ouside the class like this

          // A a1 = new A();

     }

}

**5)One class can have more than one constructors. It is called Constructor Overloading. Through constructor overloading, you can have multiple ways to create objects.**

class A

{

     A()

     {

        // First Constructor

     }

     A(int i)

     {

        // Second Constructor

     }

    A(int i, int j)

    {

       // Third Constructor

    }

}

you can create the objects to the above class in three ways like below,

class MainClass

{

     public static void main(String[] args)

     {

          A a1 = new A();      //Using First Constructor

          A a2 = new A(10);    // Using Second Constructor

          A a3 = new A(10, 20);    // Using Third Constructor

     }

}

6)Duplicate Constructors not allowed. If you keep duplicate constructors, you will get compile time error.

class A

{

     A(int i, int i)

     {

         // Duplicate Arguments Passed. It gives compile time error

     }

}

7)Only public, protected and private keywords are allowed before a constructor name. If you keep any other keyword before a constructor name, it gives compile time error.

class A

{

     final A()

     {

**//Constructor can not be final**

     }

     static A()

     {

**//Constructor can not be static**

     }

     abstract A()

     {

**//Constructors can not be abstract**

     }

8)First statement in a constructor must be either super() or this(). If you put any other statements you will get compile time error.If you don’t include these statements, by default compiler will keep super() calling statement. super() – It is a calling statement to default constructor of super class. this()- it is a calling statement to constructor of the same class.

class A

{

     A()

     {

**//By Default, Compile will keep super() calling statement here.**

          System.out.println("First Constructor");

     }

     A(int i)

     {

**//Compiler will not keep any statement here**

          super();

          System.out.println("Second Constructor");

     }

     A(int i, int j)

     {

**//Compiler will not keep any statement here**

          this();

          System.out.println("Third Constructor");

     }

     A(int i, int j, int k)

     {

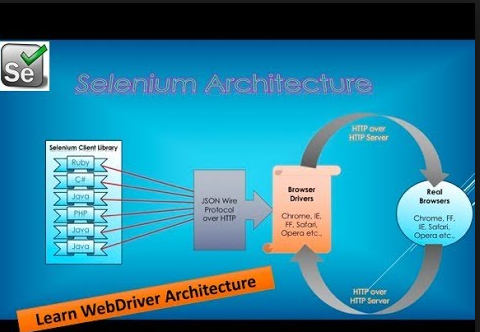
          System.out.println("Fourth Constructor");

