**Selenium Webdriver:**

* Selenium Web driver is an automation tool used for testing.
* It is an open source.
* Selenium Web driver is language and platform independent. (i.e.) it is supported in Windows xp, 7, 8 or Linux.
* Selenium Latest Version 3.0.
* Following Languages are supported in Selenium Web driver:
* Java
* PHP
* C/C#
* Python
* Selenium can be implemented in following browsers:
* Firefox
* Google Chrome
* Internet Explorer
* Safari
* Opera
* Default web browser used in selenium is “**Firefox**”. Syntax to invoke **Firefox** browser.

Webdriver driverobj = new FirefoxDriver();

**Points to be Remember:**

1. **Webdriver** => Interface
2. **FirefoxDriver**=> Class
3. Here creating the reference to the Webdriver(interface) and object/instance to the class FirefoxDriver.
4. We cannot create object/instance for an Interface like Webdriver driver = new Webdriver();
5. At the same time we can use FireFoxDriver driverob= new FirefoxDriver(), but for chrome and Internet Explorer we have to create separate objects, in order to avoid that we follow Webdriver driver = new FirefoxDriver(); It is useful in case of parallel testing in multiple browsers.  
   **eg:**Webdriver driver;

switch(browserChoice){

case IE:

driver = new InternetExplorerDriver();

break;

case FF:

driver = new FirefoxDriver()

break;  
}

Selenium to invoke IE or Chrome or Safari Browser, we need the third party exe files i.e. ChromeDriver, InternetExplorerDriver, safariDriver for these browsers. Then use the below syntax:

* System.setProperty(“webdriver.ie.driverobj”,”C://Testing/IEDriverServer.exe”);

Webdriverdriverobj = new InternetExploreDriver();

* System.setProperty(“webdriver.chrome.driverobj”,”C:/Testing/chromedriver. exe”);

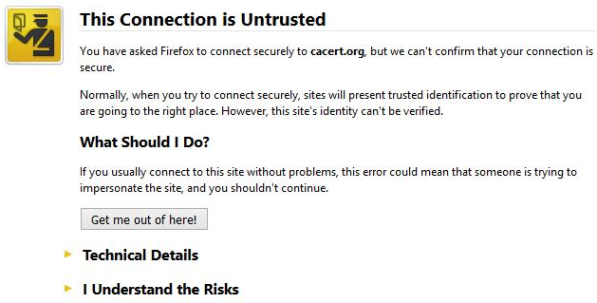
Webdriverdriverobj = new ChromeDriver();  
**Why we go for System.setProperty() for chrome and IE()?**  
WebDriver uses native browser approach. Selenium offers inbuilt driver for Firefox but not for other browsers. All drivers (Chrome Driver, IE driver, etc.) are built based on the special JS Engine used by each browser.

* To pass URL in Selenium, we can use two methods GET() and Navigate().  
  **driver.get("URL")**Syntax: get(java.lang.String url)url - The URL to load. It is always best to use a fully qualified URL.Example: driver.get("http://www.google.com");  
   which loads a new web page in the current browser window.  
  **driver.navigate().to("URL")**  
  There is another overloaded method navigate().to(java.net.URL url) which makes it easy to pass in a URL. Here java.net.URL class represents a URL which has set of methods to manipulate URL in Java.  
  driver.navigate().refresh();  
  driver.navigate().forward();

driver.navigate().back();  
**GET() VS Navigate()**

|  |  |
| --- | --- |
| GET() | Navigate() |
| 1. In **GET(),** we are just pass URL. Histories are not stored. | 1. **Navigate()** interface used to pass URL at the same time it stored in the browser history. So that it can move forward and backward. |
| 1. Webdriver will wait until the page has fully loaded before returning the control to test or script. If there are many ajax calls in the current page which webdriver is loading then the webdriver may not know when the page has loaded completely. If you need to make sure such pages are fully loaded then you can use waits and proceed further. | 1. **NAVIGATE** will just redirect to our required page and will not wait for the contents to load. |
| 1. Syntax: driverobj.get(<http://www.gmail.com)>; | 1. Syntax**:**   driverobj.navigate().to(<http://www.gmail.com>);  driverobj.navigate().refresh()  driverobj.navigate().back()  driverobj.navigate().forward() |

* Handling Untrusted SSL Certificates in Firefox browser .i.e.



* **Syntax for Firefox:***import org.openqa.selenium.firefox.FirefoxProfile;*

*import org.openqa.selenium.firefox.internal.ProfilesIni;*

ProfilesIni prof = new ProfilesIni();

FirefoxProfile ff = prof.getProfile("myProfile");//FireFoxProfile is class

ff.setAcceptUntrustedCertificates(true);

ff.setAssumeUntrustedCertificateIssuer(false);

WebDriver driver = new FirefoxDriver(ff);//Webdriver is interface & FireFoxDriver is class  
**Note:** to create **myProfile** - > refer this link http://www.technorms.com/182/how-to-create-a-firefox-profile-and-manage-multiple-profiles **Syntax for Chrome:***import org.openqa.selenium.remote.CapabilityType;*

*import org.openqa.selenium.remote.DesiredCapabilities;*DesiredCapabilities capabilityob = DesiredCapabilities.chrome();//DesiredCapabilities is class

capabilityob.setCapability(CapabilityType.ACCEPT\_SSL\_CERTS, true);  
System.setProperty("webdriver.chrome.driver", "E:/chromedriver.exe");  
Webdriverdriverobj = new ChromeDriver(capabilityob);//Webdriver is interface & FireFoxDriver is class  
**Syntax for InternetExplorer:**System.setProperty("webdriver.ie.driver", "D:/IEDriverServer.exe");  
driverobj = new InternetExplorerDriver();  
driverobj.get("https://cacert.org/");  
driverobj.get("javascript:document.getElementById('overridelink').click();");  
 (or)  
By DesiredCapability similar to chromebrowser .

* To pass Username and Password in the authentication popup of the site.

driverobj.get(“http://username:password@www.thesite.com”);

* To maximize the window size.  
  driver.manage().window().maximize();  
  **Time Outs**There are three timeout in selenium webdriver:  
  **implicitlyWait**(long time, TimeUnit unit);  
  **setScriptTimeout**(long time, TimeUnit unit);  
  **pageLoadTimeout**(long time, TimeUnit unit);  
  **Remember:** Timeout is an interface.  
  **pageLoadTimeout:** Selenium defines different timeouts and wait mechanisms. One of the timeouts is focused on the time a webpage needs to be loaded – the pageLoadTimeout limits the time that the script allots for a web page to be displayed. If the page loads within the time then the script continues. If the page does not load within the timeout the script will be stopped by a TimeoutException.  
  **eg:**  
  driver.manage().timeouts().pageLoadTimeout(40, TimeUnit.SECONDS);  
  try {  
  driver.get(myWebPage);  
  driver.findElementByXpath(verificationField);  
   } catch (TimeoutException e) {  
  System.out.println("Page: " + myWebPage + " did not load within 40 seconds!"); // treat the timeout as needed  
   }  
  **setScriptTimeout:** It sets the amount of time to wait for an asynchronous script to finish execution before throwing an error. If the timeout is negative, then the script will be allowed to run indefinitely. Use below code to implement setScriptTimeout. It is mainly used for JS Scripts.  
  driver.manage().timeouts().setScriptTimeout(100,TimeUnit.SECONDS);  
  JavascriptExecutor js = ((JavascriptExecutor) driver).executeAsyncScript(“Some asynchronous script”);  
  eg: js.ExecuteAsyncScript("setInterval(function(){ alert('Hello');},3000); callback();"); //this will give alert after 3 seconds.  
  **What is asynchronous script?(executeAsyncScript)**  
  This method doesn't block the execution of next line of code till execution of this method is completed. This method will execute as well as next line of code will be executed asynchronously. (Without blocking each other).  
  **What is synchronous script?(executeScript)**This method will block the execution till it's execution is completed and then it moves to next line of code. In short your automation code will halt till the Javascript is executed via this method.  
  **Note**: Since there is a chance that next line of code may execute soon, before its previous line in **executeAsyncScript,** we can use **setScriptTimeout** it will make to wait until the previous line of code is executed.  
  **Implicitly Wait:**An implicit wait is to tell WebDriver to poll the DOM for a certain amount of time when trying to find an element or elements if they are not immediately available. The default setting is 0. Once set, the implicit wait is set for the life of the WebDriver object instance.  
   **Syntax:**WebDriver driver = new FirefoxDriver();  
  driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);  
  driver.get("http://www.gg.com");  
  WebElementmt DynamicElement = driver.findElement(By.id("myDynamicElement"));  
  **Webdriver Waits**:  
  Normally implicitly wait is a type of timeouts. But in general when anyone asks webdriver waits we should mention 3 types:  
  **1. Implicitly wait**  
  **2. Explicit wait  
  3. Fluent wait**

**Explicit wait:**  
An explicit wait is code you define to wait for a certain condition to occur before proceeding further in the code. There are some conveniences methods provided that help us to write code that will wait only as long as required. WebDriverWait in combination with ExpectedCondition is one way this can be accomplished.  
**Syntax:**WebDriver driverob = new FirefoxDriver();

driver.get("http://somedomain/url\_that\_delays\_loading");  
WebDriverWait waitob = new WebDriverWait(driverob, 1000);

waitob.until(ExpectedConditions.presenceOfElementLocated(By.linkText("QAmail")));  
**Some common conditions we use in in explicit wait:**

1. alert\_is\_present
2. element\_located\_selection\_state\_to\_be
3. element\_located\_to\_be\_selected
4. element\_selection\_state\_to\_be
5. element\_to\_be\_clickable
6. element\_to\_be\_selected
7. frame\_to\_be\_available\_and\_switch\_to\_it
8. invisibility\_of\_element\_located
9. presence\_of\_all\_elements\_located
10. presence\_of\_element\_located
11. staleness\_of
12. text\_to\_be\_present\_in\_element
13. text\_to\_be\_present\_in\_element\_value
14. title\_contains
15. title\_is
16. visibility\_of
17. visibility\_of\_element\_located

**Fluent Wait:**

When we try to test the presence of an element that may appear after every x seconds/minutes.

**Syntax:**

Wait waitob = new FluentWait(driverob).withTimeout(30, TimeUnit.SECONDS).pollingEvery(5, TimeUnit.SECONDS).ignoring(NoSuchElementException.class);  
WebElement foo = wait.until(new Function() {

public WebElement apply(WebDriver driver) {

return driver.findElement(By.id("foo")); }  
Above code it waits until the element with id "foo" is found. If the element is not found, retry every 5 seconds. But wait only up to a maximum of 30 seconds. It does this by calling the following function every 5 seconds, until it doesn't return null:

|  |  |
| --- | --- |
| **Implicit Wait** | **Explicit Wait** |
| Implicit wait is set for the entire duration of the web Driver object. Suppose, you want to wait for a certain duration, let's say 5 seconds before each element or a lot of elements on the webpage load. Now, you wouldn't want to write the same code again and again. Hence, implicit wait. However, if you want to wait for only one element, use explicit. | You not only need web element to show up but also to be clickable or to satisfy certain other property of web elements. Such kind of flexibility can be provided by explicit wait only. Especially helpful if dynamic data is being loaded on webpage. You can wait for that element to be developed (not just show up on DOM) using explicit wait. |

* Locators available in the Selenium Webdriver:
* ID & Name
* CSS Selector
* X-path (Absolute and Relative)
* Tag Name
* Link Text and Partial Link Text

**Note**:   
1. Finding elements by ID is usually going to be the fastest option, because at its root, it eventually calls down to document.getElementById(), which is optimized by many browsers.

2. Finding elements by XPath is useful for finding elements using very complex selectors, and is the most flexible selection strategy, but it has the potential to be very slow, particularly in IE. In IE 6, 7, or 8, finding by XPath can be an order of magnitude slower than doing the same in Firefox. IE provides no native XPath-over-HTML solution, so the project must use a JavaScript XPath implementation, and the JavaScript engine in legacy versions of IE really is that much slower.

3. If you have a need to find an element using a complex selector, I usually recommend using CSS Selectors, if possible. It's not quite as flexible as XPath, but will cover many of the same cases, without exhibiting the extreme performance penalty on IE that XPath can.

**Syntax**

driver.findElement(By.id(“name”));

driver.findElement(By.name(“name”));  
driver.findElement(By.xpath(“//tagname[@attributename=’value’]”));

driver.findElement(By.xpath(“//tagname[@attributename=’value’][[@attributename1=’value1’]”));  
driver.findElement(By.xpath(“//’\*[@attributename=’value’]”));  
\* means it will check all tags.  
driver.findElement(By.xpath (“//\*[contains(text(),’sometexthere)]”)  
driver.findElement(By.xpath(“html/body/div[0]/”));

driver.findElement(By.linktext(“txt in between the <a></a>”));

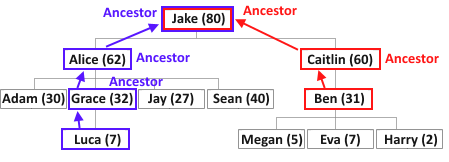
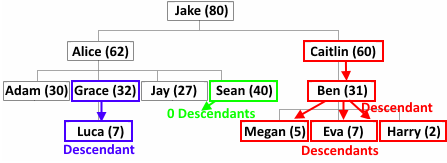
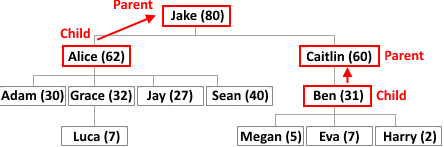
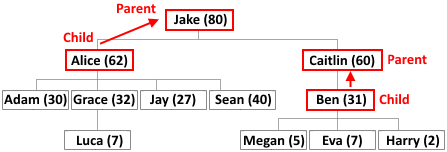
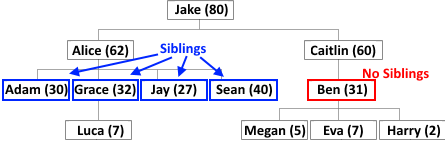
driver.findElement(By.partialLinkText(“txt in between the <a></a>”));

driver.findElement(By.cssselector(“.classname”));  
driver.findElement(By.cssselector(“#idname”));  
driver.findElement(By.cssSelector("tagname.classname"));

