**Selenium WebDriver with Java - Interview Guide**

An Interview Preparation guide for Selenium 3.0 & Java 1.8

Abhay Bharti

**About the Author**

**Abhay Bharti** is a proactive test engineer with over 12 years of experience in test automation (Selenium, Java, Cucumber). He has worked in various spectrum of IT field including telecom, finance etc. As a result, he has been involved in planning, development, implementation, integration and testing of multiple applications, including multi-tiered, standalone, distributed, cloud based application. He currently resides in Bangalore,India. He enjoy spending time with his wife Prabha & daughter shanvi.

**Table of Contents**

Contents

[1 Chapter: Questions on Java Concepts 8](#_Toc6567715)

[1.1 Explain Access Modifier explanation or What are the different access modifiers available in Java? 8](#_Toc6567716)

[1.2 Explain data types in java 9](#_Toc6567717)

[1.3 What is an Array 12](#_Toc6567718)

[1.4 Difference between Array and ArrayList? 15](#_Toc6567719)

[1.5 Why do we need Maven? What are the advantages? 16](#_Toc6567720)

[1.6 What is Maven project phase? 16](#_Toc6567721)

[1.7 What is Maven Goals 16](#_Toc6567722)

[1.8 What is the difference between HashMap and Hashtable? 17](#_Toc6567723)

[1.9 What is overriding? 18](#_Toc6567724)

[1.10 What is overloading? 18](#_Toc6567725)

[1.11 Explain exception handling 19](#_Toc6567726)

[1.12 What is the use of finally{ } block 21](#_Toc6567727)

[1.13 What is the need of user defined exception? Have you ever created a user defined exception? What is the syntax & how to do it? 25](#_Toc6567728)

[1.14 What is difference between throw and throws? 26](#_Toc6567729)

[1.15 Can value be key-value in a hashtable 26](#_Toc6567730)

[1.16 What is difference between StringBuffer, StringBuilder and String classes? 27](#_Toc6567731)

[1.17 What is garbage collector in java 27](#_Toc6567732)

[1.18 Can we achieve multiple inheritance in java? 28](#_Toc6567733)

[1.19 Final and finally keyword difference 28](#_Toc6567734)

[1.20 What is the difference between set and list 28](#_Toc6567735)

[1.21 When to use List and Set in Java 28](#_Toc6567736)

[1.22 What is arrayList and how it works? 29](#_Toc6567737)

[1.23 Can we have multiple catches after the single try? 31](#_Toc6567738)

[1.24 How to iterate ArrayList? 31](#_Toc6567739)

[1.25 What is hashmap and how it works? 33](#_Toc6567740)

[1.26 Which version control tool you are using? Or Which version control tool you used? What are its usage? 35](#_Toc6567741)

[1.27 How you are merging your code to git? 35](#_Toc6567742)

[1.28 What all commands you used in git? 36](#_Toc6567743)

[1.29 How to declare Interface 37](#_Toc6567744)

[1.30 Why constructor cannot be inherited in java? 37](#_Toc6567745)

[1.31 What is linked list? 37](#_Toc6567746)

[1.32 When do we get null pointer exception? 38](#_Toc6567747)

[1.33 What is default constructor 38](#_Toc6567748)

[1.34 What is constructor 39](#_Toc6567749)

[1.35 Why Constructors are not inherited in Java? 39](#_Toc6567750)

[1.36 Can we overload static methods? 40](#_Toc6567751)

[1.37 Can we Override static methods in java? 40](#_Toc6567752)

[1.38 What is difference between Interface and abstract class 41](#_Toc6567753)

[1.39 What is difference between Abstract Class and Concrete Class 42](#_Toc6567754)

[1.40 What is “this” keyword in java? 42](#_Toc6567755)

[1.41 What is “super” keyword in java? 43](#_Toc6567756)

[1.42 What is wrapper class 43](#_Toc6567757)

[**2** **Chapter: Algorithmic/Data structure problem** 44](#_Toc6567758)

[2.1 Write a program to print the number pattern given below 44](#_Toc6567759)

[2.2 How to reverse a String in Java? 46](#_Toc6567760)

[2.3 How to reverse words in a String sentence 47](#_Toc6567761)

[2.4 Find duplicate characters in the provided String and also find the count of each duplicate character. 48](#_Toc6567762)

[2.5 Replace substring with another string in a string 50](#_Toc6567763)

[2.6 How To Find The Largest Value From The Given Array 50](#_Toc6567764)

[2.7 How to check if given number is prime 51](#_Toc6567765)

[2.8 Write a program to print Fibonacci Series 52](#_Toc6567766)

[**3** **Chapter: Question on Selenium WebDriver** 54](#_Toc6567767)

[3.1 What is Selenium WebDriver 54](#_Toc6567768)

[3.2 What is StaleElementException? 55](#_Toc6567769)

[3.3 What are the programming languages supported by Selenium WebDriver? 55](#_Toc6567770)

[3.4 How can you send text input to a focused element? 55](#_Toc6567771)

[3.5 What are locators, different types of locators that can be used in Selenium and their priorities? 55](#_Toc6567772)

[3.6 When should I use Selenium Grid? 56](#_Toc6567773)

[3.7 How do I launch the browser using WebDriver? 56](#_Toc6567774)

[3.8 What are the different types of drivers available in WebDriver? Or What are the browsers supported by Selenium WebDriver? 57](#_Toc6567775)

[3.9 What functions/methods can you use to wait for a page to load, or some element on the page to show up? 58](#_Toc6567776)

[3.10 Without using sendkeys how will you type text in a text box? Is there any method or class in Java? 58](#_Toc6567777)

[3.11 How do you deal with frame elements in Selenium on a page? Or How to handle frames in WebDriver? 59](#_Toc6567778)

[3.12 Why do we use headless drivers? How can you visually investigate test failure when using headless drivers? 59](#_Toc6567779)

[3.13 What is Page Object Design Pattern, explain with example 59](#_Toc6567780)

[3.14 How Selenium WebDriver works? OR What is architecture of Selenium WebDriver? OR Why do we require browser drivers? OR What is WebDriver hierarchy? OR Explain hierarchy of Classes and interfaces in Selenium WebDriver 61](#_Toc6567781)

[3.15 What is WebElement interface hierarchy? 66](#_Toc6567782)

[3.16 Why did you select Selenium WebDriver for front end automation of your application? OR What factors you will take into consideration before deciding a tool to automate front end of your application? 70](#_Toc6567783)

[3.17 How to assert title of the web page? 71](#_Toc6567784)

[3.18 How to capture screen-shot in Selenium WebDriver ? 72](#_Toc6567785)

[3.19 What are the limitations of Selenium WebDriver? 72](#_Toc6567786)

[3.20 What is the super interface of WebDriver? 73](#_Toc6567787)

[3.21 Is WebDriver a class or interface? 73](#_Toc6567788)

[3.22 Is FirefoxDriver a class or interface? 73](#_Toc6567789)

[3.23 What is the use of JavaScriptExecutor? 73](#_Toc6567790)

[3.24 How PageFactory helps you in achieving Page Object Model design? 74](#_Toc6567791)

[3.25 What are the types of waits available in Selenium WebDriver? Explain usage of each type of waits with scenarios or How many wait commands are in Selenium WebDriver 74](#_Toc6567792)

[3.26 How to input text in the text box using Selenium WebDriver? Or How to type into a text box using Selenium? 77](#_Toc6567793)

[3.27 How to input text in the text box without calling the sendKeys()? 77](#_Toc6567794)

[3.28 How to clear the text in the text box using Selenium WebDriver? 78](#_Toc6567795)

[3.29 How to get a text of a web element? Or How can we get a text from a web element using Selenium? 78](#_Toc6567796)

[3.30 How to get an attribute value using Selenium WebDriver? 79](#_Toc6567797)

[3.31 How to click on a hyperlink using Selenium WebDriver? Or How to click on a hyperlink using Selenium WebDriver? 79](#_Toc6567798)

[3.32 How to submit a form using Selenium WebDriver? 79](#_Toc6567799)

[3.33 How to press ENTER key on text box In Selenium WebDriver? 80](#_Toc6567800)

[3.34 How to fetch the current page URL in Selenium? 80](#_Toc6567801)

[3.35 How can we maximize browser window in Selenium? 80](#_Toc6567802)

[3.36 How to delete cookies in Selenium? 80](#_Toc6567803)

[3.37 What are the ways to refresh a browser using Selenium WebDriver? 80](#_Toc6567804)

[3.38 What is the difference between driver.getWindowHandle() and driver.getWindowHandles() in Selenium WebDriver? 81](#_Toc6567805)

[3.39 What is the difference between driver.close() and driver.quit() methods? 81](#_Toc6567806)

[3.40 What is the difference between driver.findElement() and driver.findElements() commands? Or How do you use findElement() and findElements()? 81](#_Toc6567807)

[3.41 How to select a value in a dropdown? How to handle a drop-down field and select a value from it using Selenium? 82](#_Toc6567808)

[3.42 How to handle Ajax calls in Selenium WebDriver? 82](#_Toc6567809)

[3.43 What are the advantages of Page Object Model Framework? 83](#_Toc6567810)

[3.44 How can you use the Recovery Scenario in Selenium WebDriver? 84](#_Toc6567811)

[3.45 What is desired capabilities? 84](#_Toc6567812)

[3.46 How to delete Browser Cookies with Selenium WebDriver? 85](#_Toc6567813)

[3.47 Explain the different exceptions in Selenium WebDriver 85](#_Toc6567814)

[3.48 Write a code to wait for an alert to appear 88](#_Toc6567815)

[3.49 What is use of Select class of Selenium WebDriver? What are its limitations? Or How to handle a dropdown in Selenium WebDriver? How to select a value from dropdown? 89](#_Toc6567816)

[3.50 How to send ALT/SHIFT/CONTROL key in Selenium WebDriver? 91](#_Toc6567817)

[3.51 How to set the size of browser window using Selenium? 92](#_Toc6567818)

[3.52 How to switch to a new window (new tab) which opens up after you click on a link? 92](#_Toc6567819)

[3.53 How do you upload a file using Selenium WebDriver? 93](#_Toc6567820)

[3.54 How do you find all clickable links in a page? 93](#_Toc6567821)

[3.55 Which library you used to read data from excel in Selenium WebDriver scripts? 94](#_Toc6567822)

[3.56 Challenges to run Selenium Scripts with IE Browser 94](#_Toc6567823)

[3.57 What is Action interface & Actions class 97](#_Toc6567824)

[3.58 What are the Challenges you faced in Selenium automation? How did you solve them? 98](#_Toc6567825)

[3.59 What are best practices for creating a UI test automation framework? 98](#_Toc6567826)

[**4** **Chapter – Junit / TestNG Framework** 99](#_Toc6567827)

[**5** **Chapter – Cucumber Questions** 99](#_Toc6567828)

[5.1 What is difference between scenario and scenario outline 99](#_Toc6567829)

[**6** **Chapter – Question on API Testing** 100](#_Toc6567830)

[6.1 What is API Testing 100](#_Toc6567831)

[6.2 What are different HTTP methods for REST API? 100](#_Toc6567832)

[6.3 What are different response status code & their meaning? 101](#_Toc6567833)

# Chapter: Questions on Java Concepts

## Explain Access Modifier explanation or What are the different access modifiers available in Java?

Access modifier or access specifies are keywords in Java that set the accessibility of classes, methods, and other members. There are four types of Access Control Modifiers:

* **default:** When no modifier is used then member variables and methods are accessible in same class or in any other class in same package. Default Access Control Modifiers have the access control/Scoping will be visible only to code inside the class as well as in the same package
* **private:** The scope of private access modifier is only within the classes.

Note: Class or Interface cannot be declared as private

* **protected:** The scope of protected access modifier is within a package and also outside the package through inheritance only. Accessible in same package and also in subclass in some other package

Note: Class cannot be declared as protected

* **public:** The scope of public access modifier is everywhere. It has no restrictions. Data members, methods and classes that declared public can be accessed from anywhere. It is accessible to any class where ever it is inherited or imported

Order of Access Control specifies, from most to least order of Access Control Specifies.

Public -> Protected -> Default -> Private

## Explain data types in java

Data types defines the type and size of the variable and the kinds of operations it supports. There are two types of data types available in Java:

1. Primitive type
2. Non – Primitive type

**Boolean**

**Floating Type**

**Integer**

**Float**

**Double**

**byte**

**short**

**int**

**long**

**char**

**Data Types**

**Primitive**

**Non - Primitive**

**Array Type**

**Interface Type**

**Class Type**

**Numeric**

**Primitive Data Types**

Java supports eights types data types, primitives are the one which are already defined in the language and associated with a keyword. List is as follows -

|  |  |
| --- | --- |
| **Data Types** | **Description** |
| **boolean** | * Used to save two possible values - true and false * Default value is false * Useful to set flags to evaluate true/false conditions * Syntax : boolean flag = true; |
| **Float** | * Used to save decimal number * Default value is 0.0f * It’s capacity is 4 bytes * Syntax : float val = 12.5f; |
| **double** | * Used to save decimal number * Default value is 0.0d * It’s capacity is 8 bytes * Syntax : double val = 45.34; |
| **Byte** | * Used to store numeric values * Default value is 0 * Allows value range of -128 to 127 * it‘s capacity is 1 byte * Syntax : byte val = 50; |
| **short** | * Used to store numeric values, it is bigger than byte but smaller than int * Default value is 0 * Allows value range of -32768 to 32767 * it‘s capacity is 2 byte * Syntax : short val = 60; |
| **int** | * Used to store numeric values * Default value is 0 * Allows value range of -231 to 231-1 * it‘s capacity is 4 byte * Syntax : int val = 1; |
| **Long** | * Used to store numeric values * Default value is 0L * Allows value range of -263 to 263-1 * it‘s capacity is 8 byte * Syntax : long val = 60 |
| **Char** | * Used to store unicode character * It’s capacity is 2 bytes * Allows the value range of '\u0000' (or 0) to 65535 * Syntax : char val = 'C'; |

**Non Primitive Data Types**

It is also known as Reference types or reference variables. Reference variable are used to access the objects. User can create reference variables using constructors of classes. Default value of reference variables is null.