**// super(); It will give error if you keep super() here**

     }

}

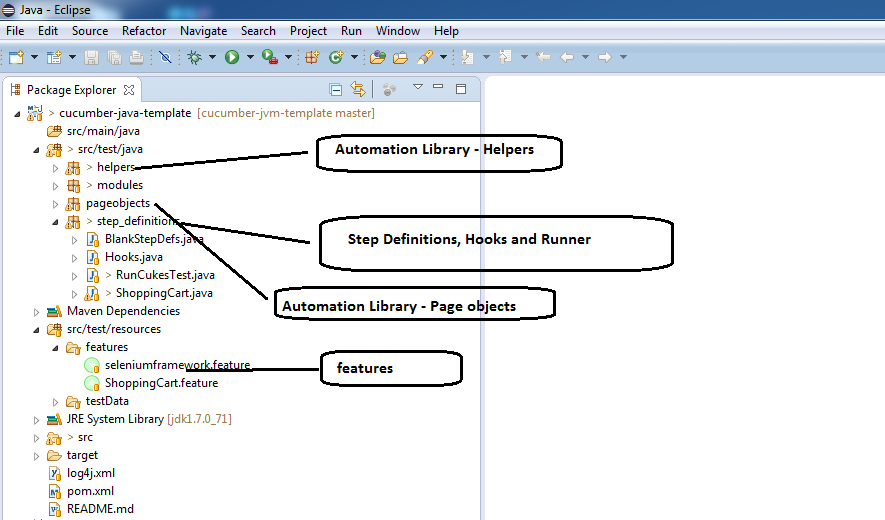
Selenium WebDriver Architecture



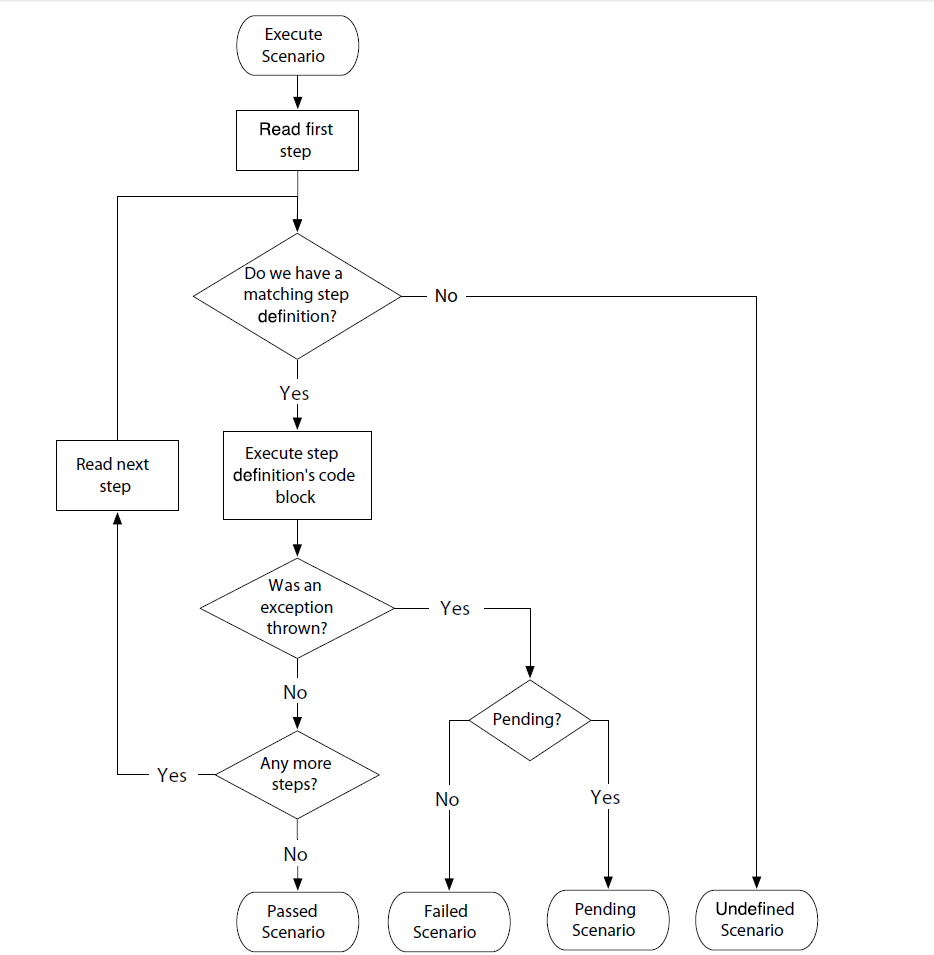
* When we execute the script Selenium Client library i.e. based on language will send the request to the server in the form of JSON Wire Protocol over HTTP.
* Over the HTTP Protocal all the requests are sending to Browser drivers.
* And these drivers are performing actions on the real browser.
* Once the action is completed the real browser wil send the response to the Browser driver in which it will send the response to the Client library.
* Everything will be available in the form of REST API( i.e. JAVA API , C# API, ….)

### **Cucumber & Java Project (illustration)**

**An illustration of a cucumber Java project looks as below.**



### **Cucumber Overall Sequence Flow/Work Flow**



**How to execute Test Runner Class from POM.XML file**

|  |
| --- |
| <plugins> |
|  |  |
|  | <plugin> |
|  |  |
|  | <groupId>org.apache.maven.plugins</groupId> |
|  | <artifactId>maven-compiler-plugin</artifactId> |
|  | <version>3.7.0</version> |
|  | <configuration> |
|  | <compilerVersion>3.7.0</compilerVersion> |
|  | <source>1.8</source> |
|  | <target>1.8</target> |
|  | </configuration> |
|  | </plugin> |
|  |  |
|  | <plugin> |
|  | <groupId>org.apache.maven.plugins</groupId> |
|  | <artifactId>maven-surefire-plugin</artifactId> |
|  | <version>2.20.1</version> |
|  | <configuration> |
|  | <includes> |
|  | **<exclude>\*\*/\*TestRunner.java</exclude>** |
|  | </includes> |
|  | </configuration> |
|  |  |
|  | </plugin> |
|  |  |

