**Exception handling?**

An exception is an event, which occurs during the execution of a program. Where undesired condition is reached and script is just stop that arise during the execution of program.

For example, if a program has bunch of statements and an exception occurs mid way after executing certain statements then the statements after the exception will not execute and the program will terminate abruptly.

**Different between error and exception?**

Some of the examples of errors are system crash error and out of memory error. Errors mostly occur at runtime that's they belong to an unchecked type. Compile time error belong to checked type. Ex – file is missing then unhandled exception.

Exceptions are the problems which can occur at runtime and compile time. ... Exceptions are divided into two categories such as **checked exceptions and unchecked exceptions**.

.Errors mostly occur at runtime that's they belong to an unchecked type. Compile time error belong to checked type. Ex – file is missing then unhandled exception.

classNotFound exception, IOException-checked

NullPointerException, Airthmatic -unchecked

**What are the three types of errors in Java?**

There are three kinds of errors: syntax errors, runtime errors, and logic errors.

**What are the 3 blocks used to handle exception?**

The try block contains set of statements where an exception can occur. A try block is always followed by a catch block, which handles the exception that occurs in associated try block. A try block must be followed by catch blocks or finally block(always execute) or both.

**Difference between final and finally?**

A finally block of code always executes, whether or not an exception has occurred.

Final- cant change variable

If it is applicable for function, cant override the function.

If final applied in class, can not extend the class.

Different between Throws and Throw?

Throws- can be used to declare multiple exceptions. It is used in the method. It is used when the function has some statements that can lead to some exceptions. It is followed by class.

Throw- It is used inside the function. It is followed by instance.

``

**Collection framework**

A Collection represents a single unit of objects, i.e., a group

The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

3 components-

(Interface)List- ArrayList(class)

Set-Hashset

Map-HashMap

List: It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values.

List <data-type> list1= **new** ArrayList();

List1.add(“Mallika”);

List2.add(“Rahil”);

List3.add(“Sahil”);

1. //Traversing list through Iterator
2. Iterator itr=list.iterator();
3. **while**(itr.hasNext()){
4. System.out.println(itr.next());

Add// adding the element

Get//retrieve the element, list1.get(1)

Set//changing the element, list1.set(1,”hi”)

Collections.sort//sorting the element

Map: Java **HashMap** class implements the Map interface which allows us to store key and value pair, where keys should be unique. If you try to insert the duplicate key, it will replace the element of the corresponding key.

* Java HashMap contains values based on the key.
* Java HashMap contains only unique keys.
* Java HashMap may have one null key and multiple null values.
* Java HashMap is non synchronized.
* Java HashMap maintains no order.

HashMap map= new HashMap();

Map.put(“name”, “Mallika”);

Map.put(“Place”, “Jalandhar”);

Map.put(“Tool”,”Selenium”);

1. **for**(Map.Entry m : map.entrySet()){
2. System.out.println(m.getKey()+" "+m.getValue());
3. }

Map.putIfAbsent(103, "Gaurav")

1. HashMap<Integer,String> map=**new** HashMap<Integer,String>();
2. map.put(104,"Ravi");
3. map.putAll(Map);
4. //key-based removal
5. map.remove(100);
6. System.out.println("Updated list of elements: "+map);

//key-value pair based removal

    map.remove(102, "Rahul");

  System.out.println("Updated list of elements: "+map);

**HashSet**: Java HashSet class is used to create a collection that uses a hash table for storage.

* HashSet contains unique elements only.
* HashSet allows null value.
* HashSet class is non synchronized.
* HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.
* HashSet is the best approach for search operations.

1. HashSet<String> set=**new** HashSet<String>();
2. set.add("Ravi");
3. set.add("Vijay");
4. set.add("Ravi");
5. set.add("Ajay");
6. //Traversing elements
7. Iterator<String> itr=set.iterator();
8. **while**(itr.hasNext()){
9. System.out.println(itr.next());
10. }

**Data type**

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** Non-Primitive data types refer to objects and hence they are called **reference types.**The non-primitive data types Examples of non-primitive types include Strings, Arrays, Classes, Interface, etc

**ww**

**string** is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

[Java String | String Functions In Java With Examples | Edureka](https://www.edureka.co/blog/java-string/)

**Arrays**

* Arrays are an important structure to hold data.
* Java allows us to hold many s of the same type using arrays.
* It can be used with the help of a loop to access the elements by their index.

**Single-dimension Array:** In a single-dimension array, a list of variables of the same type can be accessed by a common name. You can initialize the array using the following syntax:

int a[] = new int[12];

**Multi–dimension Array:**In a multi-dimension array, your data is stored in a matrix form. Here, you can initialize the array using the following syntax:

int table[][]= new int[4][5];

**Class:** A class in Java is a blueprint which includes all your data.  A class contain fields(variables) and methods to describe the behaviour of an object

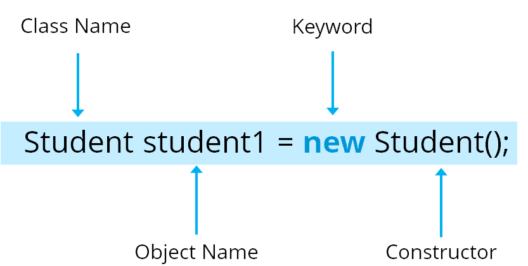
Class ABc{

Member variables

methods

}

But how can you access these member variables and methods? Here comes the concept of ***Object***.



An object is a major element in a class which has a state and behaviour. It is an instance of a class which can access your data.

**Interface:** Like a class, an interface can have methods and variables, but the methods declared in [interface](https://www.edureka.co/blog/java-collections/#interface) are by default abstract (only method signature, no body).

**Why we use non-primitive data type?**

Bcz they refer to the objects. Primitive data types are predefined in java whereas non-primitive types are created by programmer. Non-primitive data types can be used to call methods to perform certain operations.

Different types of locators?

Selenium uses what is called locators to find and match the elements of your page that it needs to interact with. There are 8 locators strategies included in Selenium:

* Identifier//
* Id
* Name
* Link
* DOM//
* XPath
* CSS
* UI-element//

**Wait statements?**

**Implicit wait**: If webDriver can’t find element immediately bcz of its unavailability , it will keep polling (250 mili sec)the DOM to get the element. If the element is not found within the specific time an NoSuchElement Exception will be raised.