Syntax – ClassName RefObj = new ClassName();

### 

## What is an Array

Arrays are objects in Java that store multiple variables of the same type. It can hold ether primitives or object references, but the array itself will always be an object on the heap, even if the array is declared to hold primitive elements. Each Array has a size and it can’t take value more than the size.

In order to work on array, we need to know three things -

1. **How to make an array reference variable ( Declare)**

Syntax :

<Data Type> <array name> []; //square bracket can be at left/right

of the identifier

Or

<Data Type>[] <array name> ;

Let’s create array of an integer

Example :

int arrayOne[]; //an array is created with name arrayOne

Or

Int [] arrayOne; //an array is created with name arrayOne

1. **How to make an array object (Construct)**

Syntax :

<Data Type> <name> [] = new <Data Type>[size of array];

Example :

int arrayOne[] = new int[3]; //an array is create having size 3

1. **How to populate the array with elements (initialize)**

We can initialize array in following ways, Assign value to each index one by one

arrayOne[0] = 1; //Assigned value 1 at 0th position

arrayOne[1] = 2; //Assigned value 2 at 1st position

arrayOne[2] = 3; //Assigned value 3 at 2nd position

Another ways to initialize array is

int arrayOne ={1,5,2}; //initialization is done automatically

Example code -

*public class ArrayExample {*

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

*// Step 1: declaration:*

*int[] arrayOne;*

*// Step 2: Array creation:*

*arrayOne = new int[3];*

*// Step 3: Array initialization:*

*arrayOne[0] = 1;*

*arrayOne[1] = 2;*

*arrayOne[2] = 3;*

*// Let’s see how many elements are present in array*

*System.out.println("size of array is ===" + arrayOne.length);*

*// Printing individual value using index example lets print value*

*System.out.println("Value present on 1st place is " + arrayOne[1]);*

*// Printing all values of array using index*

*for (int i = 0; i <= arrayOne.length; i++) {*

*System.out.println("Value at index= " + i + " is "*

*+ arrayOne[i]);*

*}*

*// Second type of array initialization*

*int[] array1 = {2, 4, 5, 6, 10, 12, 13};*

*System.out.println("size of array is ===" + arrayOne.length);*

*// example of taking the max value of the size of array*

*for (int j = 0; j < array1.length; j++) {*

*System.out.println("Value at index= " + j + " is "*

*+ array1[j]);*

*}*

*}*

*}*

**Multi-dimensional** Array are array of arrays. A two dimensional array of int is really an object of type int array, with element in that array holding a reference to another int array.

Syntax to create multi dimensional array -

int arrayTwo [] [] = new int [3][3]; //two dimensional array

int arrayThree [][][]= new int[2][3][5]; // three dimensional array

Example code -

public class ArrayExample2D {

public static void main(String[] args) {

System.out.println("===2 Dimensional Array Example===");

// Step 1: declaration:

int array2D[][] = null;

// Step 2: Array creation:

array2D = new int[2][2];

// Step 3: Initialization:

array2D[0][0] = 2;

array2D[0][1] = 3;

array2D[1][0] = 4;

array2D[1][1] = 5;

//Printing all values

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

System.out.println("Value at index= " + i + j + " is "

+ array2D[i][j]);

}

}

System.out.println("====3 Dimensional Array Example===");

int[][][] array3D = {{{1, -2, 3},{2, 3, 4}},{{-4, -5, 6, 9},{1},{2, 3}}};

// for..each loop to iterate through elements of 3d array

for (int[][] array2Dtmp: array3D) {

for (int[] array1D: array2Dtmp) {

for(int item: array1D) {

System.out.println(item);

}

}

}

}

}

## Difference between Array and ArrayList?

1. ArrayList is type-safe while Array is not.
2. Array can be defined multi-dimensional while ArrayList can be one dimensional.
3. Read & Insertion of data takes constant time in both (Up to initial size of ArrayList) if we are calling add() and get() in ArrayList. But automatic resizing in ArrayList might cause slowness in insertion of data.
4. Length: Array uses length variable to find actual size or length of array, while ArrayList uses size() function to find the length.
5. Array size is fixed at the time of creation while ArrayList size is dynamic

### 

## Why do we need Maven? What are the advantages?

Maven is a build management tool that is based on POM (project object model). It is used for projects build, dependency and documentation. It simplifies the build process. It helps to manage- Builds, Documentation, Reporting, SCMs, Releases, Distribution.

## What is Maven project phase?

A Maven phase represents a stage in the Maven build lifecycle. Each phase is responsible for a specific task. Here are some of the most important phases in the default build lifecycle:

* clean
* validate
* compile
* test
* package
* verify
* install
* site
* deploy

## What is Maven Goals

When we run a phase – all goals bound to this phase are executed in order.

Here are some of the phases and default goals bound to them:

* compiler:compile – the compile goal from the compiler plugin is bound to the compile phase
* compiler:testCompile is bound to the test-compile phase
* surefire:test is bound to test phase
* install:install is bound to install phase
* jar:jar and war:war is bound to package phase

### 

## What is the difference between HashMap and Hashtable?

1. HashMap is non synchronized whereas Hashtable is synchronized.
2. Hashtable does not allow null keys or values while HashMap allows one null key and any number of null values.
3. HashMap is unsorted and unordered map while Hashtable is order map
4. Both supports unique keys & duplicate values.

Example Code -

*import java.util.HashMap;*

*import java.util.Hashtable;*

*import java.util.Map;*

*public class hasstableToHashmap {*

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

*//----------hashtable -------------------------*

*Hashtable<Integer, String> htEx = new Hashtable<Integer, String>();*

*htEx.put(101, "akash");*

*htEx.put(101, "Vikash"); //does not allow duplicate key*

*htEx.put(102, "Raji");*

*htEx.put(103, "Rahul");*

*htEx.put(104, "Rahul"); //allows duplicate values*

*System.out.println("-------------Print Hash table values--------------");*

*for (Map.Entry m : htEx.entrySet()) {*

*System.out.println(m.getKey() + " " + m.getValue());*

*}*

*//----------------hashmap--------------------------------*

*HashMap<Integer, String> hmEx = new HashMap<Integer, String>();*

*hmEx.put(100, "Ajit");*

*hmEx.put(100, "Ajit"); //does not allow duplicate key*

*hmEx.put(104, "Ajit"); //hash map allows duplicate values*

*hmEx.put(101, "Vikay");*

*hmEx.put(102, "Rahul");*

*System.out.println("-----------Print Hash map values-----------");*

*for (Map.Entry m : hmEx.entrySet()) {*

*System.out.println(m.getKey() + " " + m.getValue());*

*}*

*}*

*}*

## What is overriding?

Overriding is related to run-time polymorphism. A subclass (or derived class) provides a specific implementation of a method in super class (or base class) at runtime.

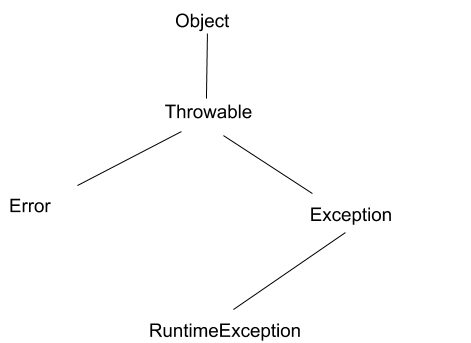
## What is overloading?

Overloading is related to compile time (or static) polymorphism. This feature allows different methods to have same name, but different signatures, especially number of input parameters and type of input parameters.

## Explain exception handling

An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e. at run time that disrupts the normal flow of the program’s instructions. When an exceptional event occurs, an exception is thrown. The code that is responsible for doing something about exception is called an “exception handle” and it catches the thrown exception.

Exception class diagram

****

Example Code -

*public class ExceptionExample {*

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

*try{*

*int data=25/0;*

*System.out.println(data);*

*}*

*catch(ArithmeticException e){*

*System.out.println(e);*

*}*

*System.out.println("rest of the code...");*

*}*

*}*

*Output:*

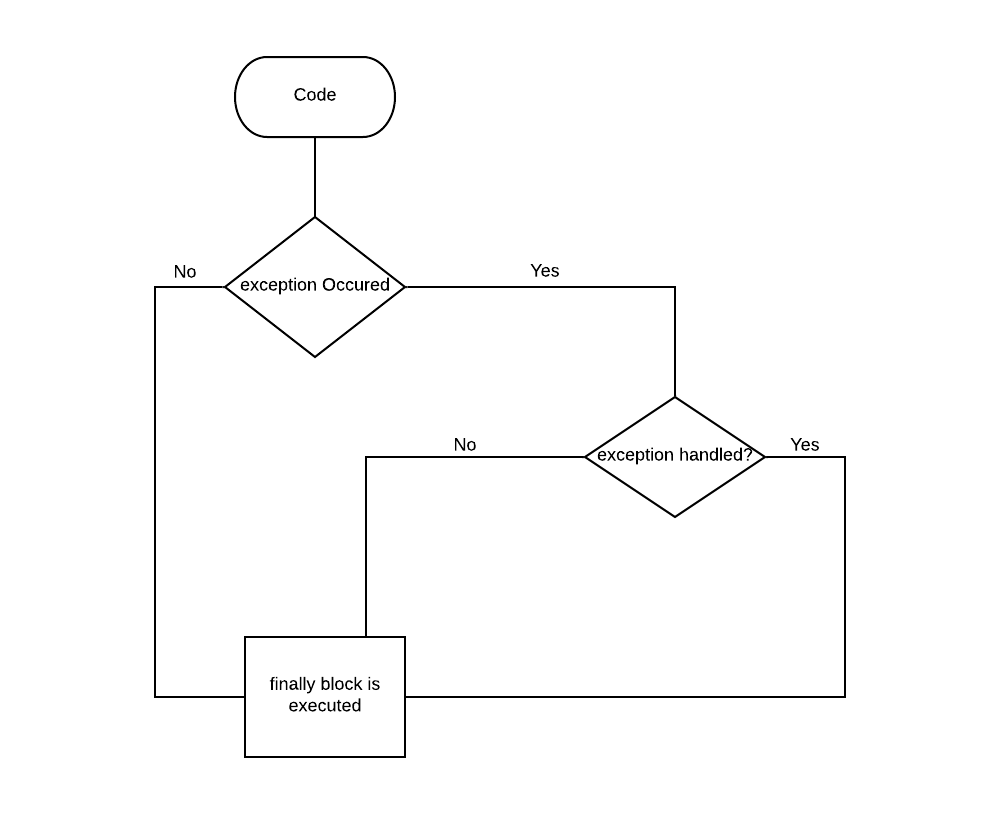
*java.lang.ArithmeticException: / by zero*

*rest of the code...*

## What is the use of finally{ } block

Java finally block is always executed whether exception is handled or not. It is used to execute important code such as closing connection, stream etc. Java finally block follows try or catch block.

Execution flow diagram for finally block



Let’s see finally block behavior for different scenario -

**Case 1 :** **finally block where exception doesn't occur**

*public class FinallyCase1 {*

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

*try{*

*int number=30/5;*

*System.out.println(number);*

*}*

*catch(NullPointerException e){*

*System.out.println(e);*

*}*

*finally{*

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

*}*

*System.out.println("rest of the code...");*

*}*

*}*

*Output:6*

*finally block is always executed*

*rest of the code...*

**Case 2 : finally block where exception occurs and not handled**

*public class FinallyCase2 {*

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

*try{*

*int data=25/0;*

*System.out.println(data);*

*}*

*catch(NullPointerException e){*

*System.out.println(e);*

*}*

*finally{*

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

*}*

*System.out.println("rest of the code...");*

*}*

*}*

*Output:finally block is always executed*

*Exception in thread "main" java.lang.ArithmeticException: / by zero*

*At javalearning.chapter6exceptionexample.FinallyCase2.main(FinallyCase2.java:6)*

**Case 3 : finally block where exception occurs and handled**

*public class FinallyCase3 {*

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

*try{*

*int data=25/0;*

*System.out.println(data);*

*}*

*catch(ArithmeticException e){*

*System.out.println(e);*

*}*

*finally{*

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

*}*

*System.out.println("rest of the code...");*

*}*

*}*

*Output:java.lang.ArithmeticException: / by zero*

*finally block is always executed*

*rest of the code...*

## What is the need of user defined exception? Have you ever created a user defined exception? What is the syntax & how to do it?

Java provides ways to create user defined or custom exceptions that throws our own exception using throw keyword. This can be done by extending the exception class. This is helpful for errors which are not handled in Java.