</plugins>

**Handling dropdown in selenium:**

**By using select class**

Select oSel=new Select(driver.findElement(By.xpath(“”));

List<WebElement> elementCount=oSel.getOptions();

Int elementsize=elementCount.size();

For(int i=0;i<elementsize;i++)

{

String sValue=elementCount.get(i).getText();

Syso(sValue)’

}

Different ways for selecting the option from DropDown

**1)SelectByVisibleText:**

Select oSel=new Select(driver.findElement(By.xpath(“”));

oSel.selectByVisibleText(“2010”);

**2)SelectByIndex:**

Select oSel=new Select(driver.findElement(By.xpath(“”));

oSel.selectByIndex(index);

**3)SelectByValue**

Select oSel=new Select(driver.findElement(By.xpath(“”));

oSel.selectByValue(“value”)

**Handling keyboard and Mouse events in Selenium:**

By using Actions class

**Actions act = new Actions(driver);**

**Keyboard events :**

1)act.SendKeys(KeysToSend)

2)act.KeyDown(Keys.ALT)

3)act.KeyUP(Keys.ALT)

**Mouse events:**

Actions act = new Actions(driver);

Act.moveToElement(WebElement).click().perform;

Act.clickAndHold()

Act.contextClick

Act.dragAndDrop(sourceElement, targetElement)

Act.release();

**Database connection:**

**Steps to follow:**

1)Create a connection for the data base using Connection statement.

**Connection con= DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");**

2)Create a statement using Statement

**Statement stmt=con.createStatement();**

3)Create a result set using Statement object to execute the query

**ResultSet rs=stmt.executeQuery("select \* from EMPLOYEE1");**

4)moves the cursor to last row

**rs.last();**

5)After moving the cursor to last row then getting the last row number

**int numResults = rs.getRow();**

**System.out.println("Total Rows are " + numResults);**

6)Iterate through the rows and get the data of a psecific column.

**while(rs.next())**

**{**

**String empname=rs.getString("EMP\_NAME");**

**System.out.println(empname);**

**}**

package testCases;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import org.testng.annotations.Test;

public class OracleDatabaseConnection {

@Test

public static void DBConnection() throws ClassNotFoundException, SQLException

{

//driver connection

Class.forName("oracle.jdbc.driver.OracleDriver");

//connection to Database

Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");

System.out.println("Connection loaded");

Statement stmt=con.createStatement();

ResultSet rs=stmt.executeQuery("select \* from EMPLOYEE1");

//moves the cursor to last row

rs.last();

//after moving the cursor to last row then getting the last row number

int numResults = rs.getRow();

System.out.println("Total Rows are " + numResults);

while(rs.next())

{

String empname=rs.getString("EMP\_NAME");

System.out.println(empname);

if(empname.equals("Geeta")){

System.out.println("Employee " + empname + "Found");

break;

}

}

}

}

**TestNG Topic**

**1)Difference between Invocation count and ThreadCount.**

***Invocation Count:*** The number of times a method should be invoked

Example: @Test(InvocationCount=5**)**

***ThreadCount:*** To execute the test case in different threads

**2)How to Skip the Test case in TestNG**

a) By using **@Test(enable-false)** we can skip the test case

b) By using t **throw new SkipException**

**Example :**

@Test

**public** **void** **testCaseSkipException**()

{

System.out.println("Im in skip exception");

**throw** **new** SkipException("Skipping this exception"); }

}

**How to execute Failed test cases in Cucumber**

Provide the below pllug in in Test runner class

@RunWith(Cucumber.class)

@CucumberOptions(

monochrome = true,

features = "classpath:features",

plugin = {"pretty", "html:target/cucumber-reports",

"json:target/cucumber.json",

**"rerun:target/rerun.txt**"} //Creates a text file with failed scenarios

,tags = "@mytag"

)

public class MyScenarioTests {

}

Then provide the target/rerun.txt in Test runner class in feature path

@RunWith(Cucumber.class)

