1. **Types of polymorphism.**

It is one of the OOPS principle where **one object showing different behavior at different stages of its life cycle**.

The object **behavior goes on changing based on stage.**

In java polymorphism is classified in **2 types**

Compile time polymorphism (Overloading)

Runtime Polymorphism (Overriding).

2. **Difference between method overloading and method overriding. And where you use this concept in Selenium?**

Method Overloading	Method Overriding
Method overloading is **compile time** polymorphism.	Method overriding is **runtime** polymorphism.
When two or more methods in one class having same name but **different signature** then it is called as method overloading.	When parent class and child class have same method name with **same signature** then it is called as method overriding.
Different signature means	Same signature means
No. of parameters/ arguments is different.	No. of parameters/ arguments is same.
Type of parameters/arguments is different.	Type of parameters/arguments is same.
Sequence of parameter/arguments is diff.	Sequence of parameter/arguments is same.
It is also called as **early binding**.	It is also called as **late binding**.
We can overload non static as well as static methods.	We can override only non static methods We cannot override static methods.
Method overloading is **performed within the class.**	For method overriding, we **need two class i.e. parent class and child class.**
In selenium there are few methods are overloaded :	**In selenium** there are few methods are overrided:

1) implicitlyWait()	**1) findElement()**
2) pageLoadTimeOut();	**2) findElements()**
3) frame()	
4) navigate().to()	
5) actions() method	
We can overload constructor.	We cannot override the constructor.
We can overload private method.	We cannot override the private method.
In method overloading which method to call is decided by compiler at compile time.	In method overriding which method to be executed is decided by JVM at run time. The decision takes place according to object that we called.

3. Can we override static method?

No, we cannot override static method because method overriding is based on dynamic binding at runtime and the static methods are bonded using static binding at compile time.

4. Difference between Abstract class and Interface and where you use this concept in Selenium?

Abstract class	**Interface**
An abstract class can extend only one class or one abstract class at a time.	An interface can extend any number of interfaces at a time.
An abstract class can extend another concrete (regular) class or abstract class.	An interface can only extend another interface.
An abstract class can have both abstract and concrete method.	An interface can have only abstract methods.

If we want to use abstract class in another class we use "**extends**" keyword	If we want to use interface in other class then we use "**implements**" keyword.
The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
A Java abstract class can have class **members** like **private, protected, etc**.	All **members** of a Java interface are **public**.
Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
We can write constructor in abstract class.	We cannot write constructor in Interface.
In Selenium most of the **methods are Abstract.**	**In Selenium WebDriver, WebElement, WebElements, navigate, Alert, options timeOuts, window** are some Interfaces.

5. Static block and Instance block.

Static block	**instance block**
Known only as static initialization block in java.	Also known as non-static initialization block in java.
Static blocks executes before instance blocks in java.	Instance blocks executes after static blocks in java.
Only static variables can be accessed inside static block	Static and non-static variables (instance variables) can be accessed inside instance block.
static blocks can be used for initializing static variables	instance blocks can be used for initializing instance variables
or	or
calling any static method in java.	calling any instance method in java.
Static blocks executes when class is	Instance block executes only when instance of

loaded in java.	class is created, not called when class is loaded in java.
this keyword **cannot be used** in static blocks.	**this** keyword **can be used** in instance block.

6. Encapsulation and common use cases

Encapsulation:

It is one of the OOPS concept It is a **process of wrapping code and data in a single unit.**

To access private fields in another class we use encapsulation.

In encapsulation we have control over the data.

Encapsulation is technique making private fields in class accessible in other class by **using getter and setter methods**.

By providing **getter and setter** method we can **read and write** the private data members.

Syntax:

```
        privateString name="India";
publicString getName()                          ------    For getter
        {
                Statement;
                Returnname;
        }
public voidsetName(String newname)              ------    For setter
        {
                name= newname;
        }
```

Then we have to call getter and setter method in other class with the help of object.

7. Inheritance in java and where you use this concept in java?

Inheritance:

It is one of the OOPS principle where one class acquire the property of another class with the help of extend keyword.