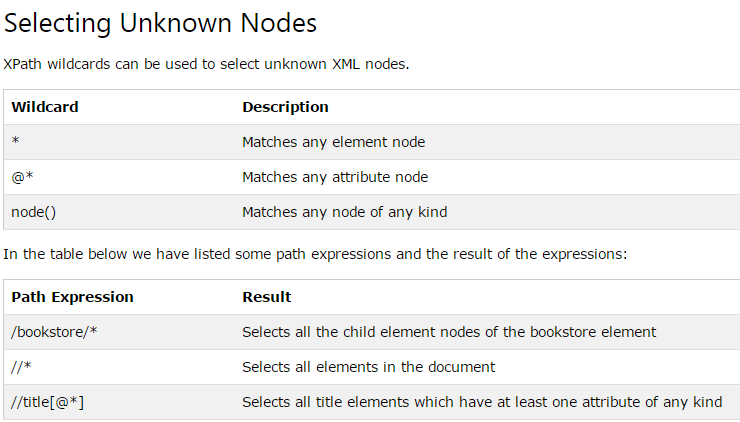
driver.findElement(By.xpath(“//\*[tagname=’text in between open and close tag’]”));

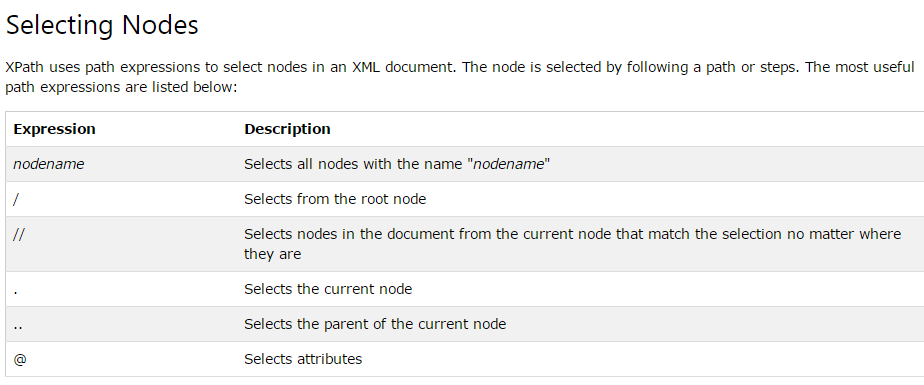
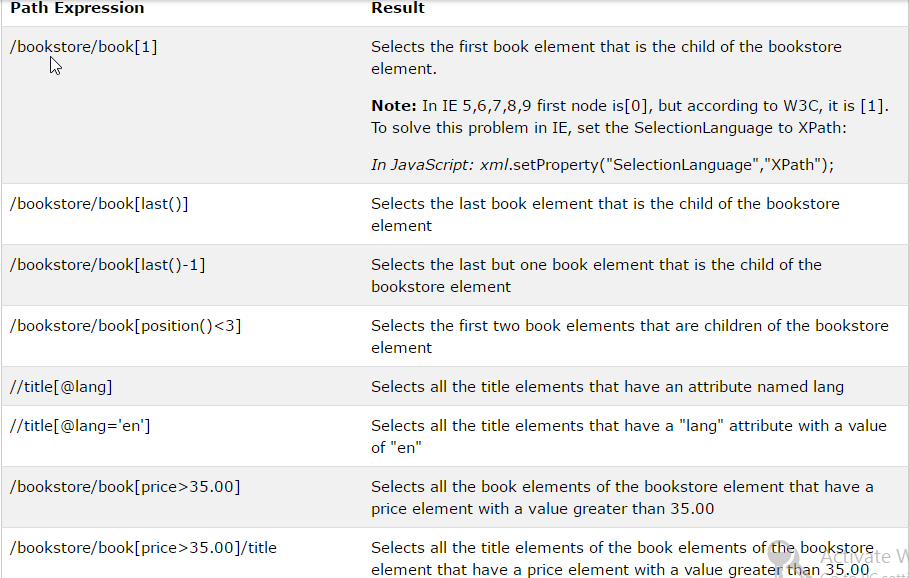
driver.findElement(By.cssSelector("tagname[atribute=value]"));  
driver.findElement(By.cssSelector("tagname[attribute^=’value’]"));// ^ is value start from that letter

driver.findElement(By.cssSelector(“tagname [attribute =’value’][attribute =’value’]”));

driver.findElement(By.tagName(“tagname”));  
**Xpath ancestor, Descedents, Sibling, Child**->An ancestor is any element which is connected further up in the hierarchy tree – no matter how many levels higher.  
-> An ancestor is a parent, grandparent, great-grandparent, and so on.->A descendant refers to any element that is connected lower down the hierarchy tree – no matter how many levels lower.  
-> A descendant is a child, grandchild, great-grandchild, and so on.  
  
-> AParent element which is directly above and connected to an element below in the hierarchy tree.  
-> Each element has **only one parent** – except for root (top most elements). It does not have parent as it does not have any element linked above it.  
  
->A child element that is directly below and connected to an element above in the hierarchy tree.  
->Element can have 0 or many child items.  
  
->A sibling is an element which shares the same parent with another element.  


**Xpath Predictions**



  
**Predicators**

**Eg: To choose parent node  
//\*[title="50"]/parent::\***

**//\*[title="50"]/..  
Return type** for FindElement is =>**Webelement**In order to get the page source of an web page using selenium: **Syntax:  
driver.getPageSource();**

* **Css VsXpath**: Make sure try to go with CSSselector, in case if there is no choice then go with x-path.

|  |  |
| --- | --- |
| CSS Selector | Xpath |
| CSS selectors are less brittle and fast in all browsers. | Xpath are more brittle and slow in browsers like IE and chrome. |
| Css Provides good balance in structure and attributes of the element. | Xpath fails to do that. It completely depends only on Structure. |
| CSS is lower HTTP requests, which means the website load faster. | Xpath requires more HTTP requests; hence it is slower loading of web pages. |

* To pass the texts or contents to the textbox or to click a button.

driver.findElement(By.id(“name”)).sendKeys(“hi”);

driver.findElement(By.id(“buttonid”)).click();

**Note:** Sometime **click()** operation will not work even though if we gave right path. It will raise an exception called i**llegaloperationecxeption.** To overcome this we can go with javascript executor.  
jsob.executeScript(“document.getElementByid(‘set’).click();”);  
jsob.executeScript(“document.getElementByid(‘set’).value=’asd’”;);

* To access the drop down box using selenium webdriver.

**Syntax:**

WebElement wb = driver.findElement(By.id(“dropdownbox”));

Select selobj = new Select (wb);

selobj.SelectByIndex(**0**);// It will choose the option based on the index value (starts from 0,1..)

selobj.SelectByText(“**Test**”);// It will choose text which we are giving. <option value=”as”>**Test** </option>

selobj.SelectByValue (“**as**”);// It will choose the value which is given inside the <option value=”**as**”>Test </option>

**Return Type** of SelectByIndex(**),**SelectByValue(),SelectByText() is **Boolean**.

* **Above code is for single select drop down box, if it is multiple then we have to manipulate the code.**

**Eg:**

WebElement wb = driver.findElement(By.id(“dropdownbox”));

Select selobj = new Select (wb);

If(selobj.isMultiple()){ //isMultiple() -> check whether it has multiple option select

selobj.SelectByIndex(**0**);  
selobj.SelectByText(“**Test”);**selobj.SelectByValue (“**as**”);  
}  
WebElement we = selobj.getFirstSelectedOption(); //selects the first optionselected  
System.out.println(we.getText());  
WebElement wb = selobj.getAllSelectedOption();//selects all the selected option

for(WebElement val:wb){

System.out.println o(wb.getText());   
}

* To deselect the value:

selbj.deselectByValue(“**as**”);

selbj.deselectByText(“**Test**”);

selbj.deselectByIndex(0);

selbj.deselectAll();

* To get the options from drop down:

**Syntax:**

WebElementwb = driver.findElement(By.id(“dropdownbox”));  
Select selobj = new Select (wb);

List<webElement>selOpt = wb.getOptions();

IntselOptsize = selOpt.size();

For(inti=0;i<selOpt;i++){

Syso(selOpt.get(i).getText());

}  
**Return Type** of getOptions(**)** is **String**.

* To map the drop down values in such a way that whether the actual and expected order are same.  
  **Syntax:**ArrayList<String>expectedDrpdownOrder = new ArrayList<String>();

expectedDrpdownOrder.add(“test1”);

expectedDrpdownOrder.add(“test2”);

WebElement wb = driver.findElement(By.id(“dropdownbox”));

List<WebElement>selOpt = wb.getOptions();

ArrayList<String>actualDropdownOrder = new ArrayList<String>();

for(WebElement value: selOpt){

actualDropdownOrder.add(value.getText());

}

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

if(!expectedDrpdownOrder.get(i).equals(actualDropdownOrder.get(i)));

System.out.println("Drop-down values are NOT in correct order");

}

* Web driver Action is a constructor mainly used to perform Keyboard and Mouse Operations.

**Syntax for Drag and Drop:**

Actions actobj = new Actions(driverobj);

WebElement toSource = driverobj.findElement(By.id(“sample”));

WebElement toDest = driverobj.findElement(By.id(“sample2”));

driverobj.dragAnddrop(oSource ,oDest).perform(); //build() not used bcoz single action

driverobj.clickAndHold(oSource).moveToelement(oDest).release().build().perform();//build() used action  
**Note**:   
Here 'build()' method is used to compile all the list of actions into a single step and ready to be performed.  
perform() A convenience method for performing the actions without calling build() first.  
release() Releases the depressed left mouse button at the current mouse location.  
**Remember:** We use build() when we are performing sequence of operations and no need to use if we are performing single action.  
**Syntax for Keyboard Operation:**

WebElement oTex = driverobj.findElement(By.id(“sample”));

Actions actobj = new Actions(driverobj);  
actobj.click(oTex).Keysup(Keys.SHIFT).sendKeys(“sdff”).build().perform();

actobj.click(oTex).sendKeys(Keys.TAB).build().perform();  
driver.findElement(By.id("Value")).sendKeys(Keys.ENTER);

**Syntax Mouse Right Click Operation:**

actobj.contextClick(driverobj.findElemeny(By.id(“test”)).sendKeys(Keys.ARROW\_DOWN). sendKeys(Keys.ARROW\_DOWN).sendKeys(Keys.RETURN).build().perform(); **Some Common Operations used in Actions**:

* contextClick(Webelement)
* doubleClick(Webelement)
* dragAnddrop(Webelement1,Webelement2);
* clickAndhold(Webelement1,Webelement2);
* moveToelement(Webelement)
* KeyDown/KeyUp(Keys.Keyname)
* Selenium Webdriver for retrieving values from list and tables

**findElements** method will return list of all the matching elements from current page as per given element locator mechanism. If not found any element on current page as per given element locator mechanism, it will return empty list.

**Syntax for Table values Retrieving:**

driver.findElement(By.tagName(“table”));

List <WebElement>oRow = driver.findElements(By.tagName(“tr”));

IntoRowsize = oRow.size();

for(inti=0;i<oRowsize;i++){  
List <WebElement>oCol = oRow .get(i).findElements(By.tagName(“td”));  
IntoColsize = oCol.size();

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

Syso(oCol.get(j).getText());

}

Syso(“--------”);

}  
(or) using foreach logic

int i\_RowNum=1;

for(WebElement rowElement:rowCollection)

{

List<WebElement> colCollection=rowElement.findElements(By.xpath("td"));

int i\_ColNum=1;

for(WebElement colElement:colCollection)

{

System.out.println("Row "+i\_RowNum+" Column "+i\_ColNum+" Data "+colElement.getText());

i\_ColNum=i\_ColNum+1;

} i\_RowNum=i\_RowNum+1;

}

driver.close();

Return Type of findElements is List <WebElement>  
**HashMap Collections for Table:**driverobj.get("file:///C:/Users/lenovo/Desktop/tabletest.html");

WebElement wb = driverobj.findElement(By.tagName("table"));

String key=null;

String val = null;

List<WebElement> tr = driverobj.findElements(By.tagName("tr"));

HashMap<String,String>hp = new HashMap<String,String>();

Iterator<WebElement>row = tr.iterator();

while(row.hasNext()){

WebElement orow = row.next();

List<WebElement> td = orow.findElements(By.tagName("td"));

Iterator<WebElement>cell = td.iterator();

WebElement cellval = cell.next();

key = cellval.getText();

while(cell.hasNext()){

WebElement colval = cell.next();

val = colval.getText();

}

If(key!=null&&val!=null){

hp.put(key, val);

key=null;

val=null;

}

}  
for(Map.Entry m:hp.entrySet()){

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

}

System.out.println(hp.entrySet());  
**Remember**: To check whether the particular key is present then we can use  
 hp.containskey(keyvalue);

* Excel Reader using Selenium Webdriver.

**Syntax:**

FileInputStream ofile = new FileInputStream("D:\\testing\\testing\_sw\_links.xlsx");

XSSFWorkbook wbobj = new XSSFWorkbook(ofile);

XSSFSheet sheetobj = wbobj.getSheetAt(0);

int oRow = sheetobj.getLastRowNum();

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

XSSFRow Rowval = sheetobj.getRow(i);

int ocol = Rowval.getLastCellNum();

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

System.out.println(Rowval.getCell(j).getStringCellValue());

}

System.out.println("===");

}

Wbobj.close();

**Note:** To read (.xls) we can use HSSFWorkbook and for (.xlsx) we can use XSSFWorkbook but note that to read any excel format just give Workbook alone.

**Excel Reader using Collection hashmap to store its data:  
Note: excel sheet will have two columns username and password, here if username is dinesh we are getting its respective password by storing it in hash map.**

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

FileInputStream oFile = new FileInputStream("D:\\testing\\Java\_practice\\test1.xlsx");

XSSFWorkbook wb = new XSSFWorkbook(oFile);

XSSFSheet sheet = wb.getSheet("Sheet1");

String key = null;

String value = null; // use a default value that will never occur in the sheet

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

Iterator<Row> rowIterator = sheet.iterator();

while(rowIterator.hasNext()) {

Row row = rowIterator.next();

Iterator<Cell> cellIterator = row.iterator();

Cell cell = cellIterator.next();

key = cell.getStringCellValue();

while(cellIterator.hasNext()){

Cell cells = cellIterator.next();

value = cells.getStringCellValue();

}

if(key != null && value != null)

{

map.put(key, value);

key = null;

value = null;

}

}

if(map.containsKey("Dinesh")){

System.out.println(map.get("Dinesh"));

}

}}  
**Remember**: To check whether the particular key is present then we can use  
 hp.containskey(keyvalue);

* To take screenshots in the Selenium web driver.