Driver.manage().timeouts().implicitwait(10,Timeunit.SECONDS)

**Explicit wait**: You can set it up to wait for any condition you might like. Usaually, you can use some of prebuilt expectedconditions like clickable, visible, invisible etc. With the help of explicit wait for an element to be visible or clickable.

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

Wait.until(ExpectedConditions.presenceOfElementLocated(By.n0ame(“”);

Wait.until(ExpectedConditions.VisibiltyOfElementLocated(By.name(“”);

**Tell me about the Selenium WebDriver architecture?**

WebDriver is made of following components:  
– Language Bindings  
– JSON Wire Protocol  
– Browser Drivers  
– Real Browsers

When a test script is executed with the help of WebDriver, the following tasks are performed in the background:

– An HTTP request is generated and it is delivered to the browser driver for every Selenium Command.  
– The HTTP request is received by the driver through an HTTP server.  
– All the steps/instructions to be executed on the browser is decided by an HTTP server.  
– The HTTP server then receives the execution status and in turn sends it back to the automation scripts.

**Why we use TestNG?**

TestNG is a testing framework that is capable of making Selenium tests easier to understand and of generating reports that are easy to understand.

Annotations are easier to use and understand. Test cases can be grouped more easily. TestNG allows us to **create parallel tests**.

**TestNG**

[Top 17 TestNG Interview Questions - javatpoint](https://www.javatpoint.com/testng-interview-questions)

TestNG stands for "**Testing Next Generation**". It is an` automation testing framework used for java programming language .

It generates the report in a proper format, which includes the following information:

* Number of test cases executed.
* Number of test cases passed.
* Number of test cases failed.
* Number of test cases skipped

Multiple test cases can be grouped easily by converting them into a testng.xml file, in which you can set the priority of each test case that determines which test case should be executed first.

With the help of TestNG, you can execute the multiple test cases on multiple browsers known as cross-browser testing.

The TestNG framework can be easily integrated with other tools such as Maven. Jenkins, etc.

Annotations used in a TestNG framework are easily understandable such as @BeforeMethod, @AfterMethod, @BeforeTest, @AfterTest.

he sequence of actions is maintained by the annotations only.

What are the annotations used in the TestNG?

**The following are the annotations used in the TestNG are:**

* **Precondition annotations**  
  Precondition annotations are executed before the execution of test methods The Precondition annotations are @BeforeSuite, @BeforeClass, @BeforeTest, @BeforeMethod.
* **Test annotation**  
  Test annotation is specified before the definition of the test method. It is specified as @Test.
* **Postcondition annotations**  
  The postcondition annotations are executed after the execution of all the test methods. The postcondition annotation can be @AfterSuite, @AfterClass, @AfterTest, @AfterMethod.

What is the sequence of execution of all the annotations in TestNG?

**The sequence of execution of all the annotations in TestNG is given below:**

* @BeforeSuite
* @BeforeTest
* @BeforeClass
* @BeforeMethod
* @Test
* @AfterSuite
* @AfterTest
* @AfterClass
* @AfterMethod

How to set the priorities in TestNG?

If we do not prioritize the test methods, then the test methods are selected alphabetically and executed. If we want the test methods to be executed in the sequence we want, then we need to provide the priority along with the @Test annotation.

**Let's understand through an example.**

1. **package** com.javatpoint;
2. **import** org.testng.annotations.Test;
3. **public** **class** Test\_methods
4. {
5. @Test(priority=2)
6. **public** **void** test1()
7. {
8. System.out.println("Test1");
9. }
10. @Test(priority=1)
11. **public** **void** test2()
12. {
13. System.out.print("Test2");
14. }
15. }

 Define grouping in TestNG?

The group is an attribute in TestNG that allows you to execute the multiple test cases. For example, if we have 100 test cases of it\_department and 10 test cases of hr\_department, and if you want to run all the test cases of it\_department together in a single suite, this can be possible only through the grouping.

**Let's understand through an example.**

1. **package** com.javatpoint;
2. **import** org.testng.annotations.Test;
3. **public** **class** Test\_methods
4. {
5. @Test(groups="it\_department")
6. **public** **void** java()
7. {
8. System.out.println("I am a java developer");
9. }
10. @Test(groups="it\_department")
11. **public** **void** dot\_net()
12. {
13. System.out.println("I am a .Net developer");
14. }
15. @Test(groups="it\_department")
16. **public** **void** tester()
17. {
18. System.out.println("I am a software tester");
19. }
20. @Test (groups="hr")
21. **public** **void** hr()
22. {
23. System.out.print("I am hr");
24. }
25. }

**testng.xml**

1. ?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
3. <suite name="Suite">
4. <test name="It Company">
5. <groups>
6. <run>
7. <include name="it\_department"/>
8. </run>
9. </groups>
10. <classes>
11. <**class** name="com.javatpoint.Test\_methods"></**class**>
12. </classes>
13. </test>
14. </suite> <!-- Suite -->

What is dependency in TestNG?

When we want to run the test cases in a specific order, then we use the concept of dependency in TestNG.

**Two types of dependency attributes used in TestNG:**

* **dependsOnMethods**  
  The dependsOnMethods attribute tells the TestNG on which methods this test will be dependent on, so that those methods will be executed before this test method.

1. **package** com.javatpoint;
2. **import** org.testng.annotations.Test;
3. **public** **class** Login
4. {
5. @Test
6. **public** **void** login()
7. {
8. System.out.println("Login page");
9. }
10. @Test(dependsOnMethods="login")
11. **public** **void** home()
12. {
13. System.out.println("Home page");
15. }
16. }

* **dependsOnGroups**  
  It is similar to the dependsOnMethods attribute. It allows the test methods to depend on the group of test methods. It executes the group of test methods before the dependent test method.

1. **package** com.javatpoint;
2. **import** org.testng.annotations.Test;
3. **public** **class** Test\_cases
4. {
5. @Test(groups="test")
6. **public** **void** testcase1()
7. {
8. System.out.println("testcase1");
9. }
10. @Test(groups="test")
11. **public** **void** testcase2()
12. {
13. System.out.println("testcase2");
14. }
15. @Test(dependsOnGroups="test")
16. **public** **void** testcase3()
17. {
18. System.out.println("testcase3");
19. }
20. }

What is the importance of testng.xml file?

**The testng.xml file is important because of the following reasons:**

* It defines the order of the execution of all the test cases.
* It allows you to group the test cases and can be executed as per the requirements.
* It executes the selected test cases.
* In TestNG, listeners can be implemented at the suite level.
* It allows you to integrate the TestNG framework with tools such as Jenkins.