@CucumberOptions(

monochrome = true,

**features = "@target/rerun.txt", //Cucumber picks the failed scenarios from this file**

format = {"pretty", "html:target/site/cucumber-pretty",

"json:target/cucumber.json"}

)

public class FailedScenarios {

}

**Exception Handling in Java:**

Exception Hanlding in Java is implemented using five keywords

1)Try

2)Catch

3)Finally

4)throw

5)throws

try, catch and finally keywords are main fundamentals of exception handling in java. The syntax for using these three keywords is,

try

{

**//This is the try block**

**//In this block, keep those statements which may**

**//throw run time exceptions**

}

catch(Exception e)

{

**//This is the catch block.**

**//It takes one argument of type java.lang.Exception**

**//This block catches the exceptions thrown by try block**

}

finally

{

**//This is the finally block.**

}

Example:

public class ExceptionHandling

{

    public static void main(String[] args)

    {

        String[] s = {"abc", "123", "xyz", "456"};   //String Array containing valid and invalid numeric values

        for (int i = 0; i < s.length; i++)

        {

            try

            {

                int intValue = Integer.parseInt(s[i]); **//This statement may throw NumberFormatException**

            }

**catch(NumberFormatException ex)**

            {

**System.out.println("The thrown NumberFormatException will be caught here");**

            }

            finally

            {

**System.out.println("This block is always executed");**

            }

        }

    }

}

**Key Points for try, catch and finally block:**

1. **When a statement throws an exception in the try block, the remaining part of the try block will not be executed. Program control comes out of the try block and enters directly into catch block**

public class ExceptionHandling

{

    public static void main(String[] args)

    {

        try

        {

            int i = 10/0;           **//This statement throws ArithmeticException**

**System.out.println("This statement will not be executed");**

        }

        catch(Exception ex)

        {

**System.out.println("This block is executed immediately after an exception is thrown");**

        }

        finally

        {

**System.out.println("This block is always executed");**

        }

    }

}

**2) try, catch and finally blocks form one unit. i.e You can’t keep other statements in between try, catch and finally blocks**.

public class ExceptionHandling

{

    public static void main(String[] args)

    {

**System.out.println("You can keep any number of statements here");**

        try

        {

            int i = 10/0;           //This statement throws ArithmeticException

**System.out.println("This statement will not be executed");**

        }

**//You can't keep statements here**

        catch(ArithmeticException ex)

        {

**System.out.println("This block is executed immediately after an exception is thrown");**

        }

**//You can't keep statements here**

        finally

        {

            System.out.println("This block is always executed");

        }

        System.out.println("You can keep any number of statements here");

    }

}

3**) You can display the description of an exception thrown using Exception object in the catch block.**

public class ExceptionHandling

{

    public static void main(String[] args)

    {

        try

        {

            String s = null;

            System.out.println(s.length());   //This statement throws NullPointerException

            System.out.println("This statement will not be executed");

        }

        catch(Exception ex)

        {

**System.out.println(ex);**    //Output : java.lang.NullPointerException

            ex.printStackTrace();     //This prints stack trace of exception

        }

        finally

        {

            System.out.println("This block is always executed");

        }

    }

}

**Multiple Catch Blocks**

1)In some cases, A single statement may throw more than one type of exception. In such cases, Java allows you to put more than one catch blocks. One catch block handles one type of exception.  When an exception is thrown by the try block, all the catch blocks are examined in the order they appear and one catch block which matches with exception thrown will be executed. After, executing catch block, program control comes out of try-catch unit.

public class ExceptionHandling

{

    public static void main(String[] args)

    {

        String[] s = {"abc", "123", null, "xyz"};   //String array containing one null object

        for (int i = 0; i < 6; i++)

        {

            try

            {

                int a = s[i].length() + Integer.parseInt(s[i]);

                //This statement may throw NumberFormatException, NullPointerException and ArrayIndexOutOfBoundsException

            }

**catch(NumberFormatException ex)**

            {

                System.out.println("NumberFormatException will be caught here");

            }

**catch (ArrayIndexOutOfBoundsException ex)**

            {

                System.out.println("ArrayIndexOutOfBoundsException will be caught here");

            }

**catch (NullPointerException ex)**

            {

                System.out.println("NullPointerException will be caught here");