**Syntax:**

File srcFile = ((TakeScreenshot)driverobj).getScreenshotAs(OutputType.FILE)); //File class is constructor

FileUtils.copyFile(scrFile, new File(“c:\\tmp\\Screenshotfilename.png”)); //FileUtils also class  
**Note:** Here **TakeScreenshot** is interface and we are following type casting here which converts Webdriver interface’s reference (driverobj) to the reference srcFile of Takescreenshot to access the method getScreenshotAs().

If your cast fails, however, you'll get a *ClassCastExcepion* at run-time.

* Implementing JavaScript in Selenium Webdriver. In order to perform javascript in the selenium we have to use “JavascriptExecutor”

**Some Javascript Operations:**

JavascriptExecutor jsob = (JavascriptExecutor)driverobj;

jsob.executeScript(“Window.scrollBy(100,200);”);

jsob.executeScript(“alert(‘dad’);”);

Alert alertob = driverob.switchTo.alert(); //handling alertbox

alertob.accept();

//or

alertob.dismiss();  
**Some alert() functions:**

accept() To accept the alert

dismiss() To dismiss the alert

getText() To get the text of the alert

sendKeys() To write some text to the alert

**Note:** There are three types of alert popus in javascript:

1. Simple Alert.
2. Confirmation Alert (OK and Cancel button available)
3. Prompt Alert(Text Box and Ok,Cancel Button will be there) => alert.sendkeys(“text”) //to pass value in that text box

**Note:** sometimes click() or sendkeys() operation will not work in some browsers, at that time it can be handled using Javascript.

jsob.executeScript(“document.getElementByid(‘set’).click();”);  
js.executeScript("document.getElementById('test').value='test';");

**JavascriptExecutor** is an interface.

**JavascriptExecutor jsob = (JavascriptExecutor)driverobj;** this code nothing but type casting in java which means we are converting the (driverobj) reference of Webdriver to (jsob) reference of JavascriptExecutor because WebDriver interface does not contain the executeScript() so inorder to use that we are converting.  
**Page Factory**Page Factory Pattern is like an extension to Page Object Model , but Page Factory is much enhanced model. To start with, we just need to import package ‘org.openqa.selenium.support.PageFactory’. Factory class can be used to make using Page Objects simpler and easier.

We use Page Factory pattern to initialize web elements which are defined in Page Objects.We should initialize page objects using initElements() method from PageFactory Class as below, Once we call initElements() method, all elements will get initialized. PageFactory.initElements() static method takes the driver instance of the given class and the class type, and returns a Page Object with its fields fully initialized.

HomePage homePage = PageFactory.initElements(driver, HomePage.class);

In Page Factory, Annotations are used to give descriptive names for WebElements to improve code readability. And annotation @FindBy is used to identify Web Elements in the page.

**Note**: By default, PageFactory will search for elements on the page with a matching id attribute, If that fails, then it will search by the name attribute.

The @FindBy annotation supports all the other locators strategies that we use:

**id, name, className, css, xpath, tagName, linkText and partialLinkText**We can either use this annotation by specifying both "How" and "using" or by specifying any one of the location strategies

@FindBy(how = How.ID, using = "username")

private WebElement userName;

We can re-write the above one as below:

@FindBy(id="username")

private WebElement userName;

To work with class name, we will define as below:

@FindBy(className=".input.username")

private WebElement userName;

When we have multiple elements (list of WebElements), we can initialize them using PageFactory as below :

@FindBy(tagName = "mylist")

private List<WebElement> links;

@FindBy(how=How.XPATH,using="//div[contains(text(),'Preferences saved']");  
private WebElement userName;

**Eg:**import org.openqa.selenium.support.FindBy;

import org.openqa.selenium.support.How;

public class PAGEFACT {

@FindBy(how=How.ID,using="nameid")

WebElement userName;

@FindBy(how=How.PARTIAL\_LINK\_TEXT,using="Cancel")

WebElement link;

@FindBy(how=How.TAG\_NAME,using="h3")

WebElement title;

@FindBy(how=How.NAME, using="VehicleType")

WebElement dropdown;

public void name(String name){

userName.sendKeys(name);}

public void link(){

link.click();}

public String title(){

return title.getText();}}  
**Test.java**  
public class Test{

public static void main(String[] args){

driver = new FirefoxDriver();

driver.get("http://ntltaxi.com/");

PAGEFACT homePage = PageFactory.initElements(driver, PAGEFACT.class);//using homepage instance we are accessing the pagefactory elements and methods.

homePage.name("test");

homePage.link();

System.out.println(homePage.title());

Select sb = new Select(homePage.dropdown);

sb.selectByIndex(0);}}  
**Find Broken Image/Invalid Image:**There are cases where we have seen image loading is failed due to many reasons, the most see is "Image not loading - Failed to load the given URL” because of image file is not located in the same location as that is specified or may be image file is corrupted. And it will be very difficult to identify invalid images when there are many in the applications. To achieve this, we can use HTTPClient library to check status codes of the images on a page. If they don't load correctly, then it will be registered with likely a 404 but not a 200 status code. We can easily say whether the link is broken or not with status codes.  
First we will try to find all images on the page by using Webdriver. Below is the syntax:  
**Syntax:**

List<WebElement>imagesList = driver.findElements(By.tagName("img"));

Now iterate through each image and verify response code with HttpStatus and it should be 200 if not, increment invalid images count. We can get the response code using below statement:  
**Syntax:**

**response.getStatusLine().getStatusCode();  
eg:**@BeforeClass

public void setUp() {

driver = new FirefoxDriver();

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

}

@Test

public void validateInvalidImages() {

try {

int invalidImageCount = 0;

List<WebElement>imagesList = driver.findElements(By.tagName("img"));

System.out.println("Total no. of images are " + imagesList.size());

for(WebElementimgElement : imagesList) {

if(imgElement != null) {

verifyimageActive(imgElement);

}}

System.out.println("Total no. of invalid images are " + invalidImageCount);

} catch (Exception e) {

e.printStackTrace();

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

@AfterClass

public void tearDown(){

if(driver!= null)

driver.quit();

}

public void verifyimageActive(WebElementimgElement) {

try {

HttpClient client = HttpClientBuilder.create().build();

HttpGet request = new HttpGet(imgElement.getAttribute("src"));

HttpResponse response = client.execute(request);

// verifying response code he HttpStatus should be 200 if not,

// increment as invalid images count

if (response.getStatusLine().getStatusCode() != 200)

invalidImageCount++;

} catch (Exception e) {

e.printStackTrace();

}}}

* To pass a value in disabled text box.

**Syntax:**

**jsob.executeScript("document.getElementById('txt2').value='sanjay'");**

* File Uploading With Selenium Webdriver, it can be done in three ways:
* AutoIT
* Send Keys
* Robot API/Class
* Steps to upload file through Auto IT:

1. Primary thing download AutoIT exe file along with its editor SCTEA4.AutoITs.
2. With the help of AuTOIT finder tool get the id of the Browse button, and the id of the Open button in the Windows Popup.
3. In Editor 3 Keywords are used for uploading the file:

ControlFocus{“Title”,””,”BrowsebuttonID”};

ControlSetText{“Title”,””, ”BrowsebuttonID”,”filepath to upload”};

ControlClick{“Title”,””,”OpenbutonID”}

1. Save this file in ’.au3’ format.
2. After running in the ‘.au3’ file an ‘.exe’ file is generated which need to be included in the selenium code.
3. Webdriverdriverobj = new FireFoxFriver();

driverobj.get(“URL”);

driver.findElement(By.id(“browsebuttonID”)).click();

Runtime.getRuntime().exec(“path of the .exe file generated from ‘.au3’”);

* Steps to upload file through Send Keys:  
  WebElement element = driverobj.findElement(By.name("browsebuttonid"));  
  element.sendKeys("D:/file.txt");
* Steps to upload file through Robot Class:

**Watch Video for autoit and Robertclass**

* Selenium Event Listeners: Mainly used to log or monitor the webdriver actions or process.  
  **Syntax:**

FirefoxDriver driver = new FirefoxDriver();

EventFiringWebDriver eventDriver = new EventFiringWebDriver(driver);

EventHandler handler = new EventHandler(); //EventHandler is class.

eventDriver.register(handler);

eventDriver.get("http://toolsqa.com/automation-practice-switch-windows/");

WebElement element = eventDriver.findElement(By.id("target"));

element.click();  
**Note:** Here EventHandler is a class which implements the interface called WebDriverEventListener

Common methods which involves in the **WebDriverEventListener is: (another file EventHandler.java)**

* afterChangeValueOf()
* onException()
* beforeNavigateTo()
* beforeNavigateForward()
* beforeNavigateBack()
* beforeFindBy()
* beforeClickOn()

Note: Once the events() are done do **unregister** the class at last.

* Selenium Webdriver parallel testing can be done by invoking two browsers or drivers with different driver objects.

**Selenium webdriver to handle windows and frames and Multiple Tabs.**

**Syntax:**

**driverobj.switchTo().frame(frameId/Name/Index); //handle frames**

**Syntax:**

WebDriver obj = new FirefoxDriver();

obj.get("http://hdfcbank.com/");

obj.findElement(By.linkText("Credit Card")).click();

obj.switchTo().window(obj.getWindowHandles().toArray()[1].toString());**//Switch to Child Window**

obj.close(); // close the child window or browser currently focusing.

obj.switchTo().window(obj.getWindowHandle()); // focus on the parent window.  
(or)  
 FirefoxDriver driver = new FirefoxDriver();

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

Actions act = new Actions(driver);

WebElement link = driver.findElement(By.linkText("Gmail"));

act.contextClick(link).sendKeys(Keys.ARROW\_DOWN).sendKeys(Keys.ARROW\_DOWN).sendKeys(Keys.RETURN).build().perform();

String parentWindow = driver.getWindowHandle();

Set<String> s1 = driver.getWindowHandles();

Iterator<String> li = s1.iterator();

while(li.hasNext()){

String childwindow = li.next();

if(!parentWindow.equalsIgnoreCase(childwindow)){

driver.switchTo().window(childwindow);

System.out.println(driver.getTitle());

WebElement sublink = driver.findElement(By.linkText("Sign in"));

act.contextClick(sublink).sendKeys(Keys.ARROW\_DOWN).sendKeys(Keys.ARROW\_DOWN).sendKeys(Keys.RETURN).build().perform();

driver.switchTo().window(childwindow);

System.out.println(driver.getTitle());

driver.close();

}}

driver.switchTo().window(parentWindow);

System.out.println(driver.getTitle());

}  
**Note: Return type of getwindow handle() will be lengthy alpha numeric.**

**Try and See Doubt**

* To handle a frame whose id is dynamically changes the go with its xpath:

**Eg:** driver.switchTo().frame(driver.findElement(By.xpath("//iframe[contains(@src,'FUN\_UnitList\_FilterByLevelIndexOne')]"));

driver.switchTo().frame(index/name/id);

* Difference between webdriver.Dispose(), .Close() and .Quit()
* **webDriverobj.Close()** - Close the browser window that the driver has focus of
* **webDriverobj**.**Quit**() - It is used to shut down the web driver instance or destroy the web driver instance(Close all the windows). [internally it calls the dispose()]
* **webDriverobj**.Dispose() Closes all browser windows and safely ends the session
* To handle multiple tabs, switching on to next tab then coming back to first tab.  
  WebDriver driver = new FirefoxDriver();

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

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

WebElement link = driver.findElement(By.linkText("Gmail"));

Actions act = new Actions(driver);  
act.contextClick(link).sendKeys(Keys.ARROW\_DOWN).sendKeys(Keys.RETURN).build().perform()driver.findElement(By.cssSelector("body")).sendKeys(Keys.CONTROL +"\t"); \\switching the tab

(or)

driver.findElement(By.cssSelector("body")).sendKeys(Keys.CONTROL).sendKeys(Keys.SHIFT).sendKeys(Keys.TAB);

driver.findElement(By.cssSelector("body")).sendKeys(Keys.CONTROL +"w"); \\current focused tab will close  
**Test NG:**

* TestNg is a framework, which makes user to understand the code easily with the help of its annotations.
* It is also useful to generate the test reports in the form of html format.
* Common annotations used in the TestNg are as follows:  
  Annotations mentioned below are arranged in the format of its running order.
* @BeforeSuite
* @BeforeTest
* @BeforeClass
* @BeforeMethod
* @Test
* @AfterMethod
* @BeforeMethod
* @Test
* @AfterMethod
* @AfterClass
* @AfterTest
* @AfterSuite
* In addition to above Annotations there are some additional annotations in TestNg they are :  
  **@DataProvider**: Marks a method as supplying data for a test method. The annotated method must return an Object[][] where each Object[] can be assigned the parameter list of the test method. The @Test method that wants to receive data from this DataProvider needs to use a dataProvider name equals to the name of this annotation. If it's not supplied, the name of this data provider will automatically be set to the name of the method. We will send 3 rows and 2 columns ie. we will pass three different usernames and passwords.

public class DataProviderEmployeeExample {  
@Test(dataProvider="getData")

public void setData(String username, String password)

{

System.out.println("you have provided username as::"+username);

System.out.println("you have provided password as::"+password);

}

@DataProvider

public Object[][] getData()

{

//Rows - Number of times your test has to be repeated.

//Columns - Number of parameters in test data.

Object[][] data = new Object[3][2];

// 1st row

data[0][0] ="sampleuser1";

data[0][1] = "abcdef";

// 2nd row

data[1][0] ="testuser2";

data[1][1] = "zxcvb";

// 3rd row

data[2][0] ="guestuser3";

data[2][1] = "pass123";

return data;

}}

**@Factory:** allows tests to be created at runtime depending on certain data-sets or conditions. Sometimes we may need to run a set of tests with different data values. To achieve this we may define a separate set of tests inside a suite in the testng XML and test the required scenario. The problem with this approach is that, if you get an extra set of data, you will need to redefine the test. TestNG solves this problem by providing the @Factory annotation feature. Factory in TestNG defines and creates tests dynamically at runtime.  
public class SimpleTest

{

@Test

public void simpleTest(int i) {

System.out.println("Simple Test Method."+ i);

}}

public class SimpleTestFactory

{

@Factory

public Object[] factoryMethod() {

return new Object[] { new SimpleTest(0), new SimpleTest(1) };

}} **@Listeners**(PackageName.ListenerClassName)  
**@Parameters**

* Testng.xml file can be used to run the testng file. We can run directly the script file but there are few reasons we are going with (xml) file.