Code to create user defined exception

public class UserDefinedException extends Exception {

int counter;

UserDefinedException(int errorCount) {

counter = errorCount;

}

public String toString(){

return ("Exception Counter = "+counter) ;

}

}

Code to test user defined exception

public class TestException {

public static void main(String args[]){

try{

throw new UserDefinedException(100);

*// throw is used to create a new exception and throw it.*

}

catch(UserDefinedException e){

System.*out*.println(e) ;

}

}

}

Output :

Exception Counter = 100

### 

## What is difference between throw and throws?

|  |  |
| --- | --- |
| **Throw** | **throws** |
| throw keyword is used to explicitly throw an exception. | throws keyword is used to declare an exception. |
| Checked exception cannot be propagated using throw only. | Checked exception can be propagated with throws. |
| Throw is followed by an instance. E.g. -  throw new ArithmeticException("sorry"); | Throws is followed by class. E.g. -  void method()throws IOException |
| Throw is used within the method. E.g. -  void m(){  throw new ArithmeticException("not valid");  } | Throws is used with the method signature. E.g. -  void m()throws ArithmeticException{  //method code  } |
| You cannot throw multiple exceptions. | You can declare multiple exceptions e.g.  public void method()throws IOException,SQLException |

## Can value be key-value in a hashtable

Yes

## What is difference between StringBuffer, StringBuilder and String classes?

|  |  |  |
| --- | --- | --- |
| **String** | **StringBuilder** | **StringBuffer** |
| * String is immutable means once assigned cannot be changed. * String cannot be used by two threads simultaneously. * String concat + operator internally uses StringBuffer or StringBuilder class | * StringBuilder is not thread safe. * It is a mutable class * For String manipulations in non-multi threaded environment, we should use StringBuilder else use StringBuffer class. | * StringBuffer is synchronized that is it is thread safe. * StringBuffer and mutable classes. |

## What is garbage collector in java

The garbage collector is a program which runs on the Java Virtual Machine which gets rid of objects which are not being used by a Java application anymore. It is a form of automatic memory management. It is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects. It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory. It is automatically done by the garbage collector(a part of JVM) so we don't need to make extra efforts.

## Can we achieve multiple inheritance in java?

No, We cannot extend two objects, but we can implement two interfaces. Interfaces don't simulate multiple inheritance. Java creators considered multiple inheritance wrong, so there is no such thing in Java. If you want to combine the functionality of two classes into one - use object composition.

Note – Till Java 7, multiple inheritance was not supported. In Java 8, multiple inheritance is supported using default method.

## Final and finally keyword difference

**final** keyword can be used with class method and variable. A final class cannot be instantiated, a final method cannot be overridden and a final variable cannot be reassigned.

**finally** keyword is used to create a block of code that follows a try block. A finally block of code always executes, whether or not an exception has occurred.

## What is the difference between set and list

1. List allows duplicates while Set doesn't allow any duplicate.
2. List is an Ordered Collection while Set is an unordered Collection. List maintains insertion order of elements, means any element which is inserted before will go on lower index than any element which is inserted after. Set in doesn't maintain any order.
3. Popular implementation of List interface in Java includes ArrayList, Vector and LinkedList. Popular implementation of Set interface includes HashSet, TreeSet and LinkedHashSet.

## When to use List and Set in Java

List and Set, both are derived from Collection Interface. If you need to maintain insertion order of object and you collection can contain duplicates then List is a way to go. On the other hand if your requirement is to maintain unique collection without any duplicates than Set is the way to go.

## What is arrayList and how it works?

1. ArrayList supports dynamic arrays that can grow as needed.
2. ArrayList class is in java.util package
3. The ArrayList class extends AbstractList and implements the List interface
4. ArrayList supports dynamic arrays that can grow as needed
5. ArrayList is initialized by a size, however the size can increase if collection grows or shrunk if objects are removed from the collection.
6. It maintain order of elements
7. ArrayList allows to randomly access the list.
8. ArrayList can not be used for primitive types, like int, char, etc. We need a wrapper class for such cases.

**method supported by ArrayList** *-*

**boolean addAll(Collection c)** - Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator. Throws NullPointerException, if the specified collection is null.

**boolean addAll(int index, Collection c)** - Inserts all of the elements in the specified collection into this list, starting at the specified position. Throws NullPointerException if the specified collection is null.

**int lastIndexOf(Object o)** - Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.

**removeRange(int fromIndex, int toIndex)** - Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.

Example code -

public class ArrayListExample {

public static void main(String args[]){

List<String> c = new ArrayList<>(); //create a new arrayList

c.add("oslo"); *// add() method adds element to the end arrayList*

c.add("paris");

c.add("Rome");

*//returns index of element*

int index = c.indexOf("paris");

System.*out*.println(c + " "+ index);

*//add element at index point & shifts the remaining element back*

*(e.g. what was index, is not index+1)*

c.add(index, "London");

System.*out*.println(c);

*// returns true/false, whether element is in arraylist*

System.*out*.println(c.contains("paris"));

*//returns object at index*

System.*out*.print(c.get(index));

*// remove first occurence of the object & shifts later elements*

*towards the beginning one space*

c.remove("paris");

*// remove element at that index & shifts later elements towards the*

*beginning one space*

c.remove(index);

*//return number of elements from arrayList*

System.*out*.println("size() : "+c.size());

*//remove all element from arraylist*

c.clear();

System.*out*.println(c);

}

}

## Can we have multiple catches after the single try?

Yes

## How to iterate ArrayList?

We can iterate arrayList in three ways, code example is as follows -

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Iterator;

**import** java.util.List;

**public class** ArrayListTraversal {

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

*// initializing ArrayList*

List<Integer> numbers = **new** ArrayList<>();

numbers.add(1);

numbers.add(2);

numbers.add(3);

numbers.add(4);

numbers.add( 5);

numbers.add( 6);

numbers.add( 7);

numbers.add(8);

System.***out***.println(**"Approach 1. Iterating arrayList using for loop : "**);

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

System.***out***.print(numbers.get(i) + **" "**);

}

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

System.***out***.println(**"Approach 2. Using For Each Loop :"**);

**for** (Integer i : numbers) {

System.***out***.print(i + **" "**);

}

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

System.***out***.println(**"Approach 3. Looping ArrayList using Iterator :"**);

Iterator it = numbers.iterator();

**while** (it.hasNext()) {

System.***out***.print(it.next() + **" "**);

}

}

}

Output :

Iterating arrayList using for loop :

1 2 3 4 5 6 7 8

Using For Each Loop :

1 2 3 4 5 6 7 8

Looping ArrayList using Iterator :

1 2 3 4 5 6 7 8

### 

## What is hashmap and how it works?

* It provides the basic implementation of Map interface.
* It stores the data in (Key, Value) pairs. To access a value one must know its key.
* It is a part of java.util package.
* HashMap extends an abstract class AbstractMap which also provides an incomplete implementation of Map interface.
* HashMap doesn’t allow duplicate keys but allows duplicate values. That means A single key can’t contain more than 1 value but more than 1 key can contain a single value.
* HashMap allows null key also but only once and multiple null values.
* It is un-ordered map.
* It is unsynchronized.

*Hashmap diagram*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 |

|  |
| --- |
| 110 |
| {“Selenium”} |
| 10 |
| null |

Example code -

**package** javalearning.chapter6collectionsex;

**import** java.util.HashMap;

**import** java.util.Map;

**public class** HashMapExample {

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

HashMap<String, Integer> map = **new** HashMap<>();

*print*(map);

map.put(**"Ajay"**, 10);

map.put(**"Alex"**, 30);

map.put(**"Vijay"**, 20);

System.***out***.println(**"Size of map is: "** + map.size());

*print*(map);

**if** (map.containsKey(**"Vijay"**)) {

Integer a = map.get(**"Vijay"**);

System.***out***.println(**"value for key is:- "** + a);

}

map.clear();

*print*(map);

}

**public static void** print(Map<String, Integer> map) {

**if** (map.isEmpty()) {

System.***out***.println(**"map is empty"**);

} **else** {

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

}

}

}

**Output**:

map is empty

Size of map is: 3

{Alex=30, Vijay=20, Ajay=10}

value for key is:- 20

map is empty

## Which version control tool you are using? Or Which version control tool you used? What are its usage?

In my project, we use Git.

## How you are merging your code to git?

1. Open Git Bash the terminal
2. Change the current working directory to your local project
3. Check out the branch you want to merge to
4. If there are conflicts, resolve them
5. Commit the merge
6. Push the merge to your GitHub repository

## What all commands you used in git?

List of commands, that I use frequently :

* **git --version** :- git version installed on your machine
* **git config --global user.name** :- shows logged in user name
* **git config --global**
* **git init foldername** :- create git repository in foldername
* **git init** :- create git repository in current foldername
* **git add filename** :- Ready to save (add to staging area) STEP 1
* **git commit** :- Ready to save (commit) STEP 2
* **git commit -a -m “my message"** : - Commit everything and save message
* **git status** :- Check status of work
* **git log** :- View all commits that have been done
* **git diff commitIDv1 commitIDv2** :- Check the difference between two versions
* **git branch branchname** :- Create a new branch
* **git checkout branchname** :- Moving inside a branch
* **git checkout -b branchname** :- Create a branch and move to it
* **git branch** :- View all branches
* **git branch -D branchname** :- Delete branch
* **git merge branchname** :- Merge a branch into current folder
* **git clone location new\_location** :- To clone a project
* **git remote** :- To view all the remote clones
* **git push** :- Update a remote repository with changes
* **git push location branchname** :- Update a remote repository with new branch
* **git pull location branchname (e.g. git pull origin master)** :- Receive an update on a file from remote repository
* **git stash** :- You are working on a new feature, suddenly you have to work for few hours on bug fix. You can not commit your partial code and also cannot throw away your changes. This provides a temporary space, where you can store your partial changes and later commit it.
* **git stash pop** :- This remove the changes from the stack and place them in the current working directory.
* **git reset --hard** :- reset all local changes & makes local repo copy of remote.

## How to declare Interface

interface SampleInterface {

// declare constant fields

// declare methods

}

## Why constructor cannot be inherited in java?

* Constructors are special and have same name as class name. So if constructors were inherited in child class then child class would contain a parent class constructor which is against the constraint that constructor should have same name as class name.
* Now suppose if constructors can be inherited then it will be impossible to achieving encapsulation. Because by using a super class’s constructor we can access/initialize private members of a class.
* A constructor cannot be called as a method. It is called when object of the class is created so it does not make sense of creating child class object using parent class constructor notation. i.e. Child c = new Parent();

## What is linked list?

Linked List is linear data structure where the elements are not stored in contiguous locations and every element is a separate object with a data part and address part. The elements are linked using pointers and addresses. Each element is known as a node. Due to the dynamicity and ease of insertions and deletions, they are preferred over the arrays. It also has few disadvantages like the nodes cannot be accessed directly instead we need to start from the head and follow through the link to reach to a node we wish to access.

In Java, LinkedList class implements the list interface.

*// Creating object of class linked list*

*LinkedList<String> object =* ***new*** *LinkedList<String>();*

*// Adding elements to the linked list*

*object.add(****"A"****); //add ‘A’*

*object.addLast(****"C"****); //add ‘C’ as last node*

*object.addFirst(****"D"****); //add ‘D’ as first node*

*object.add(2,* ***"E"****); //add ‘E’ at 2nd place*

*object.add(****"F"****); // add ‘F’ at the end*

*object.add(****"G"****); add ‘G’ at the end*

*System.****out****.println(****"Linked list : "*** *+ object); //print linkedlist*

***// Removing elements from the linked list***

*object.remove("B"); //remove ‘B’*

*object.remove(3); //remove 3rd node*

*object.removeFirst(); //remove first node*

*object.removeLast(); //remove last node*

*System.out.println("Linked list after deletion: " + object);*

## When do we get null pointer exception?

NullPointerException is a RuntimeException. NullPointerException is thrown when program attempts to use an object reference that has the null value. This exception can happen when you try any of following :

* Invoking a method from a null object.
* Accessing or modifying a null object’s field.
* Taking the length of null, as if it were an array.
* Accessing or modifying the slots of null object, as if it were an array.
* Throwing null, as if it were a Throwable value.

## What is default constructor

A constructor without parameters is known as default constructor. Default constructor is automatically generated by compiler. If you have typed in a constructor with arguments, you won’t have no args constructor unless you typed in yourself.

## What is constructor

A constructor in Java is a block of code similar to a method that’s called when an instance of an object is created. Here are the key differences between a constructor and a method:

* A constructor doesn’t have a return type.
* The name of the constructor must be the same as the name of the class.
* Unlike methods, constructors are not considered members of a class.
* A constructor is called automatically when a new instance of an object is created.
* Can use any access modifier
* Can’t be inherited by child class
* Interface does not have constructor
* Abstract class can have constructor and they get executed when a concrete subclass is instantiated.

## Why Constructors are not inherited in Java?

Constructor is a block of code that allows you to create an object of class and has same name as class with no explicit return type.

Whenever a class (child class) extends another class (parent class), the sub class inherits state and behavior in the form of variables and methods from its super class but it does not inherit constructor of super class because of following reasons:

* Constructors are special and have same name as class name. So if constructors were inherited in child class then child class would contain a parent class constructor which is against the constraint that constructor should have same name as class name.
* If we define Parent class constructor inside Child class it will give compile time error for return type and consider it a method. But for print method it does not give any compile time error and consider it a overriding method.
* Now suppose if constructors can be inherited then it will be impossible to achieving encapsulation. Because by using a super class’s constructor we can access/initialize private members of a class.
* A constructor cannot be called as a method. It is called when object of the class is created so it does not make sense of creating child class object using parent class constructor notation. i.e. Child c = new Parent();
* A parent class constructor is not inherited in child class and this is why super() is added automatically in child class constructor if there is no explicit call to super or this.