Why we use parameters?

Run the same test over and over again with different values

### How to pass the parameter in test case through testng.xml file?

We can also pass the value to test methods at runtime, we can achieve this by sending parameter values through the testng.xml file. We can use the **@Parameter** annotation:

1. @Parameters({"text"})
2. @Test
3. **public** **void** search()
4. {
5. // TODO Auto-generated method stub
6. System.setProperty("webdriver.chrome.driver", "D:\\chromedriver.exe");
7. WebDriver driver=**new** ChromeDriver();
8. driver.get("http://www.google.com/");
9. driver.findElement(By.name("q")).sendKeys("javatpoint tutorial");
10. }
11. }

**testng.xml file**

1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
3. <suite name="Suite">
4. <test name="It Company">
5. <parameter name="text" value="javatpoint"/>
6. <classes>
7. <**class** name="com.javatpoint.Web"></**class**>
8. </classes>
9. </test>
10. </suite> <!-- Suite -->

How can we disable the test case from running?

We can disable the test case from running by using the enabled attribute. We can assign the false value to the enabled attribute, in this way we can disable the test case from running.

1. **package** com.javatpoint;
2. **import** org.testng.annotations.Test;
3. **public** **class** Test\_cases
4. {
5. @Test(enabled=**false**)
6. **public** **void** testcase1()
7. {
8. System.out.println("testcase1");
9. }
10. @Test
11. **public** **void** testcase2()
12. {
13. System.out.println("testcase2");
14. }
16. }

### What is the difference between soft assertion and hard assertion?

**Soft Assertion:** In case of Soft Assertion, if TestNG gets an error during @Test, it will throw an exception when an assertion fails and continues with the next statement after the assert statement.

**Hard Assertion:** In the case of Hard Assertion, if TestNG gets an error during @Test, it will throw an AssertException immediately when an assertion fails and stops execution after the assert statement.

What is the use of @Listener annotation in TestNG?

TestNG provides different kinds of listeners which can perform different actions whenever the event is triggered. The most widely used listener in TestNG is ITestListener interface. The ITestListener interface contains methods such as onTestSuccess, onTestfailure, onTestSkipped, etc.

**Following are the scenarios that can be made:**

* If the test case is failed, then what action should be performed by the listener.
* If the test case is passed, then what action should be performed by the listener.
* If the test case is skipped, then what action should be performed by the listener.

[41 Top TestNG Interview Questions and Answers -- TOOLSQA](https://www.toolsqa.com/testng/testng-interview-questions/)

***What is the difference between a TestNG test and a TestNG test suite?***

TestNG test suite refers to a collection of tests that we can run simultaneously with the help of the TestNG XML file. On the other side, a TestNG test is a single test case file, and when we say “we are running a TestNG test case,” we simply mean we are running a single test case file.

## How To Generate Emailable Report In TestNG?

Emailable reports are generated in TestNG to let the user send their test reports to other team members. Emailable-reports do not require any extra work from the tester, and they are a part of overall test execution. To generate emailable reports, first, run the TestNG test class if you have not already.

Once we have run the test case, a new folder generates in the same directory with the name ***test-output***.

## How To Generate Index File In TestNG?

Emailable reports are a type of summary reports that one can transfer to other people in the team through any medium. Index reports, on the other hand, contains the index-like structure of different parts of the report, such as failed tests, test file, passed test, etc.

***What do you understand by asserts in TestNG?***

An asset is a piece of code that helps us verify if the expected result and the actual result are equal or not. In TestNG, we leverage the inbuilt “Assert” class and a lot of its method to determine whether the test case passed or failed. Additionally, in TestNG, a test case acts as a “pass” if none of the assert methods throws an exception during the execution. The syntax for TestNG assert is:

Assert.Method(actual, expected, message);

***Describe any five common TestNG assertions.***

The five common TestNG assertions are:

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

***What is meant by invocationCount in TestNG?***

The invocationCount is an attribute that defines the number of times a test method has to run in a single execution. So, if the invocationCount sets as 5, then the test method would run five times each time I execute the TestNG test case.

Syntax for invocationCount:

@Test (invocationCount = 5)

## What is Parallel Testing and Why is it important?

 In parallel testing, the program’s multiple parts (or modules) execute together, saving the testers a lot of time and effort.

<test name = "Parallel Tests" parallel = "methods">

Parallel=”methods””

all the methods that come under the annotation @Test will run parallel when we execute the test suite.

Parallel=”tests”

**executing each test inside a suite in parallel**, that is, each test that is part of the test suite execution will be executed in its own separate respective thread.

## What are DataProviders in TestNG?

The DataProviders in TestNG are another way to pass the parameters in the test function, the other one being TestNG parameters.

@DataProvider (name = “name\_of\_dataprovider”)

public Object[][] dpMethod() {

return new Object [][] { values}

}

* The TestNG DataProvider (the annotation part) contains only one single attribute, which is its name. It is always a string type in nature. For example, “name\_of\_dataprovider,” as mentioned above.
* If the tester has not specified the name of the dataprovider, then the method name becomes the dataprovider name by default.
* TestNG dataprovider returns a 2d list of objects.

***Why is the reporter class used in TestNG?***

The reporter class in TestNG logs the tester defined messages into the reports generated by TestNG. These logged messages then print into the reports, which we can share with the team.

Reporter.log(“message”);

**Import Required:***import org.testng.Reporter;*

Reporter is used for format the reports. SparkReporter is an opensource so we config like title, theme, name…..

***What do we need to generate a customized report in TestNG?***

A customized report in TestNG generates with the help of TestNG listeners. Using the interface ITestListener in TestNG, we can control the events such as method start, method pass, fail, etc., and according to these events, a tester can log appropriate messages.

***ITestContext*** and ***ITestResult***. So, the term ‘***ITestResult’ is an interface that describes the result of the test***. Therefore ‘result‘ has been passed as its instance in the syntax. Whereas***‘***ITestContext ***‘ is a class that defines an instance ‘***context***‘, which contains all the information about the test run***. We can use this information to pass to our listeners, and they can proceed with their queries.

**Maven**

It Is build management tool and It will manage your selenium test project's build compilation, documentation and other related project tasks It self. It will helps you to create right project structure, add and manage jar files In project's build path etc..