1. We can run multiple testng files in a single (xml) file.
2. Testng listeners can be run using xml file.
3. Testng parameters like groups, priority and all can be used using xml files.

* TestngGroups, Priority, alwaysRun

1. If we want to run particular set of test scripts alone out of number of test scripts in a testng file, we can go with groups.

**Syntax:**

@Test(groups={‘groupname’})

@Test(groups={“groupname1”,”groupname2”}) //multiple group name

1. If we want to run a test always at any cost no matters about the conditions. Then we can go with:  
   @Test(alwaysRun=true);
2. If there are multiple testscripts but if we want to run based on the priority which should run first, second so on.. we can set a parameter called “priority”

**Syntax:**

@Test(priority=0)  
4. TestNg to skip a test script, the we can use :

@Test(enabled=false);

* Testng Listeners is like Selenium webdriver listeners used to log or monitor the every actions.

**Steps:**Create a separate class which implements **ITestListener, ISuiteListener, IInvokedMethodListener.** And the methods which involved in the testing listeners are:

* onFinish(): Invoked after all the tests have run and all their Configuration methods have been called.
* onStart(): Invoked after the test class is instantiated and before any configuration method is called.
* afterInvocattion(): Invoke after each method
* beforeInvocation(): Invoke before each method
* onTestFailure(ITestResult result): Invoked each time a test fails.
* onTestSkipped(ITestResult result): Invoked each time a test is skipped
* onTestStart(ITestResult result): Invoked each time before a test will be invoked.
* onTestSuccess(ITestResult result): Invoked each time a test succeeds.

Note: This tesnglistener file is run with the the help of the (xml) file by including the <Listener></Listener> on it.  
Testng (xml) file:  
**Syntax:**

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

<suite name="Suite1" >

<listeners>

<listener class-name=" Packagename.testngllistenerclassfile"></listener>

</listeners>

<test name="test1">

<groups>

<run>

<include name=”groupname” />

<exclude name=”groupname”/>

</run>

</groups>

<classes>

<class name=" Packagename.testngclassfile2" />

</classes>

<classes>

<class name="Packagename.testngclassfile1" />

</classes>

</test>

</suite>

In TestNg suppose if a TestScript1 which is dependent on TestScript2 then you may wonder which test script will run first. Obviously TestScript2 only will run first after that testscript1. For this inTestNg there is parameter called **(dependsOnMethods={“methodname”}).**

Eg:  
@Test

public void testPrintMessage() {

System.out.println("Inside testPrintMessage()");

message = "Manisha";

Assert.assertEquals(message, messageUtil.printMessage());

}

@Test(dependsOnMethods = { "initEnvironmentTest" })

public void testSalutationMessage() {

System.out.println("Inside testSalutationMessage()");

message = "Hi!" + "Manisha";

Assert.assertEquals(message, messageUtil.salutationMessage());

}

@Test

public void initEnvironmentTest() {

System.out.println("This is initEnvironmentTest");

}}

In above example testscript having method initEnvironment() will run first followed by testSalutionMethod().

**Note: dependsONGroups={Group Name}** is similar to the above example but It deals with groups.

* TestNG provides an ability to run test methods, test classes and tests in parallel. By using parallel execution, we can reduce the 'execution time' as tests are started and executed simultaneously in different threads.IntestNG we can achieve parallel execution by two ways. One with testng.xml file and we can Configure an independent test method to run in multiple threads.

**Syntax:**

<suite name="Parallel test suite" parallel="methods (or) classes (or)instances(or)tests" thread-count="2">

<test name="Regression 1">

<classes>

<class name="com.parallel.TestParallelOne"/>

<class name="com.parallel.TestParallelTwo"/>

</classes>

</test>

</suite>

* **parallel="methods"(multiple @test):** TestNG will run all your test methods in separate threads. Dependent methods will also run in separate threads but they will respect the order that you specified.**Doubt**
* **parallel="tests" (in xml= multiple< test> tags should have):** TestNG will run all the methods in the same <test> tag in the same thread, but each <test> tag will be in a separate thread. This allows you to group all your classes that are not thread safe in the same <test> and guarantee they will all run in the same thread while taking advantage of TestNG using as many threads as possible to run your tests. Eg: Crossbrowser testing
* **parallel="classes":(in xml= multiple< class name> tags should have)**TestNG will run all the methods in the same class in the same thread, but each class will be run in a separate thread.
* **parallel="instances":** TestNG will run all the methods in the same instance in the same thread, but two methods on two different instances will be running in different threads.**Doubt**

**Note:** The assigning of the thread is take care by the processor. So we can't say which thread is going to execute which method. But the concept is if there is 10 test cases and 2 threads each thread will take one each and which one is completed soon it will take another test script.

* TestNg Reports or list of files generated after (xml) file is executed as follows:
* Xmlfilename-results.xml
* Xmlfilename-failed.xml
* Emailable-report.html
* Xmlfilename-reports.js
* Index.html (screen shot of soft assertion)
* For testing the condition i.e the test script is passed or failed, we use **Assertion** command instead of **IF conditions** in TESTNg.

Assertion Commands are classified into two:

* Hard Assertion.
* Soft Assertion
* **Hard Assertions:** Here when the test scripts are failed, it will not proceed further test scripts. These assertions can be used for the Show Stopper Scenarios. A test is considered successful ONLY if it is completed without throwing any exception.

**Syntax or Example:**

@Test  
public void testCaseVerifyHomePage() {  
driver= new FirefoxDriver();  
driver.navigate().to("http://google.com");  
Assert.assertEquals("Gooogle", driver.getTitle(), "Strings are not matching");  
**//Write a code to login and write a method called isUserLoggedInSuccessfully and isUserLoggedOut which returns boolean.**   
Assert.assertTrue(isUserLoggedInSuccessfully(), "User failed to login"); Assert.assertFalse(isUserLoggedOut());  
}}  
Some Commonly used Hard Assesrtion Commands:

* **assertEqual(String actual,String expected)** :- It takes two string arguments and checks whether both are equal, if not it will fail the test.
* **assertEqual(String actual,String expected, String message)** :- It takes three string arguments and checks whether both are equal, if not it will fail the test and throws the message which we provide.
* **assertEquals(booleanactual,boolean expected)** :- It takes two Boolean arguments and checks whether both are equal, if not it will fail the test.
* **assertEquals(java.util.Collection actual, java.util.Collection expected, java.lang.String message)** :- Takes two collection objects and verifies both collections contain the same elements and with the same order. if not it will fail the test with the given message.
* **Assert.assertTrue(condition)** :- It takes one Boolean arguments and checks that a condition is true, If it isn't, an AssertionError is thrown.
* **Assert.assertTrue(condition, message)** :- It takes one Boolean argument and String message. It asserts that a condition is true. If it isn't, an Assertion Error, with the given message, is thrown.
* **Assert.assertFalse(condition)** :- It takes one Boolean arguments and checks that a condition is false, If it isn't, an Assertion Error is thrown.  
  **eg:**MyAssertFalseTest asft = new MyAssertFalseTest();

assertFalse(asft.isEvenNumber(3));

* **Assert.assertFalse(condition, message)** :- It takes one Boolean argument and String message. It asserts that a condition is false. If it isn't, an Assertion Error, with the given message, is thrown.

**Note**: assert statements can be used in try…catch blocks with no "special conditions".

If an assertion fails, it throws anjava.lang.AssertionError (note this is subclass of java.lang.Error rather than java.lang.Exception). Obviously under normal circumstances you should not catch assertion errors because they indicate bugs in your code.

* **Soft Assertion:** To use testng soft assertion, you have to use **testngSoftAssert** class. This class will helps not to throw an exception on assertion failure and recording failure. If we use soft assertion then your software web application's test execution will remain continue even If any assertion fails.

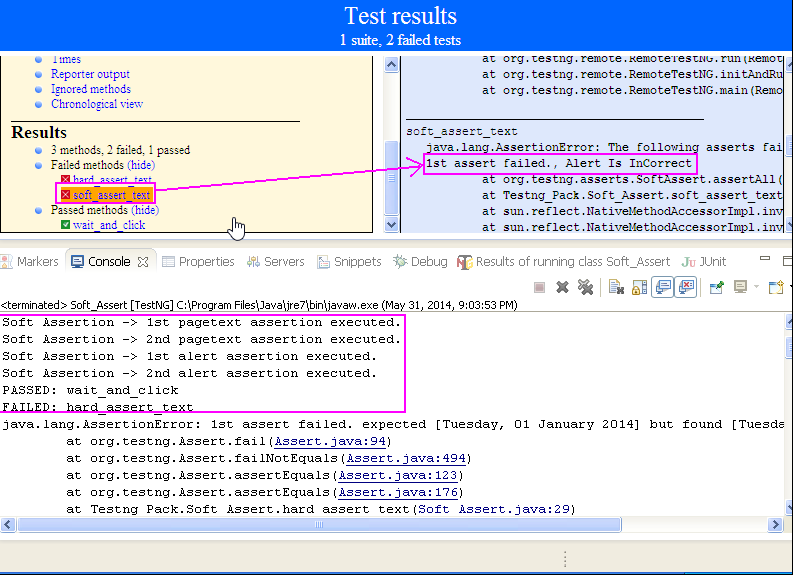
**Syntax:**

SoftAssert sassert = new SoftAssert();

sassert.assertEquals(“Actualtext”, "Tuesday,", "Alert is Incorrect");

System.out.println("Soft Assertion -> 1st pagetext assertion executed.");

**Note:The third parameter i.e.message in the above example will be shown in the TestNg Reports.**



**Cross Browser Testing in TestNg:**

In Testng we need to do cross browser testing via testing.xml that too in parallel, this is how we have to do:  
<suite name="Parallel test suite">

<test name="ChromeTest">

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

<classes>

<class name="com.parallel.TestParallelOne"/>

</classes>

</test>

<test name="FirefoxTest">

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

<classes>

<class name="com.parallel.TestParallelOne"/>

</classes>

</test>

</suite>

In Testng class file.  
@Test

@Parameters(“browser”)

Public void setup(String browsername){

If(browser.equalsIgnoreCase(“Chrome”){

System.setproperty(“webdriver.chrome.driver”,”C://Chrome.exe”)’;

Webdriver driver = new ChromeDriver();

}elseif(If(browser.equalsIgnoreCase(“Firefox”)){

Webdriver driver = new Firefoxdriver();

}}

Driver.get(“http://www.google.com”);

Here because the testing.xml has two Test tags ('ChromeTest','FirefoxTest'),this test case will execute two times for 2 different browsers.  
**Note**: Above code execute test cases sequentially, but in order to do it parallel browser testing below changes to be done in xml.  
<suite name="Parallel test suite"**parallel=”tests”**>

<test name="ChromeTest">

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

<classes>

<class name="com.parallel.TestParallelOne"/>

</classes>

</test>

<test name="FirefoxTest">

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

<classes>

<class name="com.parallel.TestParallelOne"/>

</classes>

</test>

</suite>  
**Doubt: if we didn’t specify thread-count in parallel execution then what will be default value for thread-count.**

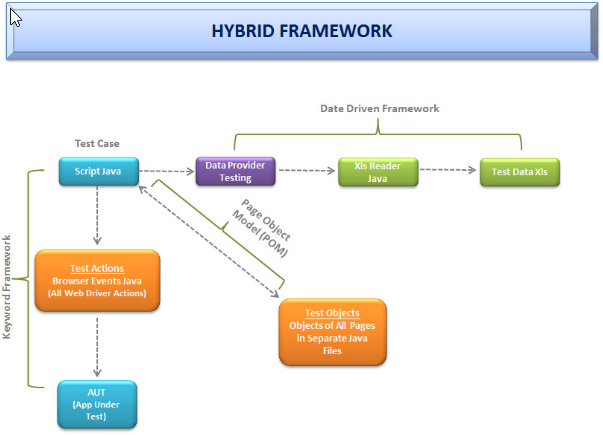
**How to Run the Failed Test cases alone:**-> Sometimes out of 100 test cases one or two will get failed, so we have to check and should run again only the failed test cases.  
-> When we run the testing automatically it will create the folder consists of the suite name which we are giving in xml, inside there will be reports in the form of xml as well as html. Then failed-testcases .xml file.  
->In failed test cases xml, it will have only the failed test cases, so next time we can run only this xml file to run only the failed test cases. **->**We can do it in programmatically too for that let create a class in a new package.  
1. Create a class name TestRunner.  
2. in that follow the below code:  
public class TestRunner{

public static void main (String[] args){

TestNG runner = new TestNG(); //Testng class import from org.testng

List<String> li = new ArrayList<String>();

li.add(“path of failedtestng xml file”);  
 runner.setTestSuites(li);

runner.run();}}  
**Hybrid Frame Work:  
  
Integration with Jenkins:**

* Jenkins is a job scheduler. You can use jenkins to configure your tests to run on a schedule (eg: every hour, every day at noon, etc), or every time you check in code. Jenkins also has features to allow you to run jobs on multiple machines, set dependencies between jobs, and many other features.
* We can also test reports or console output via mail to the respective person’s mail ids.

Steps to integrate Jenkins:

1. Download and install the Jenkins. The url or the port used by Jenkins by default is 8080.
2. Open the browser and give URL “**localhost:8080**”.
3. Click on the “Manage Jenkins”.
4. After that click on to the Configuration Jenkin. Inside that we can find JDK Installation button. Click on that and give our java JDK path in Java\_Home text box. And in name textbox give JAVA\_HOME and save it.
5. Now create folder called “lib” file in our project workspace path. Keep all the JAR files related to selenium into that lib folder of the workspace.
6. Now we have to create a batch file to run out Testng (xml) file. Save the file in notepad with the file name and extension “**run.bat**”

Commands to be placed in the notepad file:

**Java -cpbin;lib/\* org.testng.TestNG testing.xml**

1. Now click on the **New item** in Jenkins UI, Give some item name and click on the free style project and click ok.
2. It will take you to configure page, there click **Advanced Project Options -> Advanced** button.
3. Then check on the **Use custom workspace** and give our workspace path in Directory textbox.
4. Then click on Add Build Setup button and choose “Execute windows batch command”. And give the “run.bat” in Command text box and click apply.
5. Now click on “Build Now”. Build will create then clickthe build as console run. Our testing.xml file is executed through the Jenkins.