## Can we overload static methods?

The answer is **‘Yes’**. We can have two or more static methods with same name, having differences in input parameters

## Can we Override static methods in java?

No, We can declare static methods with same signature in subclass, but it is not considered overriding as there won’t be any run-time polymorphism. Static methods cannot be overridden because method overriding only occurs in the context of dynamic (i.e. runtime) lookup of methods. Static methods (by their name) are looked up statically (i.e. at compile-time).

## What is difference between Interface and abstract class

|  |  |
| --- | --- |
| **Interface**     * Java interface are implicitly abstract and cannot contains implementations. * Variables declared in a Java interface is by default final. * Members of a Java interface are public by default. * Java interface should be implemented using keyword “implements”; * A Java class can implement multiple interfaces but it can extend only one abstract class. * Interface is absolutely abstract and cannot be instantiated   Note – Interface can contain default method which contains method body. | **Abstract class**     * A Java abstract class can have instance methods that implements a default behavior. * An abstract class may contain non-final variables. * A Java abstract class can have the usual flavors of class members like private, protected, etc.. * A Java abstract class should be extended using keyword “extends”. * A Java abstract class also cannot be instantiated, but can be invoked if a main() exists. * Abstract classes are classes that contain one or more abstract methods. * An abstract method is a method that is declared, but contains no implementation. * Abstract classes may not be instantiated, and require subclasses to provide implementations for the abstract methods. |

## What is difference between Abstract Class and Concrete Class

|  |  |
| --- | --- |
| **Abstract Class** | **Concrete Class** |
| An abstract class is declared using abstract modifier. | A concrete class in Java is a type of subclass, which implements all the abstract method of its super abstract class which it extends to. |
| An abstract class cannot be directly instantiated using the new keyword. | A concrete class can be directly instantiated using the new keyword. |
| An abstract class may or may not contain abstract methods. | A concrete class cannot contain an abstract method. |
| An abstract class cannot be declared as final. | A concrete class can be declared as final. |
| An abstract class cannot be declared as final. | A concrete class is a subclass of an abstract class, which implements all its abstract method. |

## What is “this” keyword in java?

Within an instance method or a constructor, this is a reference to the current object — the object whose method or constructor is being called. You can refer to any member of the current object from within an instance method or a constructor by using this.

Usage of this keyword

* Used to refer current class instance variable.
* To invoke current class constructor.
* It can be passed as an argument in the method call.
* It can be passed as argument in the constructor call.
* Used to return the current class instance.
* Used to invoke current class method (implicitly)

## What is “super” keyword in java?

The super keyword in java is a reference variable that is used to refer parent class objects. The keyword “super” came into the picture with the concept of Inheritance. Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable. Various scenarios of using java super Keyword:

* super is used to refer immediate parent instance variable
* super is used to call parent class method
* super() is used to call immediate parent constructor

## What is wrapper class

A wrapper class wraps (encloses) around a primitive data type and gives it an object appearance. Reasons, why we need Wrapper Classes

* null is a possible value
* use it in a Collection
* Methods that support Object like creation from other types.. like String
* Wrapper classes are immutable, good candidate for hashMap key
* Integer number2= newInteger("55");//String

**Two ways of creating Wrapper Class Instances**

**1. Using a Wrapper Class Constructor**

Integer number= new Integer(55);//int

Integer number2= new Integer("55");//String

Float number3= new Float(55.0);//double argument

Float number4= new Float(55.0f);//float argument

Float number5= new Float("55.0f");//String

Character c1= new Character('C');//Only char constructor

//Character c2 = new Character(124);//COMPILER ERROR

Boolean b= new Boolean(true);//"true" "True" "tRUe" -all String Values give True

//Anything else gives false

Boolean b1= new Boolean("true");//value stored –true

Boolean b2= new Boolean("True");//value stored –true

Boolean b3= new Boolean("False");//value stored –false

Boolean b4= new Boolean("SomeString");//value stored –false

**2. valueOfStatic Methods**

Integer hundred= Integer.valueOf("100");//100 is stored in variable

Integer seven = Integer.valueOf("111", 2);//binary 111 is converted to 7

# **Chapter: Algorithmic/Data structure problem**

## Write a program to print the number pattern given below

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

|  |  |
| --- | --- |
|  |  |

Code to print number pattern

**public class** NumberDisplayPattern {

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

**for** (**int** x = 1; x <= 5; x++) {

**for** (**int** y = 1; y <= x; y++) {

System.***out***.print(y+**" "**);

}

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

}

}

}

### 

## How to reverse a String in Java?

We can reverse string in two ways -

**Solution 1 :**

1. Create a string variable & store string value
2. Find the length of the string.
3. Run a for loop where i = length of string
4. Fetch character from string using charAt() method
5. Append character int new string variable

public class ReverseAString {

public static void main(String args[]){

*/\*\* ------- Reverse String using String -> chatAt() method ------ \*\*/*

String stringToReverse = "Selenium";

StringBuilder stringBuilder = new StringBuilder();

for(int i = stringToReverse.length()-1; i >=0; i-- ){

stringBuilder.append(stringToReverse.charAt(i));

}

System.*out*.println("Reversed String "+ stringBuilder.toString());

}

}

Output:

Reversed String : muineleS

**Solution 2 :**

1. Create a stringBuffer class variable & store string value
2. Call reverse() method & pass string as argument

public class ReverseAString {

public static void main(String args[]){

*/\*\* ----- Reverse String using StringBuffer -> reverse() method ------ \*\*/*

StringBuffer newStr = new StringBuffer("Selenium Example Code"); *//Sample String*

System.*out*.println("Reversed String : "+newStr.reverse()); *//Reversed string*

}

}

Output:

Reversed String : edoC elpmaxE muineleS

**Solution 3 :**

*Using recursion approach*

## How to reverse words in a String sentence

**Solution:**

1. Take the sentence & store in string variable
2. Split the sentence basis of blank space & store in string array
3. Find the length of the array and iterate the array in reverse
4. Store array element in StringBuffer variable
5. Call reverse method
6. Append string value in stringbuffer variable

Example code :

**public class** ReverseWordsInSentence {

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

*//Sample string*

String stringToReverse = **"This is Abhay"**;

*//Split string on blank space & store in String array*

String[] brokenStringOnSpace = stringToReverse.trim().split(**" "**);

*//Create StringBuffer object*

StringBuffer sb = **new** StringBuffer();

*//length of string array*

**int** len = brokenStringOnSpace.**length**;

*//*

**for** (**int** i = len - 1; i >= 0; i--) {

StringBuffer sb1 = **new** StringBuffer(brokenStringOnSpace[i]);

*//reverse word in sentence*

sb.append(sb1.reverse());

*//append blank space in sentence*

sb.append(**' '**);

}

*//print reversed sentence*

System.***out***.println(**"Reversed Sentence :"** + sb.toString());

}

}

## Find duplicate characters in the provided String and also find the count of each duplicate character.

**Solution:**

1. Store String in a variable
2. convert String value in a character array
3. Create one hashmap where each character would act as a key and its count as value. Here we will check the presence of the key in the map and if the key exists we will increase the count of character or if it doesn’t present as the key in the map we will add it
4. Print the key and value pair where the count is > 1

Example code -

**import** java.util.HashMap;

**import** java.util.Map;

**import** java.util.Set;

**public class** FindDuplicateCharacterInString {

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

*//String value*

String strValue = **"BPharatB"**;

*//Create a HashMap object*

Map<Character,Integer > duplicateCharMap = **new** HashMap<Character,Integer>();

*//store string value to character array*

**char**[] stringToCharArray = strValue.toLowerCase().toCharArray();

*//iterate character array*

**for**(Character c : stringToCharArray){

**if**(duplicateCharMap.containsKey(c)){

*//increment count in hashmap*

duplicateCharMap.put(c, duplicateCharMap.get(c)+1);

}**else**

{

*//add in hashmap*

duplicateCharMap.put(c, 1);

}

}

*//printing character and repeat count*

Set<Character> keys = duplicateCharMap.keySet();

**for**(Character ch : keys){

*//if char count is more than 1 then only print*

**if**(duplicateCharMap.get(ch) > 1)

{

System.***out***.println(ch +**" is "**+ duplicateCharMap.get(ch) +**" repeated in String"**);

}

}

}

}

Output :

a is 2 repeated in String

b is 2 repeated in String

## Replace substring with another string in a string

**Solution :**

1. Store value in string variable
2. Use replace method which takes two argument 1. Value to be replaced 2. New value

Example code -

public class ReplaceString {

public static void main(String args[]) {

String str = "I use webdriver. WebDriver is a tool for browser automation.";

String toBeReplaced = "selenium";

String toReplacedWith = "Firefox";

System.*out*.println("Original String :- "+str);

System.*out*.println("Replaced String :- "+str.replace("webdriver","Chrome"));

}

}

## How To Find The Largest Value From The Given Array

**Solution :**

1. Store value in a integer array
2. Store first value from array in variable val
3. Run a for loop on array elements
4. Add a if loop which compares val with array elements
5. If array[i] is greater than val, it is assigned to val else comparison continues with next element

Example code -

**public class** FindLargestValueFromArray {

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

**int**[] arr={20,3,22,9,17,4,23,2};

**int** val=arr[0];

**for**(**int** i=0; i<arr.**length**; i++){

**if**(arr[i] > val){

val=arr[i];

}

}

System.***out***.println(**"Largest value Array is : "**+ val);

}

}

***Solution 2:***

***Using Arrays.stream(arrayName).max***

## How to check if given number is prime

Prime number is a number which is only divisible by 1 and itself. E.g. 2, 3, 5, 7, 11 etc.

**Solution :**

1. Store number in int variable
2. Make sure number is greater than zero
3. Run a for loop, starting point = 2, upto number -1
4. Divide number by i & check if modulus is equal to zero
5. If modulus is equal to zero, number is not prime

Example Code -

**public class** CheckPrimeNumber {

**static boolean** isPrime(**int** number) {

*// Check for zero or negative number*

**if** (number <= 1) {

**return false**;

}

*// Check from 2 to n-1*

**for** (**int** i = 2; i < number; i++) {

**if** (number % i == 0) {

**return false**;

}

}

**return true**;

}

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

**if** (*isPrime*(17))

System.***out***.println(**"Prime Number"**);

**else**

System.***out***.println(**"Not a Prime Number"**);

}

}

## Write a program to print Fibonacci Series

**Solution:**

1. Create three integer variable maxNumber (holds range of fibonacci series), previousNumber (default value = 0),nextNumber (default value = 1)
2. Run for loop upto maxNumber & start @ 1
3. Add previousNumber, nextNumber & assign to integer sum
4. In each iteration, assign second number to the previous number
5. Assign the sum of last two numbers to the next number

**public class** FibonacciNumSeries {

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

*// number of elements you want to generate in the Fibonacci Series*

**int** maxNumber = 10;

**int** previousNumber = 0;

**int** nextNumber = 1;

System.***out***.println(**"Fibonacci Series of "**+maxNumber+**" numbers:"**);

**for** (**int** i = 1; i <= maxNumber; ++i)

{

System.***out***.print(previousNumber+**" "**);

**int** sum = previousNumber + nextNumber;

*//In each iteration, assign second number to the previous number*

previousNumber = nextNumber;

*//Assign the sum of last two numbers to the next number*

nextNumber = sum;

}

}

}

# **Chapter: Question on Selenium WebDriver**

## What is Selenium WebDriver

Selenium WebDriver is an browser automation tool. Using WebDriver libraries you can drive the browser and perform actions like Click, Open URL, Enter Text, Clear Text, Take Screenshot of browser and many more actions. WebDriver works on a very simple principle of Client Server architecture. The communication between the server and client is through HTTP request and response.

### 

## What is StaleElementException?

It is the exception thrown when the invoked element is no longer attached to the DOM for any reason. Stale means old, decayed, no longer fresh. Stale Element means an old element or no longer available element. Assume there is an element that is found on a web page referenced as a WebElement in WebDriver. If the DOM changes then the WebElement goes stale. If we try to interact with an element which is staled then the **StaleElementReferenceException** is thrown.

## What are the programming languages supported by Selenium WebDriver?

* Java
* C#
* Python
* Ruby
* Perl
* JavaScript

## How can you send text input to a focused element?

This can be done by simulating key presses on the focused element. One way is to perform “actions” on the web driver object:

new Actions(webDriver).sendKeys(“some text”).perform();

An alternative way is to switch to the active element first, and send keys to it directly:

webDriver.switchTo().activeElement().sendKeys(“some text”);

## What are locators, different types of locators that can be used in Selenium and their priorities?

At the heart of any automation script is locator which helps to identify correct GUI element. Selenium WebDriver uses locators to find the elements on web page. Various locator strategies used in Selenium are as follows –

* Id
* Name
* Linktext
* TagName
* XPath
* CSS

## When should I use Selenium Grid?

* Selenium Grid gives the flexibility to distribute your test cases for execution.
* It can be used to run multiple instances of Selenium WebDriver on various operating system and browser configurations.
* Reduces test execution time.
* Can perform multi browser testing.
* Can perform multi OS testing.