            }

            System.out.println("After executing respective catch block, this statement will be executed");

        }

    }

}

2) From Java 7 onward, there is one more way for handling multiple exceptions. Multiple exceptions thrown by the try block can be handled by a single catch block using **pipe (|) operator**. By using pipe operator, the above example can be written as,

public class ExceptionHandling

{

    public static void main(String[] args)

    {

        String[] s = {"abc", "123", null, "xyz"};   //String array containing one null object

        for (int i = 0; i < 6; i++)

        {

            try

            {

                int a = s[i].length() + Integer.parseInt(s[i]);

                //This statement may throw NumberFormatException, NullPointerException and ArrayIndexOutOfBoundsException

            }

**catch(NumberFormatException | NullPointerException | ArrayIndexOutOfBoundsException ex)**

            {

                System.out.println("Now, this block handles NumberFormatException, NullPointerException and ArrayIndexOutOfBoundsException");

            }

        }

    }

}

3)**java.lang.Exception** is super class of all types of exception. (Types of exceptions will be discussed later). It handles all types of exceptions. In the above example, all catch blocks can be replaced by one catch block which handles all types of exceptions. This type of exception handling comes very handy when you are not sure about the types of exceptions your code may throw.

public class ExceptionHandling

{

    public static void main(String[] args)

    {

        String[] s = {"abc", "123", null, "xyz"};   //String array containing one null object

        for (int i = 0; i < 6; i++)

        {

            try

            {

                int a = s[i].length() + Integer.parseInt(s[i]);

                //This statement may throw NumberFormatException, NullPointerException and ArrayIndexOutOfBoundsException

            }

**catch(Exception ex)**

            {

                System.out.println("This block handles all types of exceptions");

            }

        }

    }

}

4)The order of catch blocks should be from most specific to most general ones. i.e Sub classes of Exception must come first and super classes later. If you keep the super classes first and sub classes later, you will get compile time error : **Unreachable Catch Block**.

public class ExceptionHandling

{

    public static void main(String[] args)

    {

        try

        {

            int i = Integer.parseInt("abc");   //This statement throws NumberFormatException

        }

**catch(Exception ex)**

        {

            System.out.println("This block handles all exception types");

        }

**catch(NumberFormatException ex)**

        {

**//Compile time error**

**//This block becomes unreachable as**

**//exception is already handled by above catch block**

        }

    }

}

**Retrun value from Try-Catch-Finally blocks**

If method returns a value and also has try, catch and finally blocks in it, then following two rules need to follow.

1) **If finally block returns a value then try and catch blocks may or may not return a value.**

2) **If finally block does not return a value then both try and catch blocks must return a value.**

public class ReturnValueFromTryCatchFinally

{

    public static void main(String[] args)

    {

        System.out.println(methodReturningValue());

    }

    static int methodReturningValue()

    {

        try

        {

**//This block may or may not return a value as finally block is returning a value**

        }

        catch (Exception e)

        {

**//This block may or may not return a value as finally block is returning a value**

        }

        finally

        {

            return 20;

        }

    }

public class ReturnValueFromTryCatchFinally

{

    public static void main(String[] args)

    {

        System.out.println(methodReturningValue());

    }

    static int methodReturningValue()

    {

        try

        {

**return 10;**

        }

        catch (Exception e)

        {

**return 20;**

        }

        finally

        {

**//Now, This block may or may not return a value**

**//as both try and catch blocks are returning a value**

        }

    }

}

3) **If try-catch-finally blocks are returning a value according to above rules, then you should not keep any statements after finally block. Because they become unreachable and in Java, Unreachable code gives compile time error**

public class ReturnValueFromTryCatchFinally

{

    public static void main(String[] args)

    {

        try

        {

            return;

        }

        catch (Exception e)

        {

            return;

        }

        finally

        {

            return;

        }

**System.out.println("Unreachable code");    //Compile Time Error : Unreachable Code**

    }

}

**4) finally block overrides any return values from try and catch blocks.**

public class ReturnValueFromTryCatchFinally

{

    public static void main(String[] args)

    {

        System.out.println(methodReturningValue());    //Output : 50

    }

    static int methodReturningValue()

    {

        try

        {

            return 10;

        }

        catch (Exception e)