**Scheduling Jobs and Sending Mails in Jenkins:**

**Steps:**

1. Follow the above steps to create new build.
2. Click **Manage Jenkins**and click **Configuration Jenkins.**
3. Search for Email Notifications. In that give SMTP Port number, Email Id and Pwd to the respective fields. Please note that Jenkins will send mail only it the build or test cases is failed or stopped in between.
4. Click on **Configuration** Search for **Build Triggers** under that give **BuildPeriodically**.

Inside Build Periodically we have to give Schedule time inside the Schedule Text Box with the help of CRON Command.  
# ┌───────────── min (0 - 59)

# │ ┌────────────── hour (0 - 23)

# │ │ ┌─────────────── day of month (1 - 31)

# │ │ │ ┌──────────────── month (1 - 12)

# │ │ │ │ ┌───────────────── day of week (0 - 6) (0 to 6 are Sunday to

# │ │ │ │ │ Saturday, or use names; 7 is also Sunday)

# │ │ │ │ │

# │ │ │ │ │

# \* \* \* \* \* command to execute

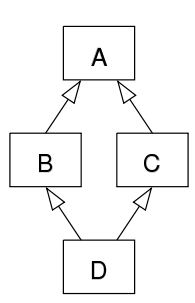
1. According to the time we set the build will automatically create and run it. If the build is failed we will receive mail if not we won’t receive it.

**SOAP UI:**Web service is something a file which will gets the raw data from the database according to the request raised from the client side. Getting the request from client and taking the response from the server and giving it to the client. Web service can be in XML or REST (Json format). In order to test the web service we are using a tool called SOAP UI. SOAP UI is mainly used to test the XML web service file but we can check for REST Web service too. Steps to test the XML file in SOAP UI:

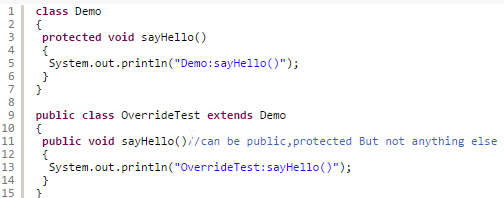
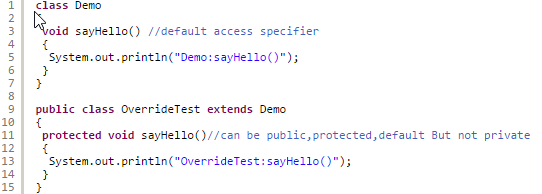
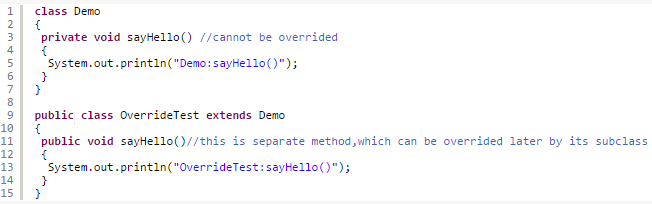
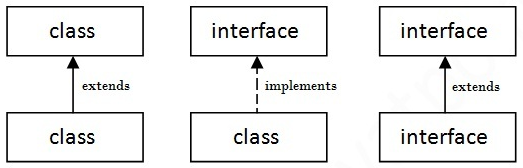
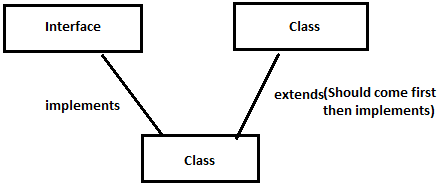
1. Download SOAP UI from its official site (soap UI.org) make sure we can download only trial version in order to download SOAP pro we have to pay for the license.
2. Click on to the Soap icon, then click on creates a New SOAP Project. Give project name as well as include the path of the WSDL File and click ok.
3. Once it is done it will import the Request and on the left corner under your new project title we can notice the request name. Double click on that.
4. Now you can see Screen will be split in two, left side it is for Request and right side you will get response for the given request. Both will be in XML format. This is how you will check the data.
5. Right Click on our Project and click New test suite and name it.
6. Right click on the Test Suite and create new test case. Right Click on that choose Soap Request and give the name.
7. Now to add assertion click on the ‘+’ symbol bottom. Now add assertion, click on the Property content in that popup and choose contains and give contents.
8. So if the response matches with the contents which has given it shows our test is passed or else it is failed. Noe we can add as many assertions which we want. Green and Red colour indicates the pass or fail of the test cases near to its name on the left hand menu.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Modifier** | **within class** | **within package** | **outside package by subclass only** | **outside package** |
| **Private** | Y | N | N | N |
| **Default** | Y | Y | N | N |
| **Protected** | Y | Y | Y | N |
| **Public** | Y | Y | Y | Y |

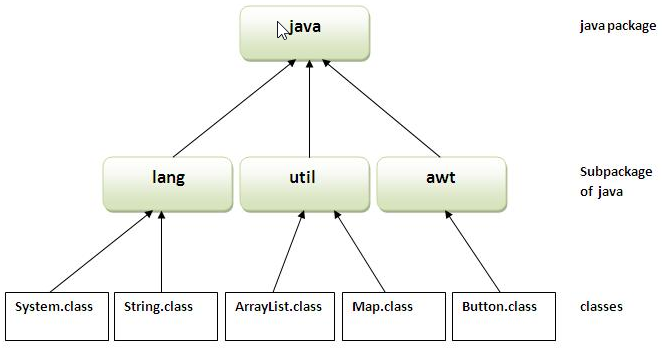
**Java Concepts:  
Access Modifiers:**

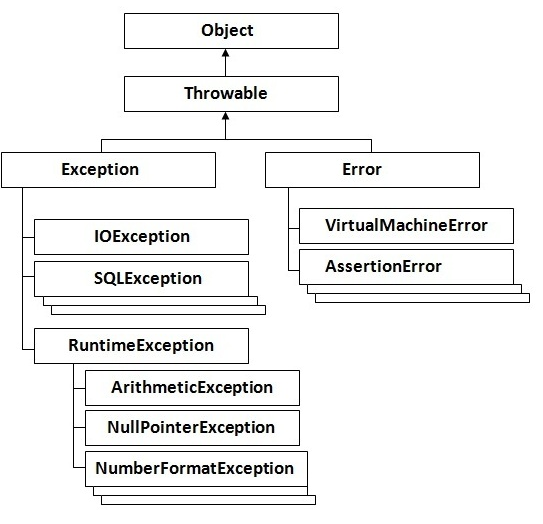
**Note**: When we declare a class or method and not mentioning any Access Modifiers then that method or class is Default Access Modifier.  
**Constructor**Constructor in java is a special type of method that is used to initialize the object. Java constructor is invoked at the time of object creation.  
**Rules for creating java constructor:**-> Constructor name must be same as its class name.  
-> Constructor must have no explicit return type.  
**There are two types of constructors:**-> Default constructor (no-arg constructor)  
-> User Define/Parameterized constructor  
If there is no constructor in a class, compiler automatically creates a default constructor. Default constructor provides the default values to the object like 0, null etc. depending on the type.  
class Student3{   
int id;   
String name;   
void display(){System.out.println(id+" "+name);}  
public static void main(String args[]){   
Student3 s1=new Student3();   
s1.display();}}  
**O/P:**0 null  
**Note**: In the above class, we are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.  
User define constructor are nothing but the creating the constructor by us which can have parameters or any particular thing to be performed by the constructor. When we are creating our own constructor at that time default constructor will not be there it will overridden.  
  
eg:  
class Student4{   
int id;  
String name;  
Student4(int i,String n){   
 id = i;   
 name = n;   
}   
void display(){  
System.out.println(id+" "+name);  
}  
public static void main(String args[]){   
 Student4 s1 = new Student4(111,"Karan");   
 s1.display();   
}}  
o/p:  
111 Karan  
**Constructor Overloading**  
Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.  
**Remember:**If we can pass arguments when creating an object then we can call it as constructor.  
**Why We Go For Constructor?**A constructor is basically a method that you can use to ensure that objects of your class are born valid. This is the main motivation for a constructor. To do other things that needs to be done when a new object is created.  
**Remember:  
1.**Constructor cannot be inherited in subclass but it can be invoked in the subclass using super().**(Doubt)**2. We can’t declare a Constructor as Static as well as static variables because constructor is invoked when object is created. But static something related to class level. So obviously it will throw error.  
3. Parent Constructor without parameter but child class has(or)equal parameter with same data type like child constructor has. Then when creating the object for the child class compiler will implicitsuper() and will call parent constructor too. If the arguments are not equal (or) child constructor doesn’t have parameter but parent constructor has then we have to create separate objects for parent and child constructor.  
4. We can’t declare a constructor as final but we can use final when declaring variables inside the constructor.  
eg:  
public class HelloWorld{  
HelloWorld() // or it can be HelloWorld(int b){  
System.out.println("Dinesh");  
 }}  
public class Test extends HelloWorld{  
Test(){  
 System.out.println("kavi");  
 }  
public static void main(String[] args){  
 Test myObject = new Test();  
 }}  
**o/p:**  
Dinesh  
kavi  
**this()**  
**this**is a reference variable that refers to the current object.  
**Usage:**  
**->this keyword can be used to refer current class instance variable.**  
public class HelloWorld{  
int b;  
public void dinesh(int b){  
b = b;**//if we put this.b**}  
void display(){  
System.out.println(b);**//will print 10**  
}  
public static void main(String args[]){  
HelloWorld e2 = new HelloWorld();  
e2.dinesh(10);  
e2.display();  
}}  
o/p: o/p  
0 10   
In the above example, parameter (formal arguments) and instance variables are same that is why we are using this keyword to distinguish between local variable and instance variable.  
**Note: What is Local and Instance variable?**An instance variable is a variable defined in a class (i.e. a member variable), for which an instance/object is used to access the variable. An instance variable is similar to a class variable. Instance variables are given default values, i.e. null if it's a String, 0 if it's an int.  
Local variables don't get default values, and therefore need to be explicitly initialized (and the compiler usually complains if you fail to do this).  
**eg:**  
class Student11{  
String m;  
Static n;  
public static void main(String args[]){  
Student11 obj = new Student11();  
System.out.print(obj.m);//instance variable gives null as output.  
System.out.print(m); //local variable which throws an error since it is not initialized.  
System.out.print(Student11.n);//Class variable.  
}}  
**->this() can be used to invoke current class constructor (Constructor Chaining).**  
class Student13{  
int id;  
Student13(){  
 System.out.println("default constructor is invoked");  
 }  
 Student13(int id){  
this();//**this()** is not used then we have to create another instance to call the first constructor.  
System.out.println(id);  
}  
public static void main(String args[]){  
Student13 e2 = new Student13(222);  
}}  
**O/P:**  
default constructor is invoked  
222  
**Note:** Why we are calling it as constructor chaining, because we can reuse the previous constructor or if we want to perform any operation of previous constructor when calling a constructor.  
**Remember:**It is must that **this()** should be the first statement in our constructor or error. messagethrown:**Exception in thread "main" java.lang.Error: Unresolved compilation problem**.  
->**The *this* keyword can be used to invoke current class method (implicitly).**class S{   
void m(){   
 System.out.println("method is invoked");   
}  
void n(){   
this.m();//no need because compiler does it for you.   
m(); //this will do, compiler automatically will create **this** keyword here.  
 }  
public static void main(String args[]){   
S s1 = new S();   
s1.n(); }}  
**->The this keyword can be used to return current class instance.(Doubt)**return\_type method\_name(){   
return this;   
}  
**super()**  
The super keyword in java is a reference variable that is used to refer immediate parent class object.  
**super.variablename** - is used to refer immediate parent class instance variable.  
**super () -**  is used to invoke immediate parent class constructor.  
**super.methodname** - is used to invoke immediate parent class method.  
Parent Constructor without parameter but child class has(or) equal parameter with same data type like child constructor has. Then when creating the object for the child class compiler will implicitsuper() and will call parent constructor too. If the arguments are not equal (or) child constructor doesn’t have parameter but parent constructor has then we have to create separate objects for parent and child constructor.  
**This vs Super:**super is used to access methods of the base classwhile **this** is used to access methods of the current class.Extending the notion, if you write **super(),** it refers to constructor of the base class**, and** if you write **this(),** it refers to the constructor of the very class where you are writing this code**. In other words:**->The this form chains to a constructor in the current class; i.e. in the A class.  
->The super form chains to a constructor in the immediate superclass; i.e. in the B class.  
**Why this() and super() can’t be used together in Constructor?**  
this() will call another constructor in the same class whereas super() will call a super constructor. If there is no super() in a constructor the compiler will add one implicitly. So this() can’t be used in parent class and child class constructor concepts.  
See we cannot use both in same, as we seen in constructor chaining this () refers to the previous constructor, as well as super () also refers to the parent class constructor. In other words both are trying to access same constructor, so obviously error will throw.  
  
eg:  
public class superParent{  
public superParent(){  
System.out.println("In Parent constructor");  
}}  
public class superThis extends superParent {  
int a;  
public superThis(){  
super();  
this(); //but if we use this.a=5; it will access no error because it is not a constructor call.  
System.out.println("In Child constructor, a="+a);  
}public static void main(String[] args) {  
superThis s = new superThis();  
}} **Encapsulation**  
Encapsulation in java is a process of wrapping code and data together into a single unit, for example capsule i.e. mixed of several medicines.  
**Static**:  
->The static keyword in java is used for memory management mainly.Static keyword is important in java, we will be using this keyword in declaring methods and declaring variables or blocks (i.e. static blocks). We call it as static methods and static variables. We call a class as static class when it has Collection of static methods or static variables. When we are declaring a Static method or static variable we can access directly by the class name, rather than creating an instance or an object. Static we mainly used of memory management, when we create object it will take lots of memory.  
**Note: Static Method cannot be override(Runtime polymorphism), 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) which means it support overloading (compile time polymorphism).  
eg:**class Student8{   
int rollno;  
String name;  
static String college ="ITS";  
Student8(int r,String n){  
 rollno = r;  
 name = n;  
}  
void display (){System.out.println(rollno+" "+name+" "+college);}  
public static void main(String args[]){   
Student8 s1 = new Student8(111,"Karan");   
Student8 s2 = new Student8(222,"Aryan");   
 s1.display();   
 s2.display();  
 }}  
O/P:  
111 Karan ITS  
222 Aryan ITS  
**Note: Why Java main method is public static void?**because object is not required to call static method if it were non-static method, jvm create object first then call main() method that will lead the problem of extra memory allocation.Since it is **static** it can be directly invoked via the class. Similarly, we use **static**sometime for user defined **methods** so that we need not to make objects. **Void** indicates that the **main**() **method** being declared does not return a value.  
->Static block is something we can execute the program without the main() method in java. And also if there is a Static block and main method then first Static block executed followed by the main method.  
-> We can have multiple Main Methods.  
**Syntax:**Static{  
System.out.print(“test”);  
}  
**Remember:**1. The static method cannot use non static data member or call non-static method directly.  
2. **this** and **super** cannot be used in static context why?  
The keyword “this” refers to an instance that’s been passed into an instance’s own method. It is inherently self-referential, and so requires an instance. The issue with “super” is similar. It requires an instance as a reference point. Static methods exist from the moment the program starts. They don’t require an instance.  
3. Static method can have return value.  
**Inheritance in Java**:  
Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.  
Java supports inheritance concepts it will support:  
1. Single Inheritance.  
2. Multilevel Inheritance  
3. Hierarchal Inheritance  
But Multiple inheritance can achieved only through interface not by classes.  
**Why use inheritance in java?**  
-> For Method Overriding (so runtime polymorphism can be achieved).  
-> For Code Reusability.  
**But why multiple inheritance not supported in java?**  
Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. If A and B classes have same method and you call it from child class object, there will be ambiguity to call method of A or B class it gives you compile time error. One of the most common scenarios is Diamond problem.  
  