Maven uses POM.xml configuration file which kept all project configuration related Information. For selenium, You need to provide selenium webdriver version related Information In POM.xml file and then It will download all required jar files automatically and store It In local repository called m2. Later on If you wants to change version of selenium webdriver then you need to modify version In POM.xml file only. Then maven will download new version jar files automatically and store In local repository. That means If you upgrade any dependency's version In POM.xml file, First It will check that version's jar files are available or not In local repository. If available then fine else It will download them from maven central repository.

[Top 20 Maven Interview Questions & Answers (guru99.com)](https://www.guru99.com/maven-interview-questions.html)

STLC

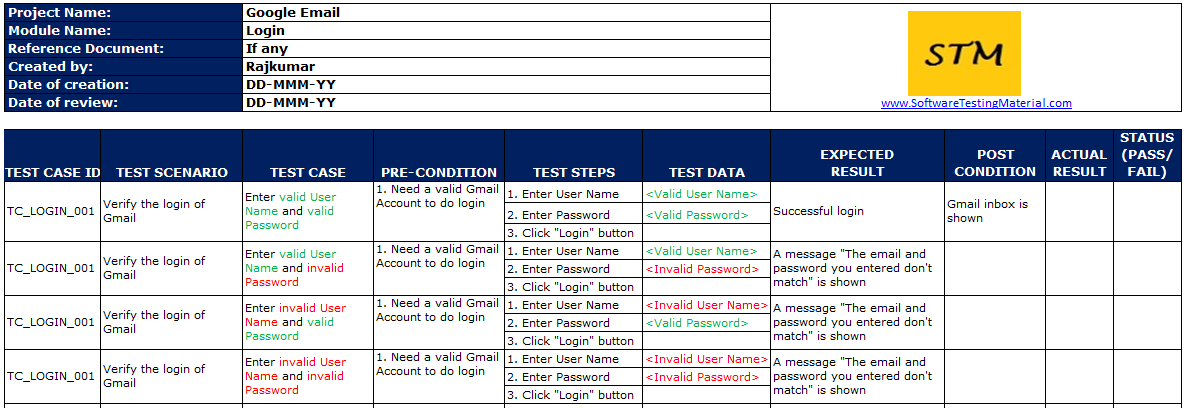
Regression, smoke, functional, bug life cycle?

**Regression Testing**is the process of testing the modified parts of the code and the parts that might get affected due to the modifications to ensure that no new errors have been introduced in the software after the modifications have been made. Regression means return of something and in the software field, it refers to the return of a bug.

Diff between periorty and severity?

Action classes?

How to write Test cases?



**Diffrnt between error, bug and failure?**

Error: error is observed by Developer. Error like syntax error, logical error, airthmatic error.

Bug: This is observed by Tester. If any functionality of application is not working, according to cust. Requirement.

Failure: Failure is noticed by Cust.. if cust is working on software and found some failure, that is also called production environment.

**What is Test Cases ?**

A set of action perform to verify the functionality or feature of a software application is known as the test cases.

**Need of Test  Cases - Why Do We write Test Cases ?**

Writing of test cases is a most important part of software testing and also it is a major activity done by testing team , development team and management team .

* If there is no document for an application , then test cases use as baseline document.
* Test cases brings together the whole testing process.
* Test cases use for training purpose on product for new team.
* Test cases act as the proof for the customer and manager that we have cover all the scenarios.
* Test cases act as the bases document for the preparation of automation script

.

**Test cases for pen, lift, search box?**

**Pen**

1. Verify the pen type gel pen, ball pen, Sketch pen, normal pen, ink pen.
2. Verify ink color green, red, blue, black.
3. Check that user able to open the pen cap easily.
4. Check that user able to write on paper.
5. Check that user able to write smoothly.
6. Verify that user able to hold the pen comfortably.
7. Verify the material used for pen(Plastic, metallic as per CRS).
8. Check that pen company name and logo is visible.
9. Check the nib size.
10. Check the water proof ink(if not a gel pen).
11. Check the outer body color of pen.
12. Check the text written by pen.(erasable or permanent)
13. Check if pen support multiple refills.
14. Check that ink leakage when tilled it down.

**Search**

1. Without entering anything in search box, click on search button.
2. Check the cancel option to clear the search keyword.
3. Click in search field and enter the key.
4. Enter any one character in search box and press enter key.
5. Enter only special characters and click on search button.
6. Enter alphabetical characters and click on search button.
7. Enter alphabetical and special characters then click on search button.
8. Enter string more than char limit of the search field.
9. Enter string in search box with spaces(before string, after string, in between) and verify the results.
10. Verify this on all pages where search box is available.
11. Try drag and drop image or other file in search field then verify the results.
12. Check weather auto suggestion is working or not.
13. Check if search box is applied on water mark updated in search box.

**Lift**

1. Verify that the lift moves to specific floor as the button of the floor is pressed.
2. Verify that the lift is stop when buttons are at particular floor are pressed.
3. Verify that if power failure, lift get halted to a particular floor.
4. Verify that if door is about to close and anybody place his/her foot between door and door senses it and stops.
5. Verify that door does not open while lift is moving.
6. Verify that the door open once it reaches the floor pressed by person.
7. Verify that lift gives alert and warning alarm in case capacity limit is reached.
8. Verify that door locks and door open automatically when destination is reached.
9. Verify the First come first Priority.

## What is Select Class in Selenium?

In HTML, the dropdowns are generally implemented either using the ***<select>*** tag or the ***<input>*** tag. Selenium WebDrivers provides a class called “***Select***” class.

**Select**” class is provided by the “**org.openqa.selenium.support.ui**” package of Selenium WebDriver. You can create an object of the Select class, by-passing the object of the “[***WebElement***](https://www.toolsqa.com/selenium-webdriver/webelement-commands/)” class, which shows the object returned by the corresponding [***locator***](https://www.toolsqa.com/selenium-webdriver/selenium-locators/) of the WebElement.

Select select = new Select(WebElement webelement);

The “**Select**” class provides various methods for handling the dropdown operations.

**select a value from a dropdown**

* *selectByIndex*
* *selectByValue*
* *selectByVisibleText*

SelectByIndex- We provide an integer value as the index number as an argument. Index is start from 0. Ex 1-blue, 2-white, 3-red, 4-pink

Select se = new Select(driver.findElement(By.xpath("//\*[@id='oldSelectMenu']")));

// Select the option by index

se.selectByIndex(3);-pink

SelectByValue- We provide a string value as the value as an argument. Ex 1-blue, 2-white, 3-red, 4-pink

Select se = new Select(driver.findElement(By.xpath("//\*[@id='oldSelectMenu']")));

// Select the option with value "4”

se.selectByValue("4"); - pink

selectByVisibleText- This method enables one to select one option from the dropdown or multi-select dropdown based on the dropdown text. You need to pass the String value of the <select> element as an argument.