        {

            return 20;

        }

        finally

        {

**return 50;    //This method returns 50 not 10 or 20**

        }

    }

}

1. finally block will be always executed even though try and catch blocks are returning the control.

public class ReturnValueFromTryCatchFinally

{

    public static void main(String[] args)

    {

        System.out.println(methodReturningValue());    //Output : 10

    }

    static int methodReturningValue()

    {

        try

        {

            return 10;    //control will not be passed to main() method here

        }

        catch (Exception e)

        {

            return 20;    //Control will not be passed to main() method here

        }

        finally

        {

            System.out.println("finally block is always executed");

            //Control will be passed to main() method after executing this block

        }

    }

}

**Types of Exceptions**

**1)Checked Exception:** Checked exceptions are known to compiler i.e they are the exceptions that are checked at compile time. Checked exceptions are also called **compile time exceptions**, because they can be known during compile time.

2)Unchecked Exception: Unchecked exceptions are not known to compiler.  They are the exceptions that are not checked at compile time, because they occur only at run time.That’s why these exceptions are also called **run time exceptions.**

**Throw Keyword:** used to explicitly throw an exception from a method or any block of code. We can throw either [checked or unchecked exception](https://www.geeksforgeeks.org/checked-vs-unchecked-exceptions-in-java/). The throw keyword is mainly used to throw custom exceptions.

**throw *Instance***

Example:

**throw new ArithmeticException("/ by zero");**

class ThrowExcep

{

    static void fun()

    {

        try

        {

            throw new NullPointerException("demo");

        }

        catch(NullPointerException e)

        {

            System.out.println("Caught inside fun().");

            throw e; // rethrowing the exception

        }

    }

    public static void main(String args[])

    {

        try

        {

            fun();

        }

        catch(NullPointerException e)

        {

            System.out.println("Caught in main.");

        }

    }

}

Throws Keyword: throws is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions. The caller to these methods has to handle the exception using a try-catch block.

**type method\_name(parameters) throws exception\_list**

exception\_list is a comma separated list of all the

exceptions which a method might throw.

**Difference between Throw and Throws keyword in Java:**

|  |  |
| --- | --- |
| Throw | Throws |
| **throw** keyword is used to throw an exception explicitly | **Throws clause** is used to declare an exception, which means it works similar to the try-catch block |
| If we see syntax wise than **throw** is followed by an instance of Exception class.  throw new ArithmeticException("Arithmetic Exception"); | **throws** is followed by exception class names.  throws ArithmeticException; |
| Throw keyword is used in the method body to throw an exception.  void myMethod() {  try {  //throwing arithmetic exception using throw  throw new ArithmeticException("Something went wrong!!");  }  catch (Exception exp) {  System.out.println("Error: "+exp.getMessage());  } | throws is used in method signature to declare the exceptions that can occur in the statements present in the method. |

**Strings in Java**

**String**: **String** is a sequence of characters, for e.g. “Hello” is a string of 5 characters. **In java, string is an immutable object which means it is constant and can cannot be changed once it has been created.**

**Two ways of creating an object:**

1. String literal
2. Using New Keyword

**1)String literal:**

String str1 = "Welcome";

String str2 = "Welcome";

**The problem with this approach:** As I stated in the beginning that String is an object in Java. However we have not created any string object using new keyword above. The compiler does that task for us it creates a string object having the string literal (that we have provided , in this case it is “Welcome”) and assigns it to the provided string instances.

But if the object already exist in the memory it does not create a new Object rather it assigns the same old object to the new instance, that means even though we have two string instances above(str1 and str2) compiler only created on string object (having the value “Welcome”) and assigned the same to both the instances. For example there are 10 string instances that have same value, it means that in memory there is only one object having the value and all the 10 string instances would be pointing to the same object.

What if we want to have two different object with the same string? For that we would need to create strings using **new keyword**.

### Using New Keyword

As we saw above that when we tried to assign the same string object to two different literals, compiler only created one object and made both of the literals to point the same object. To overcome that approach we can create strings like this:

String str1 = new String("Welcome");

String str2 = new String("Welcome");

In this case compiler would create two different object in memory having the same strin

String Methods:

[**char charAt(int index)**](https://beginnersbook.com/2013/12/java-string-charat-method-example/)**:** It returns the character at the specified index. Specified index value should be between 0 to length() -1 both inclusive. It throws IndexOutOfBoundsException if index<0||>= length of String.