**eg:**  
class Employee{  
 float salary=40000;  
}class Programmer extends Employee{  
Int bonus=10000;  
 public static void main(String args[]){  
 Programmer p=new Programmer();   
 System.out.println("Programmer salary is:"+p.salary);  
 System.out.println("Bonus of Programmer is:"+p.bonus);  
}}  
O/P:  
Programmer salary is:40000.0  
Bonus of programmer is:10000  
**Overloading:**Overloading is nothing but Compile time polymorphism. In a class two methods will have the same name but different parameter or same parameter with different data type. Then we can achieve the overloading. Let’s take the pseudo code:  
public class Test{  
public void Name(int b){}  
public void Name(String b){}  
(or)  
public void Name(int b, String b){}  
public void Name(String c){}  
}  
(or)  
public void Name(int b, String b){}  
public void Name(String c,long b){}  
}  
(But)  
public void Name(int a, int b){}  
public void Name(long a, long b){}  
} //throws complie error since long is the promotion of datatype int.  
**Remember:**  
->Method Overloading is not possible by changing the return type of the method. Because it creates ambiguity which method it is to be called.  
-> We can overload main() method. Obviously it means we can have multiple main methods in java. But remember entry point will be always **public static void main(String[] args)**.  
-> Static Methods can be overloaded with the different parameters.  
-> We cannot overload the overridden method.  
eg:  
class Overloading1{   
 public static void main(int a){   
 System.out.println(a);   
 } public static void main(String args[]){   
 System.out.println("main() method invoked");   
 main(10);   
 } }  
o/p: 10  
**Why we go for Overloading?**  
Suppose if there is two methods which performs addition operations, but one method will perform when we pass two parameters and another with three parameters, it is difficult to place a different naming convention for each method. So we can go with overloading.  
**Overriding**:  
If child class having the same method name and parameters which is already there in the parent class, then it called as overriding. Overriding is to achieve the run time polymorphism.  
**eg:Pseudo code.**public class Test{  
void Dinu(){}  
}public class Hi extends Test{  
void Dinu(){}  
}  
**Note:**  
->Static method cannot be overridden, because static method is bound with class whereas instance method is bound with object. Static belongs to class area and instance belongs to heap area.  
-> We can’t override the main() method because it is a static method.  
**Can we override two methods with different return type?Doubt**Yes it may differ but with some limitations.Before Java 5.0, when you override a method, both parameters and return type must match exactly. In Java 5.0, it introduces a new facility called covariant return type. You can override a method with the same signature but returns a subclass of the object returned. In another words, a method in a subclass can return an object whose type is a subclass of the type returned by the method with the same signature in the superclass.

|  |  |
| --- | --- |
| **Method Overloading** | **Method Overriding** |
| Method overloading is used *to increase the readability* of the program. | Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class. |
| Method overloading is performed *within class*. | Method overriding occurs *in two classes* that have IS-A (inheritance) relationship. |
| In case of method overloading, *parameter must be different*. | In case of method overriding, *parameter must be same*. |
| Method overloading is the example of *compile time polymorphism*. | Method overriding is the example of *run time polymorphism*. |
| In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter. | *Return type must be same or covariant* in method overriding. |

**Rules of Access Modifier in OverRiding:**We have to keep in mind hierarchy level of java always starts from lower to higher so the access modifier mentioned below is lower to higher level.  
1. Public  
2. Protected  
3.No-access or default  
4. Private  
->If super class method is public,while overriding it in child class,it should be public, Since it is the weakest access modifier or least restrictive access modifier.  
-> If super class method is protected, while overriding it in child class,it can be public or protected but not anything else.  
  
-> If superclass method is default, then while overriding it in child class,it can be public,protected or no access,but cannot be private.  
  
-> If superclass method is private,then while overriding it in child class,it can be anything. Since private method cannot be override.  
  
**Note:** keep in mind whenever we need to override a class child class should be obviously should have equal or weakest access modifier.  
**Why we go for overriding?**  
Method overriding, in object oriented programming, is a language feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its superclasses or parent classes.  
**Runtime Polymorphism:**In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.When reference variable of Parent class refers to the object of Child class, it is known as upcasting  
eg or pseudocode:  
class A{}  
class B extends A{}  
A a=new B();//upcasting  
**Note:**Upcasting we normally don’t use doesn’t have any effect or java will do its own, but downcasting or typecasting is something which we have to done by our self-nothing but the labelling the reference variable to another variable.**Doubt  
Abstract:**Abstraction is a process of hiding the implementation details and showing only functionality to the user.Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.  
**Rules for Abstract:**1. If a method is declared and not defined, i.e only its body is there we call it as abstract method where abstract keyword should be used followed by method name.  
eg: abstract void name();  
2. If a class has abstract method then we call it as abstract class where we have to place abstract keyword followed by class name.  
eg: abstract class Name{}  
3. If abstract class is extended then the abstract method should be defined in the extended child class.  
4. An abstract class can have data member, abstract method, Normal method, Normal constructor and even main() method.  
5. We cannot create an instance/object for the abstract class.  
**eg:**  
abstract class Shape{  
abstract void draw();  
}  
//In real scenario, implementation is provided by others i.e. unknown by end user  
class Rectangle extends Shape{  
void draw(){System.out.println("drawing rectangle");}  
}class Circle1 extends Shape{  
void draw(){System.out.println("drawing circle");}  
}  
//In real scenario, method is called by programmer or user  
class TestAbstraction1{  
public static void main(String args[]){   
Shape s=new Circle1();//In real scenario, object is provided through method e.g. getShape() method   
s.draw();  
}}o/p:  
drawing circle  
**When and Why we use Abstract Class?**We can go for abstract class by putting some common methods used by other two classes as well in order to avoid creating the object by the developer to the base class (i.e our abstract class). Code maintainability maintained here.  
**Interface:**An interface in java is a blueprint of a class. It has static constants and abstract methods only. The interface in java is a mechanism to achieve fully abstraction. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritances in Java.  
**Why we Use Interface?**It is used to achieve fully abstraction.  
By interface, we can support the functionality of multiple inheritances.  
  
  
eg:  
interface printable{  
void print(); //not necessary we need to give abstract key word since it will consider as abstract by default  
}  
class A6 implements printable{  
public void print(){System.out.println("Hello");}  
public static void main(String args[]){  
A6 obj = new A6();   
obj.print();   
}}  
**Multiple Inheritance using interface**  
Data Members in interface are **Public, Static,Final.We can implement any number of interfaces unlike abstract class which can extend only one class.  
All fields declared within an interface are implicitly public, static, and final. Why?Doubt**  
->Any implementations can change value of fields if they are not defined as final. Then they would become a part of the implementation.An interface is a pure specification without any implementation.  
If they are static, then they belong to the interface, and not the object, nor the run-time type of the object.  
->An interface provides a way for the client to interact with the object. If variables were not public, the clients would not have access to them.  
In general, a field declaration may include the following modifiers: public, protected, private, final, static, transient, volatile. But only public, final, and static are permitted for interface's variable.  
**Abstract vs Interface**

|  |  |
| --- | --- |
| **Abstract** | **Interface** |
| Abstract class can have abstract and non-abstract methods. | Interface can have only abstract methods. |
| Abstract class doesn't support multiple inheritance. | Interface supports multiple inheritance. |
| Abstract class can have final, non-final, static and non-static variables. | Interface has only static and final variables. |
| Abstract class can have static methods, main method and constructor. | Interface can't have static methods, main method or constructor. |
| Abstract class can provide the implementation of interface. | Interface can't provide the implementation of abstract class. |
| The abstract keyword is used to declare abstract class. | The interface keyword is used to declare interface. |
| Example:  public abstract class Shape{  public abstract void draw();  } | Example:  public interface Drawable{  void draw();  } |

**Java Package:  
  
Try () and catch ()**The exception handling in java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained. In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.The core advantage of exception handling is to maintain the normal flow of the application. Exception normally disrupts the normal flow of the application that is why we use exception handling.  
**Hierarchy of Java Exception Class**



* **Checked Exception:**The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.
* **Unchecked Exception:**In Java is those Exceptions whose handling is NOT verified during Compile time. These exceptions occur because of bad programming. The program won’t give a compilation error. Most of the times these exception occurs due to the bad data provided by user during the user-program interaction. All Unchecked exceptions are direct sub classes of RuntimeException class.

Below are types of Unchecked Exceptions in java:

• NullPointerException

• ArrayIndexOutOfBound

• IllegalArgumentException

• IllegalStateException

Below are checked exceptions which give meaningful message.

* SQLException
* IOException
* DataAccessException
* ClassNotFoundException
* InvocationTargetException

**Try{}:** java try block is used to enclose the code that might throw an exception. It must be used within the method**.  
Catch{}:** Java catch block is used to handle the Exception. It must be used after the try block only.  
In a program if exception is not handled the JVM will do the following process:  
1. Prints out exception description.  
2. Prints the stack trace (Hierarchy of methods where the exception occurred).  
3. Causes the program to terminate.  
**Note:**We can have multiple catch statements, but try block should be followed by either catch or finally block if not it will give us compile error. At the same time declare the specific exception in catch statement at the first when you are sure about which exception is thrown instead of going with general **Exception.  
eg:**public class Testtrycatch2{  
public static void main(String args[]){  
try{   
 int data=50/0;  
}catch(ArithmeticException e){System.out.println(e);}  
catch(Exception e){System.out.println("hgfhg");}  
 System.out.println("rest of the code...");  
}} **o/p:**ArithmeticException rest of the code  
**Remember**: Instead of that we have given general Exception first and followed by ArithmeticException, then it will give Complier Error telling that exception already caught and the rest of the code won’t be executed. **Remember:We can declare multiply try and catch block but only one finally block can be given.  
->Finally block in java can be used to put "cleanup" code such as closing a file, closing connection etc.  
-> Finally Block will not execute if the program exits (by calling System.exit() or fatal error).**->Final and Finally keyword in Java. Words can be confusing but remember both are entirely*different.*

|  |  |  |
| --- | --- | --- |
| **Final** | **finally** | **Finalize** |
| Final is used to apply restrictions on class, method and variable. Final class can’t be inherited, final method can't be overridden and final variable value can't be changed. | Finally is used to place important code, it will be executed whether exception is handled or not i.e. even though exception thrown finally code will run. | Finalize is used to perform clean up processing just before object is garbage collected. |
| Final is a keyword. | Finally is a block. | Finalize is a method. |

* The Java **throws** keyword is used to declare an exception. It gives information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained. Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.(compiletimeexception).  
  **Syntax**:

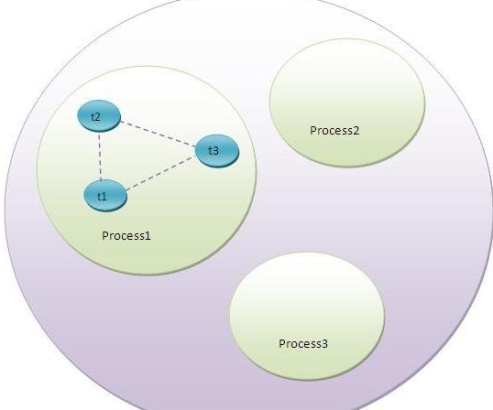
Public void Methoodname throws ArithmeticException,IOException{  
}

* The Java **throw** keyword is used to explicitly throw an exception. We can throw either checked or uncheked exception in java by throws keyword. The throw keyword is mainly used to throw custom exception.

Syntax:  
**throw** **new** IOException("sorry device error);

* Throws and Throw in Selenium Web driver

|  |  |
| --- | --- |
| **Throw** | **Throws** |
| * Java throw keyword is used to explicitly throw an exception. * Checked exception cannot be propagated using throw only. * Throw is followed by an instance. * Throw is used within the method. * You cannot throw multiple exceptions. | Java throws keyword is used to declare an exception.  Checked exception can be propagated with throws. Throws is followed by class. Throws is used with the method Signature. You can declare multiple exceptions e.g. public void method()throws IOException, SQLException. |

**Throws Custom Exceptions:**package beginnersbook.com;  
class MyOwnException extends Exception {  
 public MyOwnException(String msg){  
 super(msg);  
 }}  
class EmployeeTest {  
 static void employeeAge(int age) throws MyOwnException{  
 if(age < 0)  
 throw new MyOwnException("Age can't be less than zero");  
else  
System.out.println("Input is valid!!");  
}  
 public static void main(String[] args) {  
 try {  
employeeAge(-2);  
} catch (MyOwnException e) {  
 e.printStackTrace();  
 } }} **Rules to be followed for writing exceptions when it is overridingfrom parent to child class:**1. If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exceptionbut can declare unchecked exception.  
eg:  
class Parent{  
 void msg(){System.out.println("parent");}  
}   
class TestExceptionChild1 extends Parent{  
 void msg()throws ArithmeticException{ // this is fine but instead if we use try catch block obviously it will become checked exception which throws error  
 System.out.println("child");  
}  
 public static void main(String args[]){   
Parent p=new TestExceptionChild1();  
p.msg();  
}}  
2. If the superclass method declares angeneric exception (i.e. Exception), subclass overridden method can declare same, subclass exception (i.e. Specific exception like arithmeticexception e.t.c) or no exception but cannot declare parent exception (i.e like access specifier rules child class should have lower hierarchy or equal level of parent class).  
eg:  
class Parent{  
void msg()throws ArithmeticException{System.out.println("parent");}  
}  
class TestExceptionChild2 extends Parent{  
void msg()throws Exception{System.out.println("child");}//here we should give either arithmetic or give it in parent class as Exception   
public static void main(String args[]){   
Parent p=new TestExceptionChild2();  
 try{  
p.msg();   
}catch(Exception e){}   
}}  
O/P: compile error  **Thread:**  
Thread in java something like we can call it as subset of process. We will take real time example what is thread or multithreading and multitasking. In our system if we are opening multiple programs like word, excel, browser then we call it is multitasking. In a browser if we are opening multiple tabs that is called thread or multi treading. So thread is nothing but subset of multitasks.  
We use multithreading than multiprocessing because threads share a common memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.  
  