Select se = new Select(driver.findElement(By.xpath("//\*[@id='oldSelectMenu']")));

// Select the option using the visible text

se.selectByVisibleText("White");

**isMultiple()**” method, which determines whether the web element in say supports multiple selections. It returns a boolean value, i.e., True/False, without taking any argument.

In DOM

<select multiple name=”cars” id=”cars”>

Select oSel = new Select(driver.findElement(By.xpath(//\*[@id='cars']);

if(oSel.isMultiple()){

//Selecting multiple values by index

oSel.selectByIndex(1);

oSel.selectByIndex(2);

//Or selecting by values

oSel.selectByValue("volvo");

oSel.selectByValue("audi");

//Or selecting by visible text

oSel.selectByVisibleText("Volvo");

oSel.selectByVisibleText("Opel")

**How to get options from a dropdown in Selenium?**

the Select class provides the following methods to get the options of a dropdown:

* *getOptions()*
* *getFirstSelectedOption()*
* *getSelectedOptions()*

 this method returns all the options of the dropdown as a list of ***WebElement***.

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

List<WebElement> genders=s.getOptions();

**getFirstSelectedOption**- This method returns the first selected option of the dropdown. If it is a single-select dropdown, this method will return the selected value of the dropdown, and if it is a multi-select dropdown, this method will return the first selected value of the dropdown.

Select select = new Select(driver.findElement(By.id("oldSelectMenu")));

// Get the first selected option of the dropdown

WebElement firstSelectedOption = select.getFirstSelectedOption();

Using this method, we can retrieve the first selected option of a dropdown (be it single-select or multi-select ).

**getAllSelectedOptions**: This method returns all the selected options of the dropdown. If it is a single-select dropdown, this method will return the only selected value of the dropdown, and if it is a multi-select dropdown, this method will return all the selected values of the dropdown.

Select select = new Select(driver.findElement(By.id("oldSelectMenu")))

// Get all the selected option of the dropdown

List<WebElement> selectedOptions = select.getAllSelectedOptions();

#### Using this method, we can retrieve all the selected options of a dropdown (be it single-select or multi-select ).

**What is Alert? How to handle?**

Alert is a message/notification box that notifies the user about some information or asks for permission to perform a certain kind of operation. It may be used for warning purposes as well.

In [Selenium](https://www.browserstack.com/selenium), there are three types of Alerts as follows:

Simple Alert:

This alert is used to notify a simple warning message with an ‘OK’ button

Prompt Alert:

This alert will ask the user to input the required information to complete the task.

Confirmation Alert:

This alert is basically used for the confirmation of some tasks.

**Alert Handling:**

**1. Void dismiss():** This method is used when the ‘Cancel’ button is clicked in the alert box.

driver.switchTo().alert().dismiss();

**2. Void accept():** This method is used to click on the ‘OK’ button of the alert.

driver.switchTo().alert().accept();

**String getText():** This method is used to capture the alert message.

driver.switchTo().alert().getText();

**4. Void sendKeys(String stringToSend):** This method is used to send data to the alert box.

driver.switchTo().alert().sendKeys("Text");

**Pop Up Handling?**

Popup is a window that displays or pops up on the screen due to some activity.

**Driver.getWindowHandles();**

In order to handle the opened windows by Selenium webdriver, you can use Driver.getWindowHandles() to switch between the windows.

**2. Driver.getWindowHandle();**

When the webpage is loaded, you can handle the main window by using driver.getWindowHandle(). It will handle the current window that uniquely identifies within the driver instance.

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

System.***out***.println("Total window size " + s.size());

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

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

//it=s.iterator();

//System.out.println(it.next());

String mainwindow=it.next();

String popup1=it.next();

String popup2=it.next();

//String popup3=it.next();

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

System.***out***.println(popup1);

System.***out***.println(popup2);

//System.out.println(popup3);

driver.switchTo().window(popup1);

driver.close();

driver.switchTo().window(popup2);

driver.close();

//driver.switchTo().window(popup3);

//driver.close();

driver.switchTo().window(mainwindow);

driver.switchTo().defaultContent();

## What are Access Modifiers?

In Java, access modifiers are used to set the accessibility (visibility) of classes, interfaces, variables, methods, constructors, data members, and the setter methods.

There are four access modifiers keywords in Java and they are:

|  |  |
| --- | --- |
| Modifier | Description |
| Default | declarations are visible only within the package (package private) |
| Private | declarations are visible within the class only |
| Protected | declarations are visible within the package or all subclasses |
| Public | declarations are visible everywhere |

## Default Access Modifier

If we do not explicitly specify any access modifier for classes, methods, variables, etc, then by default the default access modifier is considered. For example,

package defaultPackage;

class Logger {

void message(){

System.out.println("This is a message");

}

}

Here, the Logger class has the default access modifier. And the class is visible to all the classes that belong to the defaultPackage package. However, if we try to use the Logger class in another class outside of defaultPackage, we will get a compilation error.

## Private Access Modifier

When variables and methods are declared private, they cannot be accessed outside of the class. For example,

class Data {

// private variable

private String name;

}

public class Main {

public static void main(String[] main){

// create an object of Data

Data d = new Data();

// access private variable and field from another class

d.name = "Programiz";

}

}

In the above example, we have declared a private variable named name and a private method named display(). When we run the program, we will get the following error:

Main.java:18: error: name has private access in Data

d.name = "Programiz";

 if we need to access those private variables. In this case, we can use the getters and setters method. For example,

class Data {

private String name;

// getter method

public String getName() {

return this.name;

}

// setter method

public void setName(String name) {

this.name= name;

}

}

public class Main {

public static void main(String[] main){

Data d = new Data();

// access the private variable using the getter and setter

d.setName("Programiz");

System.out.println(d.getName());

}

}

**This** keyword is always point to the current object. So if local name and global name is same then local will hide the global vairiable. So for access the global variable we use This keyword.

Here, we have used the setter method (setName()) to assign value to the variable and the getter method (getName()) to access the variable.