## How do I launch the browser using WebDriver?

In Selenium WebDriver, we need to call the native methods of the different browsers class to launch them. We have different implementation of WebDriver interface for different browsers like - FirefoxDriver for Firefox browser, ChromeDriver for Google Chrome, InternetExplorerDriver for Internet Explorer etc.

WebDriver driver = new FirefoxDriver();

We are creating a reference variable 'driver' of interface WebDriver, instantiated using 'FireFoxDriver' class. This will launch Firefox browser.

Please note that we can also create a reference variable of type FirefoxDriver like this-

FirefoxDriver driver = new FirefoxDriver();

Earlier approach is a better design practice, it gives flexibility to swith reference to other browser class at run time.

**Launching Firefox Driver**

*//Setting the webdriver.gecko.driver property to its executable's*

*location*

*System.setProperty("webdriver.gecko.driver","path of gecko*

*driver.exe");*

*//Instantiating driver object and launching browser*

WebDriver driver = new FirefoxDriver();

**Launching Chrome Driver**

*//Setting the webdriver.chrome.driver property to its executable's*

*location*

System.*setProperty*("webdriver.chrome.driver", "path of chromedriver.exe");

*//Instantiating driver object and launching browser*

WebDriver driver = ChromeDriver();

**Launching Internet Explorer Driver**

*//Setting the webdriver.ie.driver property to its executable's location*

System.setProperty("webdriver.ie.driver", "path of

IEDriverServer.exe");

*//Instantiating driver object*

WebDriver driver = new InternetExplorerDriver();

**Launching Safari Driver**

*//Creating a driver object referencing WebDriver interface*

WebDriver driver = new SafariDriver();

## What are the different types of drivers available in WebDriver? Or What are the browsers supported by Selenium WebDriver? Or What are the different classes which implement WebDriver?

List of different types of drivers available in Selenium WebDriver -

* FirefoxDriver
* InternetExplorerDriver
* ChromeDriver
* SafariDriver
* OperaDriver
* AndroidDriver
* IPhoneDriver
* HTMLUnitDriver

## Can Selenium Automate Desktop Applications?

No

## What functions/methods can you use to wait for a page to load, or some element on the page to show up?

To wait for a page to load, explicit wait can be used. Explicit waits stall until some specific condition is met. Although *Thread.Sleep()* is a form of hard wait where the thread stalls for a specific duration of time, it may not be the most reliable way to wait for a page to load.

For example, rather than using something like:

*Thread.Sleep(30000);*

… where the thread sleeps for 30 seconds (30,000 milliseconds), the following can be used:

*WebDriverWait wait= new WebDriverWait(webDriver,30);*

*wait.until(ExpectedConditions.visibilityOf(h1));*

… where Selenium waits until the given element is visible on the page, or throws TimeoutException after waiting for 30 seconds.

## Without using sendkeys how will you type text in a text box? Is there any method or class in Java?

Any other way is only to use native Javascript action to enter the value in the text box:

WebDriver driver = new FirefoxDriver();

JavascriptExecutor executor = (JavascriptExecutor)driver;

executor.executeScript("document.getElementById("textbox\_id").value='new value';);

## How do you deal with frame elements in Selenium on a page? Or How to handle frames in WebDriver?

In order to manipulate frame and its content you must switch to it first. This is similar to how you have to switch to a different page before you can interact with it:

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

… where *index* is the zero-based index of the frame. Switching the frame directs all further interactions through the webdriver towards the selected frame. The *frame* can also be located using name, element ID and reference to already located elements.

To switch back to the browser window, the *defaultContent* method is be used:

*driver.switchTo().defaultContent();*

## Why do we use headless drivers? How can you visually investigate test failure when using headless drivers?

Headless drivers are typically used in continuous integration (CI) setups. Headless drivers, such as PhantomJS, provide all standard web browser functionalities, but run in the command-line. These drivers are based on command-line tools and don’t produce screen output, making them ideal for completely automated setups.

To be able to visually investigate test failures, the developer needs to implement mechanisms to capture screenshots, otherwise rely on command line output.

## What is Page Object Design Pattern, explain with example

In Page Object pattern, Its idea is to have each element or action stored only once in a system. This avoids copy/paste and reduces the overhead for code maintenance.

* Each page or re-usable part of a page (i.e. header, footer, menu) is a separate class.
* Class constructor takes WebDriver as an argument and uses it internally to locate elements.
* Each element is a private property (or getter in Java).
* Actions are public and internally operate with elements.

public class HomePage extends BaseSubPage {

public WebDriver driver;

@FindBy(how = How.XPATH, using = "//a[contains(text(),'Dropdown')]")

private WebElement Lnk\_HomePage\_DropDown;

public HomePage(WebDriver driver) throws AutoKitError {

super(driver);

AppLogs.info("HomePage() starts..");

this.driver = driver;

// Check that we're on the right page.

if (!driver.getTitle().contains("The Internet")) {

throw new AutoKitError("This is not the Home page");

}

AppLogs.info("HomePage() ends..");

}

public DropDownPage openDropDownPage() throws AutoKitError{

AppLogs.info("openDrowpDownPage() starts..");

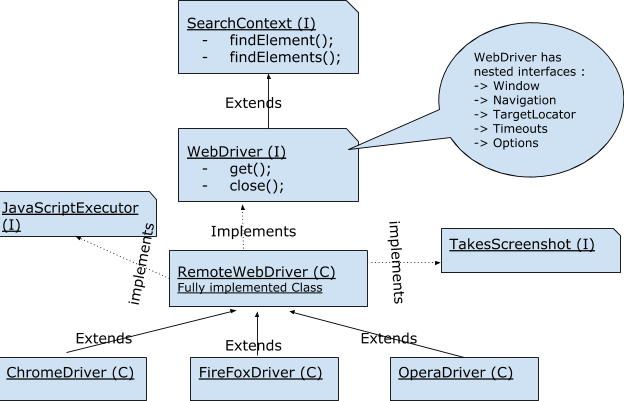
getActions().click(Lnk\_HomePage\_DropDown);

getActions().selectWindow("The Internet");

return PageFactory.initElements(driver, DropDownPage.class);

}

## How Selenium WebDriver works? OR What is architecture of Selenium WebDriver? OR Why do we require browser drivers? OR What is WebDriver hierarchy? OR Explain hierarchy of Classes and interfaces in Selenium WebDriver



WebDriver Architecture Diagram

Let’s understand above diagram in details:

1. SearchContext is a top most interface which has only two methods names findElement() and findElements(). These methods will be abstract as SearchContext is an interface.
2. WebDriver is also an interface which extends SearchContext. WebDriver has many abstract methods like get(String url), close(), quit() , getWindowHandle etc. WebDriver has nested interfaces names Window, Navigation, Timeouts etc. These nested interfaces are used to perform/group specific operation like getPosition(), back(), forward() etc.
3. RemoteWebDriver is a fully implemented class which implements Webdriver, JavascriptExecutor and TakesScreenshot. Fully implemented class means it defined body for all inherited methods.
4. Then we have browser specific driver classes like ChromeDriver(), EdgeDriver(), FirefoxDriver() etc

Interface **WebDriver** inherits from interface**[SearchContext](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/SearchContext.html)**

All Known Implementing Classes: [**ChromeDriver**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/chrome/ChromeDriver.html)**,** [**EdgeDriver**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/edge/EdgeDriver.html)**,** [**EventFiringWebDriver**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/support/events/EventFiringWebDriver.html)**,** [**FirefoxDriver**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/firefox/FirefoxDriver.html)**,** [**InternetExplorerDriver**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/ie/InternetExplorerDriver.html)**,** [**OperaDriver**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/opera/OperaDriver.html)**,** [**RemoteWebDriver**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/remote/RemoteWebDriver.html)**,****[SafariDriver](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/safari/SafariDriver.html)**

The main interface to use for testing, which represents an idealised web browser. The methods in this interface fall into three categories:

1. Control of the browser itself
2. Selection of WebElements
3. Debugging aids

**Nested class summary**

* static interface - WebDriver.ImeHandler : An interface for managing input methods.
* static interface - WebDriver.Navigation :
* static interface - WebDriver.Options : An interface for managing stuff you would do in a browser menu
* static interface - WebDriver.TargetLocator : Used to locate a given frame or window.
* static interface - WebDriver.Timeouts : An interface for managing timeout behavior for WebDriver instances.
* static interface - WebDriver.Window

**Method details –**

**Get - void get(java.lang.String url)** - Load a new web page in the current browser window. This is done using an HTTP GET operation, and the method will block until the load is complete. This will follow redirects issued either by the server or as a meta-redirect from within the returned HTML. Should a meta-redirect "rest" for any duration of time, it is best to wait until this timeout is over, since should the underlying page change whilst your test is executing the results of future calls against this interface will be against the freshly loaded page. Synonym for WebDriver.Navigation.to(String).

Parameters: url - The URL to load. It is best to use a fully qualified URL

**getCurrentUrl - java.lang.String getCurrentUrl() :** Get a string representing the current URL that the browser is looking at.

Returns: The URL of the page currently loaded in the browser

**getTitle - java.lang.String getTitle()** : The title of the current page.

Returns: The title of the current page, with leading and trailing whitespace stripped, or null if one is not already set

**findElements - java.util.List<WebElement> findElements(By by) :** Find all elements within the current page using the given mechanism. This method is affected by the 'implicit wait' times in force at the time of execution. When implicitly waiting, this method will return as soon as there are more than 0 items in the found collection, or will return an empty list if the timeout is reached.

Specified by: findElements in interface SearchContext

Parameters: by - The locating mechanism to use

Returns: A list of all WebElements, or an empty list if nothing matches

**findElement - WebElement findElement(By by) :** Find the first WebElement using the given method. This method is affected by the 'implicit wait' times in force at the time of execution. The findElement(..) invocation will return a matching row, or try again repeatedly until the configured timeout is reached. findElement should not be used to look for non-present elements, use findElements(By) and assert zero length response instead.

Specified by: findElement in interface SearchContext

Parameters: by - The locating mechanism

Returns: The first matching element on the current page

Throws: NoSuchElementException - If no matching elements are found

**getPageSource - java.lang.String getPageSource()** - Get the source of the last loaded page. If the page has been modified after loading (for example, by Javascript) there is no guarantee that the returned text is that of the modified page. Please consult the documentation of the particular driver being used to determine whether the returned text reflects the current state of the page or the text last sent by the web server. The page source returned is a representation of the underlying DOM: do not expect it to be formatted or escaped in the same way as the response sent from the web server. Think of it as an artist's impression.

Returns: The source of the current page

**Close - void close()** - Close the current window, quitting the browser if it's the last window currently open.

**Quit - void quit() :** Quits this driver, closing every associated window.

**getWindowHandles - java.util.Set<java.lang.String> getWindowHandles() :** Return a set of window handles which can be used to iterate over all open windows of this WebDriver instance by passing them to switchTo().WebDriver.Options.window()

Returns: A set of window handles which can be used to iterate over all open windows.

**getWindowHandle - java.lang.String getWindowHandle() :** Return an opaque handle to this window that uniquely identifies it within this driver instance. This can be used to switch to this window at a later date

Returns: the current window handle

**switchTo - WebDriver.TargetLocator switchTo() :** Send future commands to a different frame or window.

Returns: A TargetLocator which can be used to select a frame or window

**Navigate - WebDriver.Navigation navigate() :** An abstraction allowing the driver to access the browser's history and to navigate to a given URL.

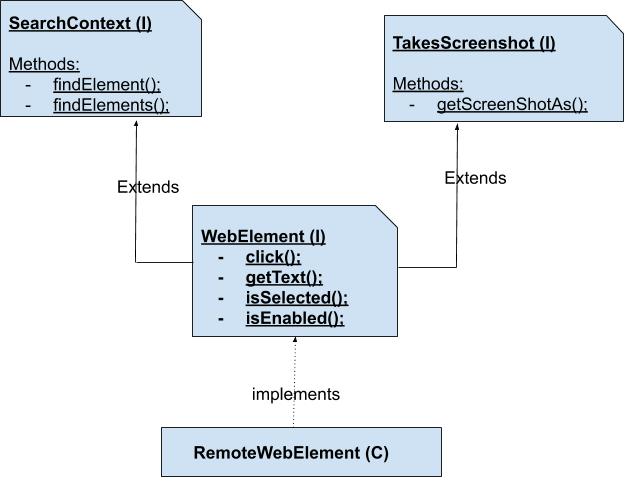
Returns: A WebDriver.Navigation that allows the selection of what to do next

**Manage - WebDriver.Options manage()** : Gets the Option interface

Returns: An option interface

### 

## What is WebElement interface hierarchy?



Represents an HTML element. Generally, all interesting operations to do with interacting with a page will be performed through this interface.

1. WebElement is an interface which extends SearchContext and TakesScreenshot interfaces. It has many useful abstarct methids like click(), sendKeys(), isSelected() etc.
2. RemoteWebELement is a fully implemented class as it implements WebElement interface. In fact, findElement() and findElements() of SearchContext interface have been defined properly in this class only.
3. When we search for some element using findElement or findElements() method, a type of RemoteWebElement class is up casted to WebElement. And when we perform any action on webelement like click(), method defined in RemoteWebElemnt class will be executed because of overriding concept.

Interface **WebElement** implements **SearchContext, TakesScreenshot. RemoteWebElement** classimplements **WebElement** interface

**Method in WebElement interface**

**Click** - **void click()** - Click this element. If this causes a new page to load, you should discard all references to this element and any further operations performed on this element will throw a StaleElementReferenceException. Note that if click() is done by sending a native event (which is the default on most browsers/platforms) then the method will \_not\_ wait for the next page to load and the caller should verify that themselves. There are some preconditions for an element to be clicked. The element must be visible and it must have a height and width greater then 0.