**int indexOf(String str):** This method returns the index of first occurrence of specified substring str.

[**int lastindexOf(String str)**](https://beginnersbook.com/2013/12/java-string-lastindexof-method-example/)**:** Returns the index of last occurrence of string str.

**String substring(int beginIndex):** It returns the substring of the string. The substring starts with the character at the specified index.

**String substring(int beginIndex, int endIndex):** Returns the substring. The substring starts with character at beginIndex and ends with the character at endIndex.

**String[] split(String regex, int limit):** It splits the string and returns the array of substrings that matches the given regular expression. limit is a result threshold here.

[**String[] split(String regex)**](https://beginnersbook.com/2013/12/java-string-split-method-example/)**:** Same as split(String regex, int limit) method however it does not have any threshold limit

[**int length()**](https://beginnersbook.com/2013/12/java-string-length-method-example/)**:** It returns the length of a String.

**Difference between equals() and == in Java**

|  |  |
| --- | --- |
| Equals() | == |
| Equals() method is used for Content comparison | == |
| Equals() is a method in java | == is an operator in Java |
| equals() is method, which can be overridden in Java | Since Java doesn’t support Operator Overloading == behaves identical for every object |
| only used for objects comparison | used to compare both primitive(boolean,int,float…) and objects |

Example:String s1=new String(“Narayana”);

String s2=new String(“Narayana”);

Syso(s1==s2); //output : false (Below diagram shows it is referring two objects even they are same.

Syso(s1.equals(s2));

**Explain public static void main(String[] args)**

**Public:** to call by JVM anywhere

**Static:** without existing object also JVM has to call the method and main method no way related to any object.

**Void:** main method wont return anything to JVM

**Main(String[] args):** command line arguments.

We can declare “String[]” in any below acceptable form:

Main(String[] args) / main(String []args) / main(String args[])

**Case 1:** Overloading of main method is possible in java but JVM call only String[] argument .

The other overloaded method we have to call explicitly then it will be executed as a normal method call.

Class Test

{

Public static void main(String[] args)

{

Syso(“String[]”);

}

Public static void main(int[] args)

{

Syso(“int[]”);

}

}

Output: String[]

Case2: inheritance concept applicable for the main method. Hence while executing child class if child class doesn’t contain main method then parent class main method will be executed.

Class p

{

Public static void main(String[] args)

{

Syso(“parent main”);

}

Class c extends p

{

}

Output: parent main

Case 3: it seems overriding concept applicable for main method but it is not overriding it is method hiding.

Class p

{

Public static void main(String[] args)

{ Syso(“parent main”)}

Class c extends p

{

Public static void main(String[] args)

{ Syso(“child main”)}

}

**Difference between String and StringBuffer?**

**1)String:** String is immutable in java i.e. once we create an object we cannot perform(modify any action on that string)

Example: String s=new String(“Narayana”);

s.append(“Barnana”);

Syso(s); **Output:** Narayana

**2)StringBuffer**: StringBuffer is mutalbe in java. i.e. we can modify the string object.

Example: StringBuffer sb=new StringBuffer(“Narayana”);

Sb.append(“Narayana”); Syso(s); //output: NarayanaBarnana

**Difference between StringBuffer and StringBuilder**

|  |  |
| --- | --- |
| StringBuffer | StringBuilder |
| Every Method present in StringBuffer is Synchronized | Every Method present in StringBuffer is not Synchronized |
| At a time only one thread is allow to operate on StringBuffer object. Hence StringBuffer object is **Thread-Safe** | At a time multiple threads are allowed to operate on StringBuilder object. Hence StringBuilder object is not **Thread-Safe** |
| It increases waiting time of threads and hence relatively performance is slow | Threads are not required to wait to operate on StringBuilder object and hence relative performance is high |
| Introduced in 1.0 version | Introduced in 1.5 version |

**Abstract Class and Methods rules:**

1)Abstract classes and methods are delcared by using abstract keyword. We cannot create object for abstract classes but we can create objects to subclasses of abstract classes ,