It is **process of inheriting or reusing class members** (variables and methods) **from one class to another class**

The class from where the class member getting inherited is called as **parent class or super class or base class.**

The class in which the class members are getting inherited is called as **child class or derived class or sub class.**

The inheritance between parent class and child class achieved using "**extends**" keyword.

There are five types of Inheritance,

1. Single Level Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance
4. Multiple Inheritance
5. Hybrid Inheritance

Single Level Inheritance:

It is an operation where **inheritance** takes place in **between two classes**.

To perform single level inheritance **2 classes are mandatory**.

Multilevel Inheritance:

Multilevel inheritance takes place **between 3 or more classes**.

In multilevel inheritance 1 sub class acquire the property of another super class and that class acquire the properties of another super class and the properties of another super class and phenomenon continued it is known as multilevel inheritance.

Hierarchical Inheritance:

Hierarchical inheritance takes place in between **once super class and multiple subclasses.**

Property of super class can be acquired by multiple sub class this is known as hierarchical in heritance.

Multiple Inheritance:

One sub class acquire the **property of 2 super class at a time** is known as multiple inheritance.

Java does not support multiple inheritance in class perspective.

Because it result in diamond ambiguity problem

Hybrid Inheritance:

It is combination of two or more inheritance type.

Java does not support Hybrid inheritance in class perspective.

8. **What is casting in java and types of casting and where we use this concept in Selenium?**

Casting:

Casting is a method or process that **converts one info type into another info type** in both ways manually and automatically.

In java casting is classified into 2 types

1. Primitive casting
2. Class/Non primitive casting

Primitive casting:

Converting one data type info into another data type is known as primitive casting

Assigning info of 1 data type into another is known as primitive casting.

There are 3 types of Primitive casting

1. **Implicit casting:**

   Converting **lower data type info into higher data type** info is called as implicit casting.

   To perform this casting operation external resource support not required. So it is also called as automatic casting.

   It is also known as widening casting where memory size goes on increasing.

2. **Explicit casting:**

   Converting **higher size data type info into lower size** data type info is known as explicit casting.

   In this type of casting **external resource support is required**.

   Explicit casting is also known as narrowing casting.

3. **Boolean Casting:**

   Java does not support boolean casting.

   Boolean casting is considered as incompatible casting type, because Boolean data type is unique type of data. Where **info is already pre declared** inside it (True/False).

Non-primitive casting:

Converting one type of class into another type of class or assigning property of one class into another class is known as class casting.

Class casting is classified into 2 types;

i. Up casting

ii. Down casting

i. Up casting:

Assigning the sub class property into super class or parent reference variable is known as up casting

To perform up casting operation inheritance is required.

After performing inheritance operation the properties which are present inside the super class come into sub class, sub class programmer can declare the methods

At the time of up casting the property which is inherited from super class are only eligible for operation the new property which is declared inside the sub class is not eligible for operation.

Up casting is known as implicit up casting because casting takes place automatically.

ii. Down casting:

Java does not support down casting ""directly".

Before performing down casting first we have to perform up casting operation

Down casting is also known as explicit down casting.

Where we use casting concept in Selenium?

We use **upcasting** concept by giving reference of WebDriver interface and creating object of ChromeDriver.

We use **down casting** concept at the time of take screen shot method.

9. String Buffer and String Builder.

StringBuffer	StringBuilder

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.	StringBuilder is *more efficient* than StringBuffer.
StringBuffer was introduced in Java 1.0	StringBuilder was introduced in Java 1.5

10. Why String is immutable in java?

The String is immutable in Java because of the security, synchronization and concurrency, caching, and class loading. The reason of making string final is to destroy the immutability and to not allow others to extend it.

The String objects are cached in the String pool, and it makes the String immutable. The cached String literals are accessed by multiple clients. So, there is always a risk, where action performs by one client affects all other clients.

For example, if one client performs an action and changes the string value from Pressure to PRESSURE, all remaining clients will also read that value. For the performance reason, caching of String objects was important, so to remove that risk, we have to make the String Immutable.