  
**The java thread states are as follows:**  
New  
Runnable  
Running  
Non-Runnable (Blocked)  
Terminated  
**Advantage of Java Multithreading:**  
1) It doesn't block the user because threads are independent and you can perform multiple operations at same time.  
2) You can perform many operations together so it saves time.  
3) Threads are independent so it doesn't affect other threads if exception occurs in a single thread.  
**Rules for Thread:**  
1. Thread is a inbuilt class in java, if we want to use it we have to extend the Thread class. It can also be achieved by implementing the Runnable interface.  
2. Then implement the method run().  
3. Don’t call run(), instead call start() so that it will run in thread manner. Or else it will become normal object.  
eg:  
public class HelloWorld extends Thread{  
 public void run(){  
 for(int i = 0;i<5;i++){  
 System.out.println(**Thread.currentThread().getName**()+" Value is:" + i);  
 try{  
 Thread.sleep(500);   
 }catch(InterruptedExceptione){  
 System.out.print(e);   
 } } }  
 public static void main(String[] args) {  
 HelloWorld obj = new HelloWorld();//thread object  
 obj.start(); //You should call start not run() if you do so then it will become normal program.  
HelloWorld obj1 = new HelloWorld();  
 obj1.start();  
 }}  
O/P:  
Thread-1 Value is:0  
Thread-0 Value is:0  
Thread-1 Value is:1  
Thread-0 Value is:1  
Thread-1 Value is:2  
Thread-0 Value is:2  
Thread-1 Value is:3  
Thread-0 Value is:3  
See in above example the output resembles it has printed the values concurrently. If you run normally then you will get output in sequential like 01234, 01234.  
eg: using Runnable interface  
public class HelloWorld implements Runnable{  
public void run(){   
for(int i = 0;i<5;i++){  
 System.out.println(Thread.currentThread().getName()+" Value is:" + i);  
} }  
public static void main(String[] args) {  
Thread t1 = new Thread(new HelloWorld()); //thread object  
(or)  
HelloWorld h1 = new HelloWorld();  
Thread t1 = new Thread(h1);//thread object  
 t1.start();  
 Thread t2 = new Thread(new HelloWorld());  
 t2.start();  
 }}  
same o/p like above program.  
**Remember:**  
1. We can’t Start a same thread twice.If does it will throw IllegalThreadStateException.  
**Commonly used methods of Thread class:**public void run(): is used to perform action for a thread.  
public void start(): starts the execution of the thread.JVM calls the run() method on the thread.  
public void sleep(long milliseconds): Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.  
public void join(): waits for a thread to die.  
public void join(long milliseconds): waits for a thread to die for the specified miliseconds.  
public int getPriority(): returns the priority of the thread.  
public int setPriority(int priority): changes the priority of the thread.  
public String getName(): returns the name of the thread.  
public void setName(String name): changes the name of the thread.  
public Thread currentThread(): returns the reference of currently executing thread.  
public int getId(): returns the id of the thread.  
public Thread.State getState(): returns the state of the thread.  
public boolean isAlive(): tests if the thread is alive.  
public void yield(): causes the currently executing thread object to temporarily pause and allow other threads to execute.  
public void suspend(): is used to suspend the thread(depricated).  
public void resume(): is used to resume the suspended thread(depricated).  
public void stop(): is used to stop the thread(depricated).  
public void interrupt(): interrupts the thread.  
public boolean isInterrupted(): tests if the thread has been interrupted.  
public static boolean interrupted(): tests if the current thread has been interrupted.  
**Note**: Now suspend(), resume() and stop() methods are deprecated.  
**Priority of a Thread:**Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread scheduler schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.  
**3 constants defiend in Thread class:**  
**public static int MIN\_PRIORITY  
public static int NORM\_PRIORITY  
public static int MAX\_PRIORITY**  
Default priority of a thread is 5 (NORM\_PRIORITY). The value of MIN\_PRIORITY is 1 and the value of MAX\_PRIORITY is 10.  
eg:  
class TestMultiPriority1 extends Thread{  
 public void run(){   
System.out.println("running thread name is:"+Thread.currentThread().getName()); System.out.println("running thread priority is:"+Thread.currentThread().getPriority());  
}  
 public static void main(String args[]){  
 TestMultiPriority1 m1=new TestMultiPriority1();  
 TestMultiPriority1 m2=new TestMultiPriority1();  
 m1.setPriority(Thread.MIN\_PRIORITY);  
 m2.setPriority(Thread.MAX\_PRIORITY);  
 m1.start();   
 m2.start();   
}}  
**Commonly used Constructors of Thread class:**Thread()  
Thread(String name)  
Thread(Runnable r)  
Thread(Runnable r,String name)  
**ThreadGroup**Java provides a convenient way to group multiple threads in a single object. In such way, we can suspend, resume or interrupt group of threads by a single method call.  
**There are only two constructors of ThreadGroup class:**  
ThreadGroup(String name) = creates a thread group with given name.  
ThreadGroup(ThreadGroup parent, String name) = creates a thread group with given parent group and name.  
eg:  
public class ThreadGroupDemo implements Runnable{  
 public void run(){  
 System.out.println(Thread.currentThread().getName());   
}  
public static void main(String[] args) {  
ThreadGroupDemo runnable = new ThreadGroupDemo();  
ThreadGroup tg1 = new ThreadGroup("Parent ThreadGroup"); //all threads into one group tg1  
Thread t1 = new Thread(tg1, runnable,"one");  
t1.start();  
Thread t2 = new Thread(tg1, runnable,"two");  
t2.start();  
Thread t3 = new Thread(tg1, runnable,"three");  
t3.start();  
System.out.println("Thread Group Name: "+tg1.getName());  
tg1.list();  
}}  
o/P:  
one  
two  
three  
Thread Group Name: Parent ThreadGroup  
java.lang.ThreadGroup[name=Parent ThreadGroup,maxpri=10]  
Thread[one,5,Parent ThreadGroup]  
Thread[two,5,Parent ThreadGroup]  
Thread[three,5,Parent ThreadGroup]  
**Thread PoolDoubt**Java Thread pool represents a group of worker threads that are waiting for the job and reuse many times. Better performance It saves time because there is no need to create new thread.  
eg:  
import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

class WorkerThread implements Runnable {

private String message;

public WorkerThread(String s){

this.message=s;

}

public void run() {

System.out.println(Thread.currentThread().getName()+" (Start) message = "+message);

processmessage();

System.out.println(Thread.currentThread().getName()+" (End)");

}

private void processmessage() {

try { Thread.sleep(2000); } catch (InterruptedException e) { e.printStackTrace(); }

}

}

public class SimpleThreadPool {

public static void main(String[] args) {

ExecutorService executor = Executors.newFixedThreadPool(5);

for (int i = 0; i < 10; i++) {

Runnable worker = new WorkerThread("" + i);

executor.execute(worker);

}

executor.shutdown();

while (!executor.isTerminated()) {

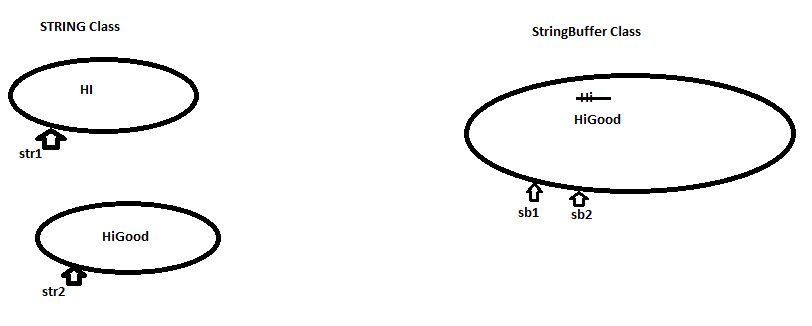
}

System.out.println("Finished all threads");

}

}  
**Properties File in Java**.properties is a file extension for files mainly used in Java related technologies to store the configurable parameters of an application. They can also be used for storing strings for Internationalization and localization; these are known as Property Resource Bundles.  
To load and get the value of property file:  
FileInputStream fileInput = new FileInputStream(propertyfile\_completepath);  
Properties propertiesob = new Properties();  
propertiesob.load(fileInput);  
String value = propertiesob.getProperty(key);  
Why We go For Properties file? **Doubt**  
Difference between Properties and txt file?**Doubt**  
**Mutable and Immutable Objects:**We call an object as Mutable only if we can able to modify its object. Immutable object is just vice versa of Mutable object. We can say **Stringclass or WrapperClass/non-primitivedatatype**as Immutable object and **StringBuffer**class as Mutable object.  
Some String Buffer methods are: s.insert(“startnumber”,newtext), s.append(“newstring”), s.reverse(), s.delete(“strtnumber”,”End number”).

|  |  |
| --- | --- |
| **String Buffer** | **String Builder** |
| StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously. | StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously. |
| StringBuffer is less efficient than StringBuilder and slower than String Builder. | StringBuilder is *more efficient* than StringBufferand faster than String buffer. |

public class ConcatTest{  
 public static void main(String[] args){  
 long startTime = System.currentTimeMillis();  
 StringBuffer sb = new StringBuffer("Java");   
 for (int i=0; i<10000; i++){  
 sb.append("Tpoint");  
 }  
System.out.println("Time taken by StringBuffer: " + (System.currentTimeMillis() - startTime) + "ms"); startTime = System.currentTimeMillis();   
 StringBuilder sb2 = new StringBuilder("Java");   
 for (int i=0; i<10000; i++){   
 sb2.append("Tpoint");   
 } System.out.println("Time taken by StringBuilder: " + (System.currentTimeMillis() - startTime) + "ms");   
 }}  
o/p:  
Time taken by StringBuffer: 16ms  
Time taken by StringBuilder: 0ms   
**String Class:**String datatype will not come under as a primitive. If we declaring a variable in String i.e for eg: String a = “Hi”; which means it is a string class internally the java will take as String a = new String(“Hi”); which means ‘a’ is acting as an object.  
**Now why we call String class as Immutable object and String Buffer as Mutable object.**  
eg:   
public static void main(String[] args) {  
String str1 = new String("Hi");  
String str2 = str1.concat("Good");  
System.out.println(str1);  
System.out.println(str2);  
System.out.println("====");  
StringBuffer sb1 = new StringBuffer("Hi");  
StringBuffer sb2 = sb1.append("Good");  
System.out.println(sb1);  
 System.out.println(sb2);  
 }  
O/P:  
Hi  
HiGood  
=======  
HiGood  
HiGood  
Some String class Functions: s.concat(), s.length(), s.toLowerCase(), s.toUppercase(), s.replace(“newstring”,”string to be replaced”), s.compareTo(string to compare), s.substring(“startnumber”,”endnumber”)  
In the above example str1 and str2 are the two different objects created and each has assigned a value for it and when we print it displayed its values. While StringBuffer also sb1 and sb2 but show same value because it is mutable it can be modified once the sb2 came it overwrites the value of sb1 too which means sb1 also refers sb2 object only. Refer below diagram to get a clear idea.  
  
  
**When and Where to Use String Class, String Builder, String Buffer?Doubt**  
1. If the Object value is not going to change use String Class because a String object is immutable.  
2. If the Object value can change and will only be accessed from a single thread, use a StringBuilder because StringBuilder is unsynchronized.  
3. In case the Object value can change, and will be modified by multiple threads, use a StringBuffer because StringBuffer is synchronized.  
So before that let see small example or let’s have an idea what is String Class and Wrapper Classs.  
**WrapperClass**

|  |  |
| --- | --- |
| **Primitive Type** | **Wrapper class** |
| Boolean | Boolean |
| Char | Character |
| Byte | Byte |
| Short | Short |
| Int | Integer |
| Long | Long |
| Float | Float |
| Double | Double |

eg: For converting a primitive to a wrapper class.  
public class WrapperExample1{   
public static void main(String args[]){   
int a=20;   
Integer I =Integer.valueOf(a);//converting int into Integer   
Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally   
System.out.println(a+" "+i+" "+j);   
}}  
o/p:  
20 20 20  
eg: For converting a Wrapper to a primitive type.  
public class WrapperExample2{   
public static void main(String args[]){   
Integer a=new Integer(3);   
int i=a.intValue();//converting Integer to int   
int j=a; //unboxing, now compiler will write a.intValue() internally   
System.out.println(a+" "+i+" "+j); }}  
o/p:  
20 20 20  
**Note:** We may feel why we want this wrapper class, We can say for an example of when wrappers are used would be in Collections, you can have an ArrayList<Integer>, but not an ArrayList<int> same with HashMaps etc. To get type safety we use generics and generics need objects not primitives.  
**What is the difference between primitive data types and wrapper classes in java?**The main difference between primitive data types and wrapper classes (wrapper types) in java is that the primitive types can be used as raw data for operations such as arithmetic, logical, etc. and wrapper classes acts as data holders for these primitive data types.  
**Remember:** The primitive data type values will be stored in Stack Memory whereas wrapper class objects (like any other java objects) are stored in Heap Memory.  
**Generic Class in Java:**The Java Generics programming is introduced in J2SE 5 to deal with type-safe objects.Before generics, we can store any type of objects in collection i.e. non-generic. Now generics, forces the java programmer to store specific type of objects.  
**Advantage of Java Generics**There are mainly 3 advantages of generics. They are as follows:  
1) Type-safety : We can hold only a single type of objects in generics. It doesn’t allow storing other objects.  
2) Type casting is not required: There is no need to typecast the object.  
**Before Generics, we need to type cast.**List list = new ArrayList();   
list.add("hello");   
String s = (String) list.get(0);//typecasting  
**After Generics, we don't need to typecast the object.**List<String> list = new ArrayList<String>(); //<>implies generic  
list.add("hello");   
String s = list.get(0);