Throws: **StaleElementReferenceException** - If the element no longer exists as initially defined

**Submit - void submit() -**  If this current element is a form, or an element within a form, then this will be submitted to the remote server. If this causes the current page to change, then this method will block until the new page is loaded.

Throws: **NoSuchElementException** - If the given element is not within a form

**sendKeys - void sendKeys(java.lang.CharSequence... keysToSend) -**  Use this method to simulate typing into an element, which may set its value.

Parameters: keysToSend - character sequence to send to the element

Throws: java.lang.IllegalArgumentException - if keysToSend is null

**Clear - void clear() -**  If this element is a text entry element, this will clear the value. Has no effect on other elements. Text entry elements are INPUT and TEXTAREA elements. Note that the events fired by this event may not be as you'd expect. In particular, we don't fire any keyboard or mouse events. If you want to ensure keyboard events are fired, consider using something like sendKeys(CharSequence...) with the backspace key. To ensure you get a change event, consider following with a call to sendKeys(CharSequence...) with the tab key.

**getTagName - java.lang.String getTagName() -** Get the tag name of this element. Not the value of the name attribute: will return "input" for the element <input name="foo" />.

Returns: The tag name of this element.

**getAttribute - java.lang.String getAttribute(java.lang.String name) -** Get the value of the given attribute of the element. Will return the current value, even if this has been modified after the page has been loaded. More exactly, this method will return the value of the property with the given name, if it exists. If it does not, then the value of the attribute with the given name is returned. If neither exists, null is returned. The "style" attribute is converted as best can be to a text representation with a trailing semi-colon. The following are deemed to be "boolean" attributes, and will return either "true" or null:

**isSelected - boolean isSelected() -**  Determine whether or not this element is selected or not. This operation only applies to input elements such as checkboxes, options in a select and radio buttons. For more information on which elements this method supports, refer to the specification.

Returns: True if the element is currently selected or checked, false otherwise.

**isEnabled - boolean isEnabled() -**  Is the element currently enabled or not? This will generally return true for everything but disabled input elements.

Returns: True if the element is enabled, false otherwise.

**getText - java.lang.String getText() -** Get the visible (i.e. not hidden by CSS) text of this element, including sub-elements.

Returns: The visible text of this element.

See Also: "Get Element Text" section in W3C WebDriver Specification

**findElements - java.util.List<WebElement> findElements(By by) -** Find all elements within the current context using the given mechanism. When using xpath be aware that webdriver follows standard conventions: a search prefixed with "//" will search the entire document, not just the children of this current node. Use ".//" to limit your search to the children of this WebElement. This method is affected by the 'implicit wait' times in force at the time of execution. When implicitly waiting, this method will return as soon as there are more than 0 items in the found collection, or will return an empty list if the timeout is reached.

Specified by: findElements in interface SearchContext

Parameters: by - The locating mechanism to use

Returns: A list of all WebElements, or an empty list if nothing matches.

**findElement - WebElement findElement(By by) -** Find the first WebElement using the given method. See the note in findElements(By) about finding via XPath. This method is affected by the 'implicit wait' times in force at the time of execution. The findElement(..) invocation will return a matching row, or try again repeatedly until the configured timeout is reached. findElement should not be used to look for non-present elements, use findElements(By) and assert zero length response instead.

Specified by: findElement in interface SearchContext

Parameters: by - The locating mechanism

Returns: The first matching element on the current context.

Throws: NoSuchElementException - If no matching elements are found

**isDisplayed - boolean isDisplayed() -** Is this element displayed or not? This method avoids the problem of having to parse an element's "style" attribute.

Returns: Whether or not the element is displayed

**getLocation - Point getLocation() -** Where on the page is the top left-hand corner of the rendered element?

Returns: A point, containing the location of the top left-hand corner of the element

**getSize - Dimension getSize() -** What is the width and height of the rendered element?

Returns: The size of the element on the page.

**getRect - Rectangle getRect() -**  Returns: The location and size of the rendered element

**getCssValue - java.lang.String getCssValue(java.lang.String propertyName) -** Get the value of a given CSS property. Color values should be returned as rgba strings, so, for example if the "background-color" property is set as "green" in the HTML source, the returned value will be "rgba(0, 255, 0, 1)". Note that shorthand CSS properties (e.g. background, font, border, border-top, margin, margin-top, padding, padding-top, list-style, outline, pause, cue) are not returned, in accordance with the DOM CSS2 specification - you should directly access the longhand properties (e.g. background-color) to access the desired values.

Parameters: propertyName - the css property name of the element

Returns: The current, computed value of the property.

## Why did you select Selenium WebDriver for front end automation of your application? OR What factors you will take into consideration before deciding a tool to automate front end of your application?

* Do you have the necessary skilled resource to allocate for automation tasks?
* Is the current tool version stable? Is the vendor company established with good customer support as well as online help resources and user manual?
* How is the tool learning curve? Is the learning time acceptable for your goals?
* Which testing types does it support? A tool which supports maximum testing types (Unit, functional, regression etc.) is always a better choice.

Warning – Don’t go for a tool just because it is supporting all testing types. It’s also important that the tool should be powerful enough to automate your complex requirements.

* How easy is it to provide input test data for complex or load tests? A tool supporting test data input from various data files such as Excel, XML, text file etc. would be a big relief for the automation the testers.

## How to assert title of the web page?

Steps to assert title of a web page -

1. Store expected page title in a String variable

String expectedTitle = "My Title";

1. Create a browser session
2. Load application under test
3. Get the title of page using method "driver.getTitle()" and store it in String variable

String actualTitle = driver.getTitle();

1. Apply the Assertion like below

Assert.assertTrue(actualTitle.equalsIgnoreCase(expectedTitle),"Page

title name not matched");

or something like this in Java

if(driver.getTitle().contains(expectedTitle))

//Pass

System.out.println("Page title contains \"some expected text");

else

//Fail

System.out.println("Page title doesn't contains”+ expectedTitle);

## How to capture screen-shot in Selenium WebDriver ?

//code to take and save screenshot as image

File screenshotFile = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

//Location to save Image locally

File DestFile = new FIle("C:\\screenshot.png")

//Code to copy file to local drive

FileUtils.copyFile(screenshotFile, DestFile);

Working Example –

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

WebDriver driver = new FirefoxDriver();

driver.get("https://www.google.co.in");

File screenshotFile = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

File DestFile = new FIle("C:\\screenshot.png")

FileUtils.copyFile(screenshotFile, DestFile));

## What are the limitations of Selenium WebDriver?

* Selenium needs very much expertise resources. The resource should also be very well versed in framework architecture.
* Selenium only supports web based application and does not support windows based application.
* It is difficult to test Image based application.
* Selenium need outside support for report generation activity like dependence on TestNG or Jenkins or Cucumber Report.
* Selenium does not support built in add-ins support.
* Selenium user lacks online support for the problems they face.
* Selenium does not provide any built in IDE for script generation and it need other IDE like Eclipse or IntelliJ IDEA for writing scripts.
* Selenium does not support file upload facility.
* Selenium partially supports for Dialog boxes.

## What is the super interface of WebDriver?

SearchContext

## Is WebDriver a class or interface?

WebDriver is an interface.

## Is FirefoxDriver a class or interface?

FirefoxDriver is a class which implements WebDriver interface.

## What is the use of JavaScriptExecutor?

JavaScriptExecutor is an interface which provides a mechanism to execute Javascript through the Selenium WebDriver. It provides “**executescript**” and “**executeAsyncScript**” methods, to run JavaScript in the context of the currently selected frame or window. An example of that is:

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript(Script,Arguments);

**Generate Alert Pop Window**

JavascriptExecutor js = (JavascriptExecutor)driver;

Js.executeScript("alert('hello world');");

**Click Action**

JavascriptExecutor js = (JavascriptExecutor)driver;

js.executeScript("arguments[0].click();", element);

**Refresh Browser**

JavascriptExecutor js = (JavascriptExecutor)driver;

driver.executeScript("history.go(0)");

**Get InnerText of a Webpage**

JavascriptExecutor js = (JavascriptExecutor)driver;

string sText = js.executeScript("return document.documentElement.innerText;").toString();

**Get Title of a WebPage**

JavascriptExecutor js = (JavascriptExecutor)driver;

string sText = js.executeScript("return document.title;").toString();

**Scroll Page**

JavascriptExecutor js = (JavascriptExecutor)driver;

//Vertical scroll - down by 150 pixels

js.executeScript("window.scrollBy(0,150)");

## How PageFactory helps you in achieving Page Object Model design?

Page Object is a class that represents a web page and hold the functionality and members. Page Factory is a way to initialize the web elements you want to interact with within the page object when you create an instance of it.

## What are the types of waits available in Selenium WebDriver? Or Explain usage of each type of waits with scenarios Or How many wait commands are in Selenium WebDriver

In Selenium we could see three types of waits such as Implicit Waits, Explicit Waits and Fluent Waits.

**Implicit Waits**

Once you set implicit wait, it will be applicable for all findElemnt() and findElements() method. There is no need to write implicit wait code again and again.

1. If we set implicit wait as 30 seconds, It will be applicable to all web element to be located by WebDriver.
2. When searching for a single element, the driver should poll the page until the element has been found, or this timeout expires before throwing NoSuchElementException. When searching for multiple elements, the driver should poll the page until at least one element has been found or this timeout has expired.
3. If you write implicit wait statement multiple times, time specified in latest implicit wait statement will be applicable. For example: You specify implicit wait as 30 seconds at line no 5 and another implicit wait as 60 seconds at line no 10, then implicit wait as 30 seconds will be applicable from line no 6-9 and implicit wait for 60 seconds will be applicable from line no 11 onward.
4. Selenium WebDriver does not provide any direct getter method to retrieve implicit wait time. But you can create your own methods to retrieve it.
5. It is good practice to define implicit wait in the beginning of the program when we create WebDriver reference variable.
6. The default setting is Zero. Once set, the implicit wait is set for the life of the WebDriver object instance.
7. Increasing the implicit wait timeout should be used judiciously as it will have an adverse effect on test run time, especially when used with slower location strategies like XPath.
8. If you use the implicit wait in selenium it applies to the web driver globally and increases the execution time for the entire script. so it is not always advisable.
9. You can not wait till some specific condition is satisfied like invisibility of element, when alert is present etc.

//syntax

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

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

// Loading a URL

driver.get(**"https://www.redbus.in/"**);

// Locating and typing in From text box.

WebElement fromTextBox= driver.findElement(By.id(**"src"**));

fromTextBox.sendKeys(**"Ban"**);

// setting implicit time

driver.manage().timeouts().implicitlyWait(30, TimeUnit.**SECONDS**);

System.**out**.println(**"Wait starts:"**+**new** Date());

System.**out**.println(**"Typed Ban"**);

// Clicking on first search result

driver.findElement(By.xpath(**"//li[@select-id='results[0]']"**)).click();

System.**out**.println(**"Selected Bangalore"**);

System.**out**.println(**"Wait ends:"**+**new** Date());

//Closing browser

driver.quit();

System.**out**.println(**"browser closed"**);

**Explicit Waits -** Explicit waits are confined to a particular web element. Explicit Wait is code you define to wait for a certain condition to occur before proceeding further in the code.

Explicit wait is of two types:

1. WebDriverWait
2. FluentWait

The following are the Expected Conditions that can be used in 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 Waits -** FluentWait can define the maximum amount of time to wait for a specific condition and frequency with which to check the condition before throwing an “*ElementNotVisibleException*” exception. To say in effortless manner, it tries to find the web element repeatedly at regular intervals of time until the timeout or till the object gets found.

Wait wait = new FluentWait<WebDriver>(driver)

.withTimeout(45, TimeUnit.SECONDS)

.pollingevery(5, TimeUnit.SECONDS)

.ignoring(NoSuchElementException.class);

## How to input text in the text box using Selenium WebDriver? Or How to type into a text box using Selenium?

By using sendKeys() method

WebDriver driver = new FirefoxDriver();

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

driver.findElement(By.xpath("xpath")).sendKeys("Software Testing Material Website");

## How to input text in the text box without calling the sendKeys()?

// To initialize js object

JavascriptExecutor JS = (JavascriptExecutor)webdriver;

// To enter username

JS.executeScript("document.getElementById('User').value='Selenium Interview'");

// To enter password

JS.executeScript("document.getElementById('Pass').value=selenium");

## How to clear the text in the text box using Selenium WebDriver?

By using clear() method

WebDriver driver = new FirefoxDriver();

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

driver.findElement(By.xpath("xpath\_of\_element1")).sendKeys("Selenium Interview");

driver.findElement(By.xpath("xpath\_of\_element1")).clear();

## How to get a text of a web element? Or How can we get a text from a web element using Selenium? Or Which method can be used to get the text of an element?

By using getText() method, we can get text from an web element. Example code -

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeDriver;

**public class** getTextExample {

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

System.*setProperty*(**"webdriver.chrome.driver"**, **"path of chromeDriver\\chromedriver.exe"**);

WebDriver driver = **new** ChromeDriver();

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

String availableText = driver.findElement(By.*xpath*(**"//\*[@id='gbw']/div/div/div[1]/div[1]/a"**)).getText();

System.***out***.println(**"Text Available is :"** + availableText);

}

}

## How to get an attribute value using Selenium WebDriver?

By using getAttribute(value); It returns the value of the attribute passed as a

parameter.