11. How to handle exceptions in java?

Exception handling:

- an exeption is an event that occur during execution of a program when normal execution of a program interrupted

- it is used handle the run time exception

there 2 types

1) syntax error [compile time error]

when we writing code that time we will get error is called as syntax error

2) run time error

after running a program or executing a program when we will get error is called run time error

----

types of execution

1) checked exception

the classes that interit the throwable class known as checked exceptions.

IOException, SQLException. classNotFoundException ect...

2) uncheked exception

the classes that inherit the Runtime exception are known as unchecked exception

AritemeticException, NullpointerException, NumberFormatException, arrayOutOfBoundException ect...

most common exception in java

1) arithmetic exception occur

if we devide any number by zero then we will get AritmeticException

```
    package Tutorial4;

public class Demo2 {

    public static void main(String[] args) {

                System.out.println("MMS");
                int a=10;
                int b=0;
                int c=a/b;
                System.out.println(c);
                System.out.println("MME");
    }

}
```

==============================

2) NullPointerException

if we have no value for any variable ,but still we are perform action or operation on variable t null is reserved keyword in java

nothing assigned

```
package Tutorial4;

public class Demo2 {

	public static void main(String[] args) {

				System.out.println("MMS");
				String a=null;
				System.out.println(a.length());
				System.out.println("MME");
	}

}
```

=====================================

3) NumberFormatException

the wrong formatting of variable

```
package Tutorial4;

public class Demo2 {

	public static void main(String[] args) {

				System.out.println("MMS");
				String a="java";
				int b=Integer.parseInt(a);
				System.out.println(b);
				System.out.println("MME");
	}

}
```

==

4) ArrayIndexOut of Bound of exception

if we assign any value of wrong index then we willl get ArrayIndexOutOfBoundException

```
package Tutorial4;

import java.util.Arrays;

public class Demo3 {

    public static void main(String[] args) {

            int a [] = new int[5];

            a[0]=10;
            a[1]=20;
            a[2]=30;
            a[3]=40;

            a[10]=50;

            System.out.println(Arrays.toString(a));

    }

}
```

=================================

try catch() block

exception handling is not solution for handle run time exception

syntax:

```
    try{
            //risky code
    }
```

```
catch(Excetion e)
{
        //statement
}
```

The **Exception Handling in Java** is *mechanism to handle the runtime errors* so that the normal flow of the application can be maintained.

The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions.

Java provides **five keywords** that are used **to handle the exception**.

- **try:**

The "try" keyword is used to specify a block where we should place an exception code.We write the risky code here in try block.

We can't use try block alone. The try block must be followed by either catch or finally.

- **catch:**

The "catch" block is used to handle the exception.

It must be preceded by try block which means we can't use catch block alone.

It can be followed by finally block later.

- **finally:**

The "finally" block is used to execute the necessary code of the program.

It is executed whether an exception is handled or not.

It is executed after try block or catch block whichever is executed.

- **throw:**

The "throw" keyword is used to throw an exception.

- **throws:**

The "throws" keyword is used to declare exceptions.

It specifies that there may occur an exception in the method.

It doesn't throw an exception. It is always used with method signature.

Common Scenarios of Java Exceptions

1) A scenario where **ArithmeticException occurs:**

If we divide any number by zero, there occurs an **ArithmeticException**.

2) A scenario where **NullPointerException occurs:**

If we have a null value in any variable, performing any operation on the variable throws a **NullPointerException**.

3) A scenario where **NumberFormatException occurs:**

If the formatting of any variable or number is mismatched, it may result into NumberFormatException. Suppose we have a string variable that has characters; converting this variable into digit will cause **NumberFormatException**.

4) A scenario where **ArrayIndexOutOfBoundsException occurs**

When an array exceeds to its size, the **ArrayIndexOutOfBoundsException** occurs.

Can I write try block without the catch block

Yes we can write try block without catch block.
We need to use finally block with the try block.

We cannot write catch block without try block.