**Protected Access Modifier**

When methods and data members are declared protected, we can access them within the same package as well as from subclasses. For example,

class Animal {

// protected method

protected void display() {

System.out.println("I am an animal");

}

}

class Dog extends Animal {

public static void main(String[] args) {

// create an object of Dog class

Dog dog = new Dog();

// access protected method

dog.display();

}

}

**Output**:

I am an animal

We then created an object dog of the Dog class. Using the object we tried to access the protected method of the parent class.

Since protected methods can be accessed from the child classes, we are able to access the method of Animal class from the Dog class.

**Public Access Modifier**

When methods, variables, classes, and so on are declared public, then we can access them from anywhere. The public access modifier has no scope restriction. For example,

// Animal.java file

// public class

public class Animal {

// public variable

public int legCount;

// public method

public void display() {

System.out.println("I am an animal.");

System.out.println("I have " + legCount + " legs.");

}

}

// Main.java

public class Main {

public static void main( String[] args ) {

// accessing the public class

Animal animal = new Animal();

// accessing the public variable

animal.legCount = 4;

// accessing the public method

animal.display();

}

}

**Output**:

I am an animal.

I have 4 legs.

* The public class Animal is accessed from the Main class.
* The public variable legCount is accessed from the Main class.
* The public method display() is accessed from the Main class.

# Java instanceof Operator

The instanceof operator in Java is used to check whether an object is an instance of a particular class or not.

Its syntax is

objectName instanceOf className;

Here, if objectName is an instance of className, the operator returns true. Otherwise, it returns false.

**Example: Java instanceof**

class Main {

public static void main(String[] args) {

// create a variable of string type

String name = "Programiz";

// checks if name is instance of String

boolean result1 = name instanceof String;

System.out.println("name is an instance of String: " + result1);

// create an object of Main

Main obj = new Main();

// checks if obj is an instance of Main

boolean result2 = obj instanceof Main;

System.out.println("obj is an instance of Main: " + result2);

}

}

### Abstract Class and Method

Though abstract classes cannot be instantiated, we can create subclasses from it. We can then access members of the abstract class using the object of the subclass. For example,

abstract class Language {

// method of abstract class

public void display() {

System.out.println("This is Java Programming");

}

}

class Main extends Language {

public static void main(String[] args) {

// create an object of Main

Main obj = new Main();

// access method of abstract class

// using object of Main class

obj.display();

}

}

**Output**

This is Java programming

In the above example, we have created an abstract class named Language. The class contains a regular method display().

We have created the Main class that inherits the abstract class. Notice the statement,

obj.display();

Here, obj is the object of the child class Main. We are calling the method of the abstract class using the object obj.

## Implementing Abstract Methods

If the abstract class includes any abstract method, then all the child classes inherited from the abstract superclass must provide the implementation of the abstract method. For example,

abstract class Animal {

abstract void makeSound();

public void eat() {

System.out.println("I can eat.");

}

}

class Dog extends Animal {

// provide implementation of abstract method

public void makeSound() {

System.out.println("Bark bark");

}

}

class Main {

public static void main(String[] args) {

// create an object of Dog class

Dog d1 = new Dog();

d1.makeSound();

d1.eat();

}

}

**Output**

Bark bark

I can eat.

**Abstraction**

Abstraction is an important concept of object-oriented programming that allows us to hide unnecessary details and only show the needed information.

A practical example of abstraction can be motorbike brakes. We know what brake does. When we apply the brake, the motorbike will stop. However, the working of the brake is kept hidden from us.

abstract class Animal {

abstract void makeSound();

}

class Dog extends Animal {

// implementation of abstract method

public void makeSound() {

System.out.println("Bark bark.");

}

}

class Cat extends Animal {

// implementation of abstract method

public void makeSound() {

System.out.println("Meows ");

}

}

class Main {

public static void main(String[] args) {

Dog d1 = new Dog();

d1.makeSound();

Cat c1 = new Cat();

c1.makeSound();

}

}

**Output**:

Bark bark

Meows

* We use the abstract keyword to create abstract classes and methods.
* An abstract method doesn't have any implementation (method body).
* A class containing abstract methods should also be abstract.
* We cannot create objects of an abstract class.
* To implement features of an abstract class, we inherit subclasses from it and create objects of the subclass.
* A subclass must override all abstract methods of an abstract class.

**Interface**

An interface is a fully abstract class. It includes a group of abstract methods (methods without a body).

We use the interface keyword to create an interface in Java. For example,

interface Language {

public void getType();

public void getVersion();

}

## Implementing an Interface

Like abstract classes, we cannot create objects of interfaces.

To use an interface, other classes must implement it. We use the implements keyword to implement an interface.

### Example 1: Java Interface

interface Polygon {

void getArea(int length, int breadth);

}

// implement the Polygon interface

class Rectangle implements Polygon {

// implementation of abstract method

public void getArea(int length, int breadth) {

System.out.println("The area of the rectangle is " + (length \* breadth));

}

}

class Main {

public static void main(String[] args) {

Rectangle r1 = new Rectangle();

r1.getArea(5, 6);

}

}

**Output**

The area of the rectangle is 30

In the above example, we have created an interface named Polygon. The interface contains an abstract method getArea().

Here, the Rectangle class implements Polygon. And, provides the implementation of the getArea() method.

## Extending an Interface

Similar to classes, interfaces can extend other interfaces. The extends keyword is used for extending interfaces. For example,

interface Line {

// members of Line interface

}

// extending interface

interface Polygon extends Line {

// members of Polygon interface

// members of Line interface

}

Here, the Polygon interface extends the Line interface. Now, if any class implements Polygon, it should provide implementations for all the abstract methods of both Line and Polygon.

**Extending Multiple Interfaces**

An interface can extend multiple interfaces. For example,

interface A {

...

}

interface B {

...

}

interface C extends A, B {

...

}

Interfaces are also used to achieve multiple inheritance in Java. For example,

interface Line {

…

}

interface Polygon {

…

}

class Rectangle implements Line, Polygon {

…

}

**Encapsulation**

**File Handling**

File Handling in Java permits us to **create, read, update, and delete the files**, which are stored on the local file system. There are two types of File handling in Java – FileWriter, and FileReader, which can perform all the file operations in Java Program.