HTML: <input name="Selenium" value="Selenium">Selenium Interview</input>

String attributeValue = driver.findElement(By.name("Selenium")).getAttribute("value");

System.out.println("Available attribute value is :"+attributeValue);

Output: Selenium

## How to click on a hyperlink using Selenium WebDriver? Or How to click on a hyperlink using Selenium WebDriver?

We use click() method in Selenium to click on the hyperlink

//Link Text is target hyperlink text

driver.findElement(By.linkText(“Link Text”)).click();

## How to submit a form using Selenium WebDriver?

We use “submit” method on element to submit a form

driver.findElement(By.id("form1")).submit();

Alternatively, you can use click method on the element which does form submission

## How to press ENTER key on text box In Selenium WebDriver?

To press ENTER key using Selenium WebDriver, We need to use Selenium Enum Keys with its constant ENTER.

driver.findElement(By.xpath("xpath")).sendKeys(Keys.ENTER);

## How do you select the drivers to launch a URL?

Selenium provides browser implementation class for every browser available in market. In order to launch your target browser, you should create object of corresponding browser class. E.g.

// to launch Firefox browser, similar way you can launch Chrome, IE etc

WebDriver driver = new FirefoxDriver();

// Loading a URL

driver.get("https://www.google.co.in/");

## How to check which check-box from multiple check-box options is selected previously using Selenium?

Selenium provides **isSelected()** method, when executed on checkbox web element, it returns true if checkbox is selected else returns false.

WebElement els = driver.findElements(By.id("input1"));

System.out.println(el.isSelected());

## What is the return type of isSelected() method in Selenium?

isSelected() method returns Boolean value.

## How to fetch the current page URL in Selenium?

To fetch the current page URL, we use *getCurrentURL()*

driver.getCurrentUrl();

## How can we maximize browser window in Selenium?

To maximize browser window in selenium we use *maximize()* method. This method maximizes the current window if it is not already maximized

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

## How to delete cookies in Selenium?

Selenium provides 3 ways to delete cookies

1. Using cookie Name, e.g.

driver.manage().deleteCookieNamed("CookieName");

1. Delete a particular Cookie, e.g.

driver.manage().deleteCookie(loadedCookie);

1. Delete all cookies, e.g.

driver.manage().deleteAllCookies();

## What are the ways to refresh a browser using Selenium WebDriver?

There are multiple ways to refresh a page in selenium

* Using driver.navigate().refresh() command
* Using driver.get(“URL”) on the current URL or using driver.getCurrentUrl()
* Using driver.navigate().to(“URL”) on the current URL or driver.navigate().to(driver.getCurrentUrl());
* Using sendKeys(Keys.F5) on any textbox on the webpage

## What is the difference between driver.getWindowHandle() and driver.getWindowHandles() in Selenium WebDriver?

**driver.getWindowHandle()** – It returns a handle of the current page (a unique identifier)

**driver.getWindowHandles()** – It returns a set of handles of the all the pages available.

## What is the difference between driver.close() and driver.quit() methods?

Purpose of these two methods (driver.close and driver.quit) is almost same. Both allow us to close a browser but still, there is a difference.

**driver.close()***:* To close current WebDriver instance

**driver.quit()***:* To close all the opened WebDriver instances

## What is the difference between driver.findElement() and driver.findElements() commands? Or How do you use findElement() and findElements()? Or its return type and few examples of where you have used in Selenium Projects?

|  |  |
| --- | --- |
| **findElement()** | **findElements()** |
| 1. Returns a single WebElement (found first) based on the locator passed as parameter. 2. Syntax of findElement() :   WebElement textbox = driver.findElement(By.id(“Locator”)); 3. if no element is found then findElement() throws NoSuchElementException 4. This command is used to identify single object. | 1. Returns a list of WebElements, all satisfying the locator value passed. 2. Syntax of findElements() :  List<WebElement> elements = element.findElements(By.id(“id1”)); 3. findElements() returns a list of 0 elements 4. This command is used when you want to fetch multiple objects from webpage. |

## How to select a value in a dropdown? How to handle a drop-down field and select a value from it using Selenium?

By using *Select* class

WebElement mySelectElement = driver.findElement(By.name("dropdown"));

Select dropdown = new Select(mySelectElement);

dropdown.selectByVisibleText(Text);

dropdown.selectByIndex(Index);

dropdown.selectByValue(Value);

## How to handle Ajax calls in Selenium WebDriver?

Handling AJAX calls is one of the common issues when using Selenium WebDriver. We wouldn’t know when the AJAX call would get completed and the page has been updated. In this post, we see how to handle AJAX calls using Selenium.

AJAX stands for Asynchronous JavaScript and XML. AJAX allows the web page to retrieve small amounts of data from the server without reloading the entire page. AJAX sends HTTP requests from the client to server and then process the server’s response without reloading the entire page. To handle AJAX controls, wait commands may not work. It’s just because the actual page is not going to refresh.

When you click on a submit button, the required information may appear on the web page without refreshing the browser. Sometimes it may load in a second and sometimes it may take longer. We have no control over loading time. The best approach to handle this kind of situations in selenium is to use dynamic waits (i.e. WebDriverWait in combination with ExpectedCondition)

Some of the methods which are available are as follows:

**titleIs()** – The expected condition waits for a page with a specific title.

wait.until(ExpectedConditions.titleIs(“Deal of the Day”));

**elementToBeClickable()** – The expected condition waits for an element to be clickable i.e. it should be present/displayed/visible on the screen as well as enabled.

wait.until(ExpectedConditions.elementToBeClickable(By.xpath("xpath")));

**alertIsPresent()** – The expected condition waits for an alert box to appear.

wait.until(ExpectedConditions.alertIsPresent()) !=null);

**textToBePresentInElement()** – The expected condition waits for an element having a certain string pattern.

wait.until(ExpectedConditions.textToBePresentInElement(By.id(“title’”), “text to be found”));

## What are the advantages of Page Object Model Framework?

**Code reusability** – We could achieve code reusability by writing the code once and use it in different tests.

**Code maintainability** – There is a clean separation between test code and page specific code such as locators and layout which becomes very easy to maintain code. Code changes only on Page Object Classes when a UI change occurs. It enhances test maintenance and reduces code duplication.

**Object Repository** – Each page will be defined as a java class. All the fields in the page will be defined in an interface as members. The class will then implement the interface.

**Readability** – Improves readability due to clean separation between test code and page specific code

## How can you use the Recovery Scenario in Selenium WebDriver?

WebDriver does not provide recovery scenario capability, we need to depend on implementation language handle error scenario. In case of Java, we use “try - catch” block to handle unexpected error & resume uninterrupted execution of test scripts. E.g.

try {

driver.get("www.SoftwareTestingMaterial.com");

}catch(Exception e){

System.out.println(e.getMessage());

}

## What is desired capabilities?

The desired capability is a series of key/value pairs that stores the browser properties like browsername, browser version, the path of the browser driver in the system, etc. to determine the behaviour of the browser at run time.

* Desired capability can also be used to configure the driver instance of Selenium WebDriver.
* We can configure driver instance like FirefoxDriver, ChromeDriver, InternetExplorerDriver by using desired capabilities.

Desired Capabilities are more useful in cases like:

* In mobile application automation, where the browser properties and the device properties can be set.
* In Selenium grid when we want to run the test cases on a different browser with different operating systems and versions

String path = "src/test/resources/geckodriver";

System.out.println("\*\*\*\*\*\*\*\* " + path);

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

DesiredCapabilities capabilities = DesiredCapabilities.firefox();

capabilities.setCapability("marionette", true);

capabilities.setCapability("networkConnectionEnabled", true);

capabilities.setCapability("browserConnectionEnabled", true);

## How to delete Browser Cookies with Selenium WebDriver?

*driver.Manage().Cookies.DeleteAllCookies();*

## Explain the different exceptions in Selenium WebDriver

Exceptions in Selenium are similar to exceptions in other programming languages. The most common exceptions in Selenium are:

The complete list of exceptions in Selenium

1. **ConnectionClosedException:** This exception takes place when there is a disconnection in the driver.
2. **ElementClickInterceptedException:** The command could not be completed as the element receiving the events is concealing the element which was requested clicked.
3. **ElementNotInteractableException:** This Selenium exception is thrown when an element is presented in the DOM but it is impossible to interact with such element.
4. **ElementNotSelectableException:** This Selenium exception is thrown when an element is presented in the DOM but you can be able to select. Hence, it is impossible to interact with.
5. **ElementNotVisibleException:** This type of Selenium exception takes place when existing element in DOM has a feature set as hidden. In this situation, elements are there, but you can not see and interact with the WebDriver.
6. **ErrorHandler.UnknownServerException:** Exception is used as a placeholder if the server returns an error without a stack trace.
7. **ErrorInResponseException:** This exception is thrown when a fault has occurred on the server side. You can see it happens when interacting with the Firefox extension or the remote driver server.
8. **ImeActivationFailedException:** This exception occurs when IME engine activation has failed.
9. **ImeNotAvailableException:** This exception takes place when IME support is unavailable.
10. **InsecureCertificateException:** Navigation made the user agent to hit a certificate warning, which is caused by an invalid or expired TLS certificate.
11. **InvalidArgumentException:** This Selenium exception is thrown if an argument does not belong to the expected type.
12. **InvalidCookieDomainException:** This happens when you try to add a cookie under a different domain rather than the current URL.
13. **InvalidCoordinatesException:** This happens if the coordinates offered to an interacting operation are not valid.
14. **InvalidElementStateException:** This Selenium exception occurs if a command cannot be finished as the element is invalid.
15. **InvalidSessionIdException:** Takes place when the given session ID is not included in the list of active sessions, which means the session does not exist or is inactive either.
16. **InvalidSwitchToTargetException**: Happens if frame or window target to be switched does not exist.
17. **JavascriptException:** This problem happens when executing JavaScript supplied by the user.
18. **JsonException:** Happens when you afford to get the session capabilities where the session is not created.
19. **MoveTargetOutOfBoundsException:** Takes place if the target provided to the ActionChains move() methodology is not valid. For example: out of document.
20. **NoAlertPresentException:** Happens when you switch to no presented alert.
21. **NoSuchAttributeException:** Occurs when the attribute of element could not be found.
22. **NoSuchContextException:** Happens in mobile device testing and is thrown by ContextAware.
23. **NoSuchCookieException:** This exception is thrown if there is no cookie matching with the given path name found amongst the associated cookies of the current browsing context’s active document.
24. **NoSuchElementException:** Happens if an element could not be found.
25. **NoSuchFrameException:** Takes place if frame target to be switch does not exist.
26. **NoSuchWindowException:** Occurs if window target to be switch does not exist.
27. **NotFoundException:** This exception is subclass of WebDriverException. It happens when an element on the DOM does not exist.
28. **RemoteDriverServerException:** This Selenium exception is thrown when server do not respond due to the problem that the capabilities described are not proper.
29. **ScreenshotException:** It is impossible to capture a screen.
30. **ScriptTimeoutException:** Thrown when executeAsyncScript takes more time than the given time limit to return the value.
31. **SessionNotCreatedException:** A new session could not be successfully created.
32. **SessionNotFoundException:** The WebDriver is performing the action right after you quit the browser.
33. **StaleElementReferenceException:** This Selenium exception happens if the web element is detached from the current DOM.
34. **TimeoutException:** Thrown when there is not enough time for a command to be completed.
35. **UnableToCreateProfileException:** You can open a browser with certain options using profiles, but sometimes a new version of Selenium driverserver or browser may not support the profiles.
36. **UnableToSetCookieException:** Occurs if a driver is unable to set a cookie.
37. **UnexpectedAlertPresentException:** This Selenium exception happens when there is the appearance of an unexpected alert.
38. **UnexpectedTagNameException:** Happens if a support class did not get a web element as expected.
39. **UnhandledAlertException:** It happens when there is an alert, but WebDriver is unable to perform Alert operation.
40. **UnknownMethodException:** Thrown when the requested command matching with a known URL but not matching with a methodology for that URL.
41. **UnreachableBrowserException:** This Selenium exception happens if the browser is unable to be opened or have crashed because of some reasons.
42. **UnsupportedCommandException:** Occurs when remote WebDriver does not send valid command as expected.
43. **WebDriverException:** This takes place when the WebDriver is performing the action right after you close the browser

## Write a code to wait for an alert to appear

We can write a code such that we specify the XPath of the web element that needs to be visible on the page and then ask the WebDriver to wait for a specified time. Look at the sample piece of code below:

WebDriverWait wait=new WebDriverWait(driver, 20);

Element = wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath( “<xpath”))

## What is use of Select class of Selenium WebDriver? What are its limitations? Or How to handle a dropdown in Selenium WebDriver? How to select a value from dropdown?

Questions on dropdown and selecting a value from that dropdown are very common Selenium interview questions because of the technicality involved in writing the code.

WebDriver API provides class Select. Select Class has many methods and function that interacts with Drop Down.