12. Difference between throw and throws?

Throw	Throws
Java throw keyword is used throw an exception explicitly in the code, inside the function or the block of code.	Java throws keyword is used in the method signature to declare an exception which might be thrown by the function while the execution of the code.
The throw keyword is followed by an instance of Exception to be thrown.	The throws keyword is followed by class names of Exceptions to be thrown.
throw is used within the method.	throws is used with the method signature.
We are allowed to throw only one exception at a time i.e. we cannot throw multiple exceptions.	We can declare multiple exceptions using throws keyword that can be thrown by the method. For example, main() throws IOException, SQLException.

13. Use of iterator in java

Iterator is a cursor in collection interface.
We use iterator to print all the values from list interface and also for set interface.

We use iterator only for collection interface methods.

We can use enhance for loop, normal for loop instead of iterator.

14. Difference in final, finally and finalize

Final	finally	finalize
final is the **keyword**.	finally is the **block**.	finalize is the **method** in Java.
It is used to **apply restrictions** on a variable, class or method.	It is used to **execute the important code** at the time of exception handling whether the exception occurs or not.	It is used to **perform clean up processing** just before object is garbage collected.
Final keyword is **used with** the **classes, methods and variables.**	Finally block is always **related to the try and catch block** in exception handling.	finalize() method is **used with the objects.**
1) Once declared, final variable becomes constant and cannot be modified. (2) final method cannot be overridden by sub class. (3) final class cannot be inherited.	(1) finally block runs the important code even if exception occurs or not. (2) finally block cleans up all the resources used in try block	finalize method performs the cleaning activities with respect to the object before its destruction.

Final method is **executed only when we call it.**	Finally block is **executed as soon as the try-catch block is executed.** Its execution is not dependent on the exception.	finalize method is **executed just before the object is destroyed.**

15. Auto Boxing and auto unboxing in java

Wrapper class:

>wrapper class are non primitive classess present inside the java.lang package

> in java, primitive data type design on the basis of wrapper classes

e.g each and every primitive data type there should be wrapper class.

> inside the wrapper class the characteritics of data types is declared.

>In java , infor can be stored usinng wrapper classes.

e.g

```
int a=10;

Integer b=a;
```

there are 2 types

1) auto boxing:

converting primitive data type info into wrapper type of info is known as auto boxing

2) auto unboxing:

converting wrapper type of info into primitive data types is called as auto unboxing

```
public class Demo1 {

public static void main(String[] args) {
        int a=100;
        System.out.println(a);
        Integer b=a;
        System.out.println(b);
```

```
            Character c='A';
            System.out.println(c);

            char d=c;
            System.out.println(d);
    }

    }
```

16. Difference between this and super keyword in java.

This	Super
This keyword is used to access global data members of current class or object.	Super keyword is used to access global data members of super class.
The current instance of the class is represented by this keyword.	The current instance of the parent class is represented by the super keyword.
In order to call the default constructor of the current class, we can use this keyword.	In order to call the default constructor of the parent class, we can use the super keyword.
It can be referred to from a static context. It means it can be invoked from the static context.	It can't be referred to from a static context. It means it cannot be invoked from a static context.
We can use it to access only the current class data members and member functions.	We can use it to access the data members and member functions of the parent class.

17. What is the Base class of all class in java

The **Object class is the parent class of all the classes** in java by default. In other words, it is the topmost class of java.

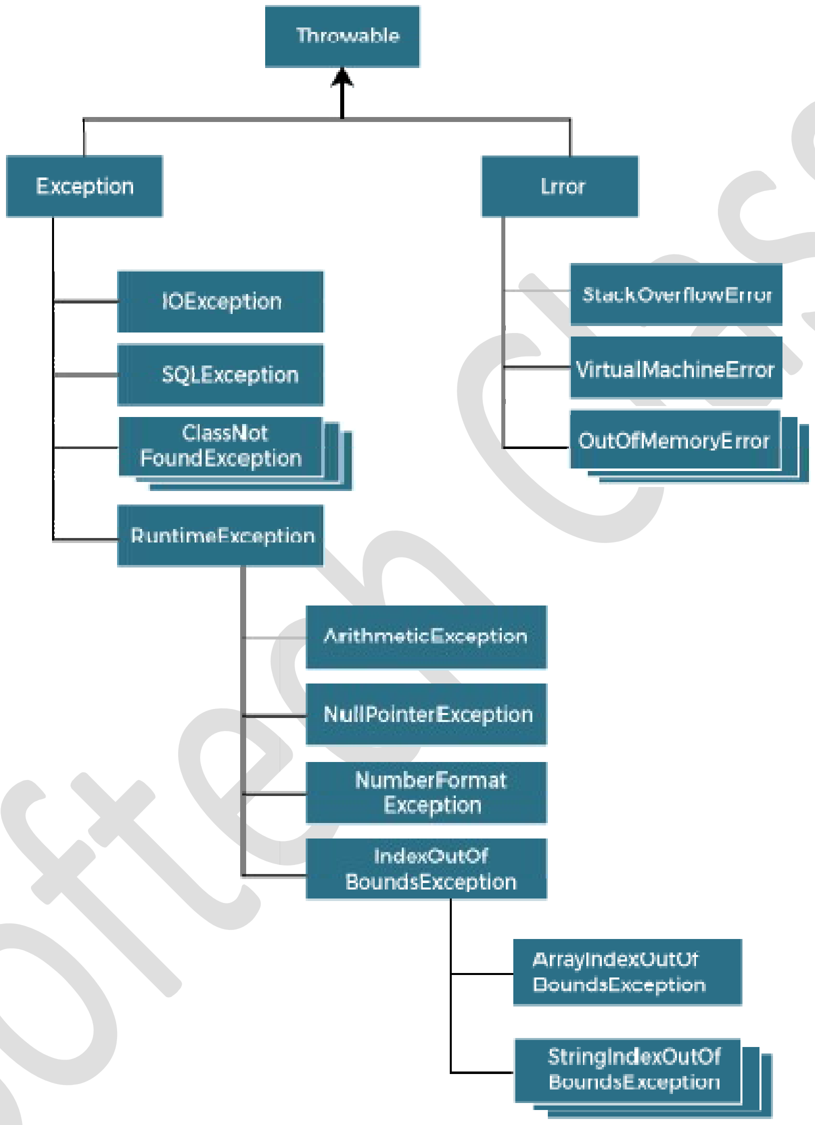
The Object class is beneficial if you want to refer any object whose type you don't know. Notice that parent class reference variable can refer the child class object, known as upcasting.

java.lang.Object class is the super base class of all Java classes.

18. Parent class of all the classes in error and exceptions in JAVA

Throwable is the base class of all exception and error classes in Java.

Under Throwable the Exception and Errors come



19. What are the different Access modifierand its type and what is the scope of each and every accessmodifier.

Access specifiers are used to represent scope of members of a class.

In java access specifiers are classified into 4 types.

I. Private

II. Default

III. Protected

IV. Public

I. Private:

If you declare any members of the classes as private then scope of that member remains only within the class, it can't be access from outer class.

II. Default:

If you declare any member of class as default than scope of that member only within the package, it can't be accessed from outer package.

There is no keyword to represented default data members.

III. Protected:

If you declare any members of the class as protected then scope of the member's remains within the package.

But we can access it outside the package also by using inheritance concept.

IV.Public:

If we declare any members of the class as public then scope of that member remains throughout the project i.e. any class can access it.

Non-Access Modifiers:

There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc. Here, we are going to learn the access modifiers only.

Access Modifier	within class	within package	outside package	outside package

			by subclass only	
Private	**Yes**	No	No	No
Default	**Yes**	**Yes**	No	No
Protected	**Yes**	**Yes**	**Yes**	No
Public	**Yes**	**Yes**	**Yes**	**Yes**

1. Difference between == and equals()

==	**equals()**
== is an operator.	equals() is a method of Object class.
== should be used during reference comparison.	equals() method should be used for content comparison

== checks if both references points to same location or not.	. equals() method evaluates the content