Java File Handling The File class from the java.io package, allows us to work with files. To use the File class, create an object of the class, and specify the filename or directory name:

**ScrollDown page using Javascript**

**Verticially scroll down**

JavascriptExecutor js=(javascriptExecutor)driver;

js.executeScript(“window.scrollBy(0,1000)”);

**To scroll down the web page by the visibility of the element.**

JavascriptExecutor js=(javascriptExecutor)driver;

Driver.get(“…..”);

WebElement element=driver.findElement(By.LinkText(“”));

Js.executeScript(“argument[0].scrollIntoView();”, Element);

In the above code, we first launch the given url in Chrome browser. Next, scroll the page until the mentioned element is visible on the current page. Javascript method scrollIntoView() scrolls the page until the mentioned element is in full view :

"arguments[0]" means first index of page starting at 0.

**Scenario 3: To scroll down the web page at the bottom of the page.**

//This will scroll the web page till end.

js.executeScript("window.scrollTo(0, document.body.scrollHeight)");

"document.body.scrollHeight" returns the complete height of the body i.e web page.

**Mouse Hover action perform**

Mouse hover action is basically an action where a user places a mouse over a designated area like a hyperlink.

***moveToElement(WebElement target): Moves the mouse to the middle of the element.***

First, ***instantiate an Actions class:***

WebElement golf=driver.findElement(By.*xpath*("//\*[@id='header-navigation']/div[1]/ul/li[2]/a"));

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

act.moveToElement(golf).build().perform();

* **Build**: build() method is used to generate a composite action containing all actions.  
  In this case, it’s just a single move mouse action. If you observe, we have not invoked it in our above command. This is because the build is executed in the perform method internally.
* **Perform**: perform() method performs the actions we have specified. But before that, it internally invokes build() method first. After the build, the action is performed.

**How to move the mouse cursor to an element offset?**

**moveToElement(WebElement target, int xOffset, int yOffset): Moves the mouse to an offset from the top-left corner of the element.**

Basically, the purpose of this method is to take a mouse on specific points on target elements.

**why do we need to move the mouse to any specific point on the element?**

 An answer to this question is, there are some cases where to perform some actions on an element, it is mandatory to move the mouse to a specific location on the element only.

Let’s consider **Slider** which is used to select some range

//Instantiate Action Class

        Actions actions = new Actions(driver);

        //Retrieve WebElemnt 'slider' to perform mouse hover

     WebElement slider = driver.findElement(By.id("slider"));

     //Move mouse to x offset 50 i.e. in horizontal direction

     actions.moveToElement(slider,50,0).perform();

     slider.click();

**OR**

WebElement e=driver.findElements(By.*xpath*("//div[@class='noUi-base']")).get(1);

actions.clickAndHold(e).moveByOffset(100,0).release().build().perform();

**Action Classes**

**Actions** class is a collection of individual **Action** that you want to perform. For e.g. we may want to perform a mouse click on an element. In this case we are looking at two different **Action**

1. Moving the mouse pointer to the element
2. Clicking on the element

**Actions class & Action class** reside in org.openqa.selenium.Interactions package of **WebDriver API**. To consume these, import their packages:

import org.openqa.selenium.interactions.Actions;

import org.openqa.selenium.interactions.Action;

A modifier key is a key that modifies the action of another key when the two are pressed together like Shift, Control & Alt.

Find the web-element and generate the sequence:

WebElement element = driver.findElement(By strategy to identify element);

actions.keyDown(element, Keys.SHIFT);

actions.sendKeys(“TextToBeConvertAndSendInUpperCase”);

actions.keyUp(Keys.SHIFT);

 all actions can be clubbed together as below:

actions.keyDown(element, Keys.SHIFT).sendKeys(“TextToBeConvertAndSendInUpperCase”).keyUp(Keys.SHIFT).perform()

## Methods in Actions class of Selenium

There are a lot of methods in this class which can be categorized into two main categories:

* ***Keyboard Events***
* ***Mouse Events***

;

**Different Methods for performing Keyboard Events:**

* keyDown(modifier key): Performs a modifier key press.
* sendKeys(keys to send ): Sends keys to the active web element.
* keyUp(modifier key): Performs a modifier key release.

**Different Methods for performing Mouse Events:**

* **click():** Clicks at the current mouse location.
* **doubleClick():** Performs a double-click at the current mouse location.
* **contextClick() :** Performs a context-click at middle of the given element.
* **clickAndHold():** Clicks (without releasing) in the middle of the given element.
* **dragAndDrop(source, target):** Click-and-hold at the location of the source element, moves to the location of the target element
* **dragAndDropBy(source, xOffset, yOffset):** Click-and-hold at the location of the source element, moves by a given offset
* **moveByOffset(x-offset, y-offset):** Moves the mouse from its current position (or 0,0) by the given offset
* **moveToElement(toElement):** Moves the mouse to the middle of the element
* **release():** Releases the depressed left mouse button at the current mouse location

right click-

actions.contextClick(btnElement).perform();

double click-

//Double Click the button

actions.doubleClick(btnElement).perform();

System.out.println("Button is double clicked");

//Following code just click on OK button on alert , this differs

//depending upon application(under test) specific test case

// Accept the Alert

driver.switchTo().alert().accept();

System.out.println("Double click Alert Accepted");

Drag and drop

//Actions class method to drag and drop

Actions builder = new Actions(driver);

WebElement from = driver.findElement(By.id("draggable"));

WebElement to = driver.findElement(By.id("droppable"));

//Perform drag and drop

builder.dragAndDrop(from, to).perform();

//verify text changed in to 'Drop here' box

String textTo = to.getText();

if(textTo.equals("Dropped!")) {

System.out.println("PASS: Source is dropped to target as expected");

}else {

System.out.println("FAIL: Source couldn't be dropped to target as expected");

}

driver.close();

| **Sr. No.** | **Key** | **==** | **equals() method** |
| --- | --- | --- | --- |
| 1 | Type | == is an operator. | equals() is a method of Object class. |
| 2 | Comparision | == should be used during reference comparison. == checks if both references points to same location or not. | equals() method should be used for content comparison. equals() method evaluates the content to check the equality. |
| 2 | Object | == operator can not be overriden. | equals() method if not present and Object.equals() method is utilized, otherwise it can be overridden. |

The String **compareTo()** method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two string variables. If:

* **s1 == s2** :0
* **s1 > s2**  :positive value
* **s1 < s2**  :negative value

|  |  |
| --- | --- |
| **Mutable** | **Immutable** |
| We can change the value of mutable objects after initialization. | Once an immutable object is initiated; We can not change its values. |
| The state can be changed. | The state can not be changed. |
| In mutable objects, no new objects are formed. | In immutable objects, a new object is formed when the value of the object is altered. |
| It provides methods to change the object. | It does not provide any method to change the object value. |
| It supports get() and set() methods to dela with the object. | It only supports get() method to pass the value of the object. |
| Mutable classes are may or may not be thread-safe. | Immutable classes are thread-safe. |
| The essentials for creating a mutable class are methods for modifying fields, getters and setters. | The essentials for creating an immutable class are final class, private fields, final mutable objects. |

|  |  |  |
| --- | --- | --- |
| **String** | | **StringBuffer** |
| 1) | String class is immutable. | StringBuffer class is mutable. |
| 2) | String is slow and consumes more memory when you concat too many strings because every time it creates new instance. | StringBuffer is fast and consumes less memory when you cancat strings. |
| 3) | String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method. | StringBuffer class doesn't override the equals() method of Object class. |

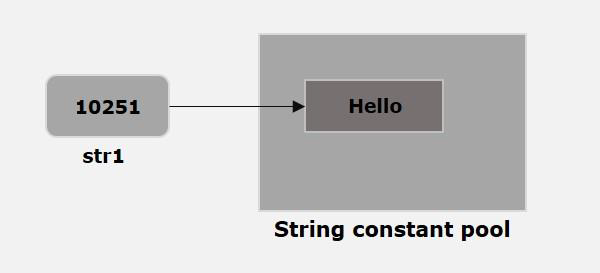
| **Sr. No.** | **Key** | **String Buffer** | **String Builder** |
| --- | --- | --- | --- |
| 1 | Basic | StringBuffer was introduced with the initial release of Java | It was introduced in Java 5 |
| 2 | Synchronized | It is synchronized | It is not synchronized |
| 3 | Performance | It is thread-safe. So, multiple threads can’t access at the same time, therefore, it is slow. | It is not thread-safe hence faster than String Buffer |
| 4 | Mutable | It is mutable. We can modify string without creating an object | It is also mutable |
| 5 | Storage | Heap | Heap |

**Difference between String Literal and String Object?**

When you store a String as

String str1 = "Hello";

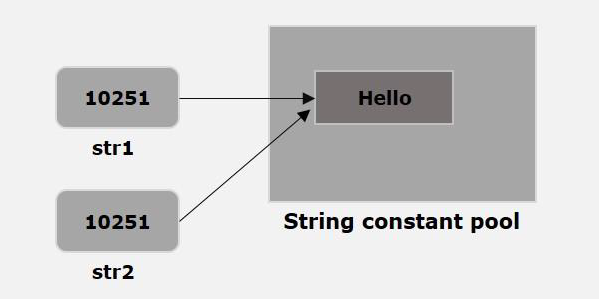
directly, then JVM creates a String object with the given value in a separate block of memory known as String constant pool.



And whenever we try to create another String as

String str2 = "Hello";

JVM verifies whether any String object with the same value exists in the String constant pool, if so, instead of creating a new object JVM assigns the reference of the existing object to the new variable.



String s1="ABC";

String s2="ABC";

String s3=**new** String("ABC");

String s4=**new** String("ABC");

String s5=s1;

String s6=**new** String(s3);

String s7="abc";

String s8 = **new** String("abc");

System.***out***.println(s3==s6);//false

System.***out***.println(s3.equals(s6));//true

System.***out***.println(s3==s7);//false

System.***out***.println(s5.equals(s7));//false

System.***out***.println(s1==s2);//true

System.***out***.println(s1==s3);//false

System.***out***.println(s3==s4);;//false

System.***out***.println(s1==s5); //true

System.***out***.println(s2==s6);//false

System.***out***.println(s1.equals(s2));//true

System.***out***.println(s1.equals(s3));//true

System.***out***.println(s3.equals(s4));//true

System.***out***.println(s1.equals(s6));//true

System.***out***.println(s4.equals(s6));//true

**Wrapper classes in java**

The **wrapper class in Java** provides the mechanism to convert primitive into object and object into primitive.

The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

where we need to use the wrapper classes.

* **Change the value in Method:** Java supports only call by value. So, if we pass a primitive value, it will not change the original value. But, if we convert the primitive value in an object, it will change the original value.
* **Serialization:** We need to convert the objects into streams to perform the serialization. If we have a primitive value, we can convert it in objects through the wrapper classes.
* **Synchronization:** Java synchronization works with objects in Multithreading.
* **java.util package:** The java.util package provides the utility classes to deal with objects.
* **Collection Framework:** Java collection framework works with objects only. All classes of the collection framework (ArrayList, LinkedList, Vector, HashSet, LinkedHashSet, TreeSet, PriorityQueue, ArrayDeque, etc.) deal with objects only

Autoboxing

The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing, for example, byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short.

Int a=20;

Integer i=Integer.valueOf(a); //converting int into Integer explicitly

Integer j=a; //autoboxing, now compiler will write Integer.valueOf(a) internally

Syso(a+ “” +i+”” +j);

o/p- 20 20 20

Unboxing

The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing. It is the reverse process of autoboxing.

Integer a = new Integer(3);

Int i=a.IntValue();

Int j=a;

System.out.println(a+" "+i+" "+j);

o/p – 3 3 3

**Garbage Collection**

**Garbage Collection in Java** is a process by which the programs perform memory management automatically.

it is a way to destroy the unused objects.

Advantage of Garbage Collection

* 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.

## How can an object be unreferenced?

There are many ways:

* By nulling the reference
* By assigning a reference to another
* By anonymous object etc.

1) By nulling a reference:

1. Employee e=**new** Employee();
2. e=**null**;

2) By assigning a reference to another:

1. Employee e1=**new** Employee();
2. Employee e2=**new** Employee();
3. e1=e2;//now the first object referred by e1 is available for garbage collection

3) By anonymous object:

1. **new** Employee();

**The finalize() method** is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

1. **protected** **void** finalize(){}

## gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

**public** **static** **void** gc(){}

**public** **class** TestGarbage1{

**public** **void** finalize(){System.out.println("object is garbage collected");}

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

  TestGarbage1 s1=**new** TestGarbage1();

  TestGarbage1 s2=**new** TestGarbage1();

  s1=**null**;

  s2=**null**;

  System.gc();

 }

}

**Class**

Class is a collection of objects which have common properties. We create object of the class.

**Object** is an instance of class and it represent state and behavior.