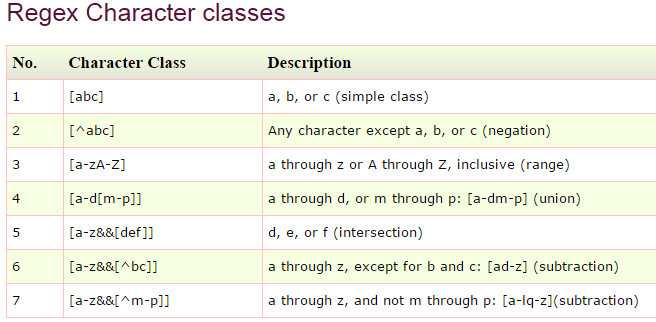
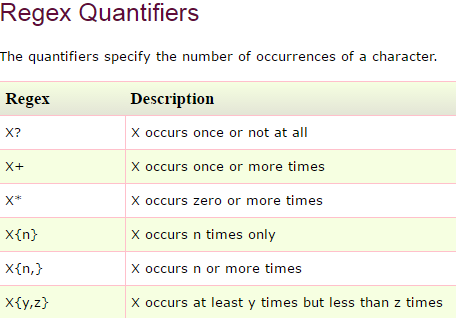
**Collections:**Collections in java is a framework that provides an architecture to store and manipulate the group of objects. All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.-> Collection represents a single unit of objects i.e. a group.  
-> Some common methods used in collections are:  
**public boolean add(Object element) -** is used to insert an element in this collection.  
**public Boolean addAll(Collection c) -** is used to insert the specified collection elements in the invoking collection.  
**public boolean remove(Object element)**  - is used to delete an element from this collection.  
**public Iterator iterator()**- returns an iterator.  
**Remember:**In the collection frame work, we have major two interfaces:  
1.List  
2.Set  
**Note:**  
List implementing classes can accept null values, and duplicate values, so we can accept two or more null values. But, in case of Set implemented classes it can accept only one null value, because, set cannot accept duplicate elements. **ArrayList:**Java ArrayList class can contain duplicate elements.  
Java ArrayList class maintains insertion order.  
In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.  
import java.util.\*;   
class TestCollection1{   
 public static void main(String args[]){   
ArrayList<String> al=new ArrayList<String>();//creating arraylist  
al.add("Ravi");//adding object in arraylist  
al.add("Vijay");   
al.add("Ravi");   
al.add("Ajay");   
ArrayList<String> al2=new ArrayList<String>();   
al2.add("Sonoo");   
al2.add("Hanumat");   
al.addAll(al2);  
al.removeAll(al2);  
Iterator<String> itr=al.iterator();//getting Iterator from arraylist to traverse elements   
 while(itr.hasNext()){   
System.out.println(itr.next());   
} }  
**Syntax**:  
al.clear() ;// clear all the elements  
al.addAll(al2);//to add two arraylist  
al.removeAll(al2); //removes the values of al which is present in al2.  
al.retainAll(al2); //print the values of al which is present in the al2  
al.equals(al2) //compare values are equal of two arraylist  
al.add(“value)”; //to add an value  
al.set(indexnumber,“new value”); //modify the value  
al.remove(“value/index”); // to remove the element from list.  
System.out.print(a.get(i)); //to print the individual value of the list  
for(String str : al) { //to print multiple values in the list we can use for or above example iterator   
 System.out.println(str);  
 }  
al.size(); //to get the size of the list  
al.contains("value") //to check whether the value exists.  
Collections.sort(al); //to sort values of al in ascending order.  
**Linked List**Java LinkedList class can contain duplicate elements.  
Java LinkedList class maintains insertion order.  
In Java LinkedList class, manipulation is fast because no shifting needs to be occurred.  
Java LinkedList class can be used as list, stack or queue.  
Import java.util.\*;   
public class TestCollection7{   
 public static void main(String args[]){   
LinkedList<String> al=new LinkedList<String>();   
al.add("Ravi");   
al.add("Vijay");   
Iterator<String>itr=al.iterator();   
while(itr.hasNext()){   
System.out.println(itr.next());   
 } } }  
**Syntax**:  
al.add(“value)”; //to add an value  
al.set(indexnumber,“new value”); //modify the value  
al.remove(“value”); // to remove the value from list.  
Syso(a.get(i)); //to print the individual value of the list  
for(String str : al){ //to print multiple values in the list we can use for loop or iterator().   
System.out.println(str);  
 }  
System.out.println(al.size()); //to get the size of the list  
al.contains("value") //to check whether the value exists.  
**Remember:**Above all methods are applicable for list also.

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList internally uses **dynamic array** to store the elements. | LinkedList internally uses **doubly linked list** to store the elements. |
| 2) Manipulation with ArrayList is **slow** because it internally uses array. If any element is removed from the array, all the bits are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses doubly linked list so no bit shifting is required in memory. |
| 3) ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| 4) ArrayList is better for storing and accessing data. | LinkedList is better for manipulating data. |

Please Note that in above example we have been using **Iterator** for getting the elements, now we use **ListIterator.** Mainly by using **ListIterator** traverse the element in backward and forward direction. Common methods in the ListIterator is listed below in that **hasPrevious**() and **previous**() is additional compared to normal iterator.  
**Public booleanhasNext();  
public Object next();  
public booleanhasPrevious();  
public Object previous();**example:import java.util.\*;   
public class TestCollection8{   
public static void main(String args[]){   
ArrayList<String> al=new ArrayList<String>();  
al.add("Amit");  
al.add("Vijay");  
al.add("Kumar");  
al.add(1,"Sachin"); // it will insert the saching in array index 1.  
System.out.println("element at 2nd position: "+al.get(2));   
ListIterator<String>itr=al.listIterator();   
System.out.println("traversing elements in forward direction...");   
while(itr.hasNext()){   
System.out.println(itr.next()); }  
System.out.println("traversing elements in backward direction...");   
while(itr.hasPrevious()){   
System.out.println(itr.previous()); }}}  
Output:element at 2nd position: Vijay  
**traversing elements in forward direction.**  
Amit  
Sachin  
 Vijay  
 Kumar  
**traversing elements in backward direction**.  
 Kumar  
 Vijay  
Sachin  
Amit  
al.contains("value") //to check whether the value exists.  
**Hash Set:**-> uses hashtable to store the elements. It extends Abstract Set class and implements Set interface.  
-> contains unique elements only. It accept null values.  
Note: List can contain duplicate elements whereas Set contains unique elements only.  
importjava.util.\*;   
class TestCollection9{   
public static void main(String args[]){   
HashSet<String> al=new HashSet<String>();   
al.add("Ravi");   
al.add("Vijay");   
al.add("Ravi");   
al.add("Ajay");   
 Iterator<String>itr=al.iterator();   
 while(itr.hasNext()){   
System.out.println(itr.next()); }}}  
O/P: Ravi,vijay,ajay  
Values cannot be modified in the Hashset.  
al.contains("value") //to check whether the value exists.  
al.remove("value/index");  
al.removeAll(al2); //removes the values of al which is present in al2.  
al.retainAll(al2); //print the values of al which is present in the al2  
al.addAll(al2);  
**Tree Set**  
Similar to hashset only difference it will be in sorted manner i.e in ascending order by default. According to the above example the output here will be: Ajay, Ravi, Vijay.  
**Hash Map**A HashMap contains values based on the key. It implements the Map interface and extends AbstractMap class.It can have duplicate values but not keys. It may have one null key and multiple null values.It maintains no order.  
eg:   
import java.util.\*;   
class TestCollection13{   
 public static void main(String args[]){   
HashMap<Integer,String>hm=new HashMap<Integer,String>();   
hm.put(100,"Amit");   
hm.put(101,"Vijay");   
hm.put(102,"Rahul");   
 **for(Map.Entry m:hm.entrySet()){** //Map entry & Entryset with for  
**System.out.println(m.getKey()+" "+m.getValue());   
 }** (or)  
**for (Integer key : hm.keySet()) {**//for with keyset **System.out.println(key+" "+hm.get(key));  
}(or)  
Iterator it = hm.entrySet().iterator();**//iterator with entryset **while (it.hasNext()) {  
 Map.Entry pair = (Map.Entry)it.next();  
System.out.println(pair.getKey()+" "+pair.getValue());  
}  
}}**Output:102 Rahul  
100 Amit  
101 Vijay  
To get all the keys we can use System.out.println(hm.**keySet**());. Similarly for values System.out.println(hm.**values**());.**isEmpty()**to check whether it empty or not, The**entrySet()**method is used to get the values and keys of this map. In hasmap if we want to change any value just give hm.put(keyvalue,”newvalue”) i.e it will overwrite if you give same key which already exists.  
hm.get(key); //will print the value of that key.  
map.containsKey(key); //Booloean will check that key is present.

|  |  |
| --- | --- |
| **ArrayList** | **Hash Map** |
| ArrayList stores the element’s value alone and internally maintains the indexes for each element. | HashMap stores key & value pair. For each value there must be a key associated in HashMap. That clearly shows that memory consumption is high in HashMap compared to the ArrayList. |
| ArrayList maintains the insertion order array, all the bits are shifted in memory. Which means ArrayList returns the list items in the same order in which they got inserted into the list | HashMap doesn’t. On the other side HashMap doesn’t maintain any order, the returned key-values pairs are not sorted in any kind of order. |
| ArrayList allows duplicate elements. | HashMap allows one null key and any number of null values. But allows unique elements. |
| In ArrayList we can get the element by specifying the index of it | In HashMap the elements is being fetched by specifying the corresponding key. |

**Hash Table**A Hashtable is an array of list. Each list is known as a bucket. The position of bucket is identified by calling the hashcode() method. A Hashtable contains values based on the key. It implements the Map interface and extends Dictionary class. It contains only unique elements. It may have not have any null key or null value.  
**Regular Expression(Doubt)**

 **eg:**import java.util.regex.\*;   
class RegexExample3{   
public static void main(String args[]){   
System.out.println(Pattern.matches("[amn]", "abcd"));//false (either a alone or m alone or n alone should be there)   
System.out.println(Pattern.matches("[amn]", "a"));//true (among a or m or n) System.out.println(Pattern.matches("[amn]", "ammmna"));//false (m and a comes more than once)   
}}  
  
import java.util.regex.\*;   
class RegexExample4{   
public static void main(String args[]){  
System.out.println(Pattern.matches("[amn]?", "a"));//true (a or m or n comes one time)  
System.out.println(Pattern.matches("[amn]?", "aaa"));//false (a comes more than one time)System.out.println(Pattern.matches("[amn]?", "aammmnn"));//false (a m and n comes more than one time)  
System.out.println(Pattern.matches("[amn]?", "aazzta"));//false (a comes more than one time) System.out.println(Pattern.matches("[amn]?", "am"));//false (a or m or n must come one time)  
System.out.println(Pattern.matches("[amn]+", "a"));//true (a or m or n once or more times)  
System.out.println(Pattern.matches("[amn]+", "aaa"));//true (a comes more than one time)  
System.out.println(Pattern.matches("[amn]+", "aammmnn"));//true (a or m or n comes more than once)  
System.out.println(Pattern.matches("[amn]+", "aazzta"));//false (z and t are not matching pattern)  
System.out.println(Pattern.matches("[amn]\*", "ammmna"));//true (a or m or n may come zero or more times)  
}}

**JIRA  
Resolutions**An issue can be resolved in many ways, only one of them being "Fixed". The defined resolutions are listed below. You can add more in the administration section.

**Fixed:** A fix for this issue is checked into the tree and tested.

**Won't Fix:** The problem described is an issue which will never be fixed.

**Duplicate:** The problem is a duplicate of an existing issue.

**Incomplete:** The problem is not completely described.

**Cannot Reproduce:** All attempts at reproducing this issue failed, or not enough information was available to reproduce the issue. Reading the code produces no clues as to why this behavior would occur. If more information appears later, please reopen the issue.

**Invalid:** Issue has found to be invalid.

**Resolved Locally:** The issue was local to the user's installation (a support request, not a bug)

**Handled by Support:** This Issue will now be handled in the http://support.atlassian.com

**Answered:** Support query has been answered, or issue confirmed

**Timed out:** Issue closed due to lack of response - feel free to reopen if necessary.

**Obsolete:** Issue was valid at some stage, but has become redundant due to other developments

**Not a bug:** The issue is not a bug (eg. it may be a usage or documentation issue).

**Unsolved Mysteries:** Issues where no real solution to the underlying problem has been found but everyone is quite happy to let it go

**Purchased:** Resulted in purchase

**Tracked Elsewhere:** The issue is not tracked on this instance, either because it's a third-party problem or because there is another JIRA instance which tracks this issue.

**Deployed:** The issue has been deployed into production

**Done**

**Support Request:** This problem should be resolved via a support request.

**Won't Do:** This issue won't be actioned.

#### Security Levels An issue has a security level which indicates which users are able to view the issue. The currently defined security levels are listed below. In addition, you can add more security levels in the administration section. Reporter and Atlassian Staff Priority Levels An issue has a priority level which indicates its importance. The currently defined priorities are listed below. In addition, you can add more priority levels in the administration section. Critical: Crashes, loss of data, severe memory leak. Major: Major loss of function. Minor: Minor loss of function, or other problem where easy workaround is present.Trivial: Cosmetic problem like misspelt words or misaligned text. Issue Types in JIRA

#### Note: Above mentioned are the main three parametersi.e Resolutions, Issue Types and Priority Levels involved in the JIRA when posting an issue. Sample Bug Life Cycle in Jira: Requirement Traceability Matrix or Test Coverage: It is something that checks whether out test cases has been covered the entire requirement document. Below screen shot will show you how our test cases are covering the requirement document. Step 1: Business Requirement Document( High level document) Step2: We are splitting BRDD to low level document (Functional Requirement Document) Step 3: Writing Test Scenarios Mapping with the Id of the FRD. Note there can be multiple Test Scenarios for each FRD. Step4: Now our RTM looks like below screen shot. Step5: Now writing the test cases for the test scenarios. Multiple test cases can be there for each test scenarios.

This is how the Rtm Looks.