HTML code of a select drop down looks like following -

<select name="Fruits">

<option value="Apple">Apple</option>

<option value=" Orange ">Orange</option>

<option value=" Mango ">Mango</option>

<option value=" Banana ">Banana</option>

</select>

We can do select/deselect operation using following methods -

**Selecting Dropdown Option using Visible Text**: We can select the dropdown option using visible text. For example, we want to select “Mango” in above drop down using selectByVisibleText() method. So the code will look like this

Example code : select.selectByVisibleText("Mango");

**Selecting Dropdown Option using Index**: We can select the dropdown option using an index. But we need to remember that index of any dropdown starts from 0 so if there are 4 options in any dropdown then there would be following index 0, 1, 2, 3

So the index of the first option would be 0 and similarly index of the second option would be 1 and so on and so forth. So if the user wants to select the 2nd option from the dropdown, then he needs to provide the index 1. So the code will look like this

Example code : select.selectByIndex(1);

**Selecting Dropdown Option using Value**: In this, we can select any option, using value attributes value. So if we want to select Banana in provided HTML. In that case, we need to call value of value attribute and that is “Banana”. So the code will look like this.

Example code : select.selectByValue("Banana ");

if we want to validate option selected is correct or not for the same we have one method **getFirstSelectedOption()** and it will return the detail of selected option and to fetch the text we can call method getText(). So the code would look like

**select.getFirstSelectedOption().getText()**

Similarly, the “Select” class has an equal number of methods to deselect the option from the dropdown.

**DeselectAll**: this option is used to deselect all options selected in dropdown. Here is code

select.deselectAll();

**DeselectByVisibleText**: This option is used to deselect selected option using visible text and this can be used like this.

**DeselectByIndex**: This method is used to deselect the selected option using the Index.

**DeselectByValue**: This method is used to deselect the selected dropdown option using the value of the value attribute.

## How to send ALT/SHIFT/CONTROL key in Selenium WebDriver?

When we generally use ALT/SHIFT/CONTROL keys, we hold onto those keys and click other buttons to achieve the special functionality. So it is not enough just to specify **keys.ALT** or **keys.SHIFT** or **keys.CONTROL** functions.

For the purpose of holding onto these keys while subsequent keys are pressed, we need to define two more methods: **keyDown(modifier\_key)** and **keyUp(modifier\_key)**

Parameters: **Modifier\_key (keys.ALT or Keys.SHIFT or Keys.CONTROL)**

Purpose: Performs a modifier key press and does not release the modifier key. Subsequent interactions may assume it’s kept pressed.

Parameters: **Modifier\_key (keys.ALT or Keys.SHIFT or Keys.CONTROL)**

Purpose: Performs a key release.

Hence with a combination of these two methods, we can capture the special function of a particular key.

public static void main(String[] args)

{

String baseUrl = “https://www.facebook.com”;

WebDriver driver = new FirefoxDriver();

driver.get("baseUrl");

WebElement txtUserName = driver.findElement(By.id(“Email”);

Actions builder = new Actions(driver);

Action seriesOfActions = builder

.moveToElement(txtUerName)

.click()

.keyDown(txtUserName, Keys.SHIFT)

.sendKeys(txtUserName, “hello”)

.keyUp(txtUserName, Keys.SHIFT)

.doubleClick(txtUserName);

.contextClick();

.build();

seriesOfActions.perform();

}

## How to set the size of browser window using Selenium?

To resize the current window to a particular dimension, you can use the **setSize()** method. Check out the below piece of code:

System.out.println(driver.manage().window().getSize());

Dimension d = new Dimension(420,600);

driver.manage().window().setSize(d);

To set the window to a particular size, use **window.resizeTo()** method. Check the below piece of code:

((JavascriptExecutor)driver).executeScript("window.resizeTo(1024, 768);");

To witness a demonstration on setting custom sizes for the browser window and finding various elements on the web page, see the video below.

## How to switch to a new window (new tab) which opens up after you click on a link?

If you click on a link in a web page, then for changing the WebDriver’s focus/ reference to the new window we need to use the **switchTo()** command. Look at the below example to switch to a new window:

**driver.switchTo().window();**

Here, ‘windowName’ is the name of the window you want to switch your reference to. In case you do not know the name of the window, then you can use the

**driver.getWindowHandles()** command to get the name of all the windows that were initiated by the WebDriver. Note that it will not return the window names of browser windows which are not initiated by your WebDriver.

Once you have the name of the window, then you can use an enhanced for loop to switch to that window. E.g.

String handle= driver.getWindowHandle();

for (String handle : driver.getWindowHandles())

{

driver.switchTo().window(handle);

}

## How do you upload a file using Selenium WebDriver?

To upload a file we can simply use the command **element.send\_keys(file path).** But there is a prerequisite before we upload the file. We have to use the html tag: **‘input’** and attribute type should be **‘file’**. Take a look at the below example where we are identifying the web element first and then uploading the file.

<input type="file" name="uploaded\_file" size="50" class="pole\_plik">

element = driver.find\_element\_by\_id(”uploaded\_file")

element.send\_keys("C:\test.txt")

## How do you find all clickable links in a page?

public class FindAllLinks {

public static void main(String[] args) {

WebDriver driver = new FirefoxDriver();

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

List allLinks = driver.findElements(By.tagName("a"));

System.out.println("All links on web page: " + allLinks.size() + " links");

for (WebElement link : allLinks) {

//print the links i.e. <https://www.example.com>

System.out.println(link.getAttribute("href"));

//print the links text

System.out.println(link.getText());

}

## Which library you used to read data from excel in Selenium WebDriver scripts?

Apache POI or JXL

## Challenges to run Selenium Scripts with IE Browser

* **Set path to IE executable driver –**

String service = "local path\\drivers\\IEDriverServer.exe";

System.setProperty("webdriver.ie.driver", service);

InternetExplorerDriver driver = new InternetExplorerDriver();

* **Protected Mode Settings -** Exception in thread “main” org.openqa.selenium.remote.SessionNotFoundException: **Unexpected error launching Internet Explorer. Protected Mode settings are not the same for all zones.** Enable Protected Mode must be set to the same value (enabled or disabled) for all zones. (WARNING: The server did not provide any stacktrace information)

Go through the above error, it clearly says that there is some issue with the **Protected Mode Settings** and the Protected Mode Settings are not same for all the zones. To avoid this error we need to set the Protected Mode Settings same for all the Zones.

* **Browser Zoom Level -** If your application is not starting or you are just getting one or multiple InternetExplorerDriver Listening on port window messages or if the below exception is appeared in the Eclipse Console, than your IE browser’s zoom level might be set to something other than 100%. Exception in thread “main” org.openqa.selenium.remote.SessionNotFoundException: Unexpected error launching Internet Explorer. Browser zoom level was set to 125%. It should be set to 100% (WARNING: The server did not provide any stacktrace information)

* **SendKeys Slowness on IE Browser -** This is again a very common issue with IE Browser that the Sendkey works very slow with Selenium script. But again not a rocket science to resolve this issue. Simply replace the current IEDriverServer.exe(if your machine is 64 bit machine) with IEDriverServer.exe of 32 bit

Download *IEDriverServer.exe of 32 bit system* even if the system is *64 bit* machine. Extract the Zip file and place the ***IEDriverServer*** in the same folder where the earlier version of *IEDriverServer* is available. It will replace the old file with new. Run the script again and this time the ***sendsKey()*** would work and the way it works in Chrome and Firefox.

* **Untrusted SSL Certificate** - Internet Explorer is the product of Microsoft and IE is much worried about security and IE is known as the most secured browser. At times using IE Browser with Selenium gives SLL Certificate pop up.

**Solution 1*-*** Add the below script just under the code to open the browser:

driver.navigate().to(“javascript:document.getElementById(‘overrideli

nk’).click()”);

**Solution 2 *-*** Another way of avoiding this error is to use DesiredCapability settings of the browser.

String service = "path of \\IEDriverServer.exe";

System.setProperty("webdriver.ie.driver", service);

// Create the DesiredCapability object of InternetExplorer

DesiredCapabilities capabilities = DesiredCapabilities.internetExplorer();

// Settings to Accept the SSL Certificate in the Capability object

capabilities.setCapability(CapabilityType.ACCEPT\_SSL\_CERTS, true);

### 

## What is Action interface & Actions class

Selenium provides API to Automate the Keyboard Events, Mouse events and drag and drop feature. We can achieve this by using both Action (an Interface) & Actions class. This API is the part of **org.openqa.selenium.interactions** package in Selenium library.

**Actions** class is based on builder design pattern which builds composite actions with the aggregation of Selenium WebDriver, where WebDriver is only used to identify the presence of web elements on web application.

**Action interface** is only used to represent the single user interaction i.e. to perform the series of action items build by Actions class.

Please find the following example using Action & Actions API:

* **Mouse Hovering on Web Element:** WebElement element=driver.findElement(By. Xpath("Locator Value"));  
   Actions actionseries=new Actions(driver);  
   Action action=actionseries.moveToElement(element);  
   action.perform()
* **Drag & Drop Web Element:** Actions actionseries=new Actions(driver);  
   WebElement sourceLocation=Wedriver.findElement(By. Xpath("Source location locator value"));  
   WebElement targetLocation=Wedriver.findElement(By. Xpath("Target location locator value"));  
   Action action=actionseries.dragAndDrop(sourceLocation, targetLocation).build();  
   action.perform();
* **Scrolling the Webpage:** Actions actionseries=new Actions(driver);  
   Action action=actionseries.sendKeys(Keys.PAGE\_DOWN).build();  
   action.perform();

We are using Action and Actions API in our Automation frameworks to automate functional Test Cases that includes above scenarios.

## What are the Challenges you faced in Selenium automation? How did you solve them?

* We faced some difficulties in handling Dynamic elements in our web pages...
* And we faced some problems in Extracting data from some web tables, and Extracting data from dynamic web tables.
* In or AUT (Application Under Test) so many uploading and downloading files scenarios and There Is no any direct command to upload or download files from web page using selenium WebDriver.
* We faced some challenges in handling multiple popup, in our AUT so many Popup windows, we can handle Popup windows using Selenium but in our application, some popup windows were not handled using regular methods
* Bitmap comparison was not supported by Selenium, we manually did this....
* And finally, it is not the challenge, we have spent more time than expected on Selenium Automation due to Framework implementation, no built in object repository, in selenium, and no built in result report facility…

## What are best practices for creating a UI test automation framework?

1. Do not rely only on UI test automation
2. Consider using a BDD framework
3. Always use test design patterns and principles
4. never use Thread.sleep() unless there are specific test requirements
5. Do not run All tests across All target browsers
6. Separate your tests from your test automation framework
7. Make your test automation framework portable
8. Name your tests wisely
9. Use soft assertions if you need to make a list of related checks on the same web page
10. Take screenshots for failed test cases
11. Make tests simpler instead of adding comments
12. Follow the “green tests run” policy
13. Use data-driven instead of repeated tests
14. All tests should be independent

# **Chapter – Junit / TestNG Framework**

TBD

# **Chapter – Cucumber Questions**

## What is difference between scenario and scenario outline

Feature: Calculator

As a user

I want to use a calculator to add numbers

So that I don't need to add myself

Scenario: Add two numbers 10 & 15

Given I have a calculator

When I add 10 and 15

Then the result should be 25

Scenario Outline: Add two numbers <num1> & <num2>

Given I have a calculator

When I add <num1> and <num2>

Then the result should be <total>

Examples:

| num1 | num2 | total |

| 10 | 15 | 25 |

When comparing a regular ***Scenario*** *Definition* with ***Scenario Outline***, values no longer need to be hard-coded in step definitions. Values are replaced with parameters as *<parameter\_name>* in step-definition itself.

# **Chapter – Question on API Testing**

## What is API Testing

API (application programming interface) testing is a type of software testing that performs verification directly at the API level. It is a part of integration testing that determines whether the APIs meet the expectations of functionality, reliability, performance, and security. This testing is performed at the message layer without GUI.

## What are different HTTP methods for REST API?

|  |  |
| --- | --- |
| **Method** | **Meaning** |
| GET | Fetches the representation of a resource state |
| POST | Create a new resource |
| PUT | Update a resource |
| DELETE | Removes a resource |
| HEAD | Fetches metadata associated with a resource’ state |
| OPTIONS | Lists the available methods |

## What are different response status code & their meaning?

|  |  |  |  |
| --- | --- | --- | --- |
| **Response Code Series** | **Response Code** | **Response Message** | **Meaning** |
| **1xx : Informational** | **Protocol information message** or **The request is received and continues to be processed** | | |
| 2xx : Success | **Server indicates that the request sent by client was successfully received, understood, and accepted.** | | |
|  | 200 | Ok | Success while processing client’s request |
|  | 201 | Created | New resource created |
| **3xxx: Redirection** | **Redirection related, Further action needs to be taken to complete the request. For example, 302 is for a temporary redirect.** | | |
|  | 301 | Moved permanently | Permanent redirection |
|  | 304 | Not Modified | Caching related response typically returned when the client has the same copy of the resource as the server |
|  | 307 | Temporary Redirect | Temporary redirection of resource |
| **4xx: Client Side Errors** | **Client sent a response which the server couldn’t comprehend, The request contains the wrong syntax or cannot be fulfilled.** | | |
|  | 400 | Bad Request | Malformed request by the client |
|  | 401 | Unauthorized | Client is not allowed to make request for access to particular resource |
|  | 402 | Forbidden | Client is forbidden to access the resource |
|  | 404 | Not Found | Resource does not exist or incorrect based on the request |
|  | 405 | Methods Not Allowed | Invalid method or unknown method used |
| **5xx : Server Side Errors** | **Server failed to fulfill the request sent by the client or The server fails to fulfill an apparently valid request** | | |
|  | 500 | Internal Server Error | Server failed to process request due to an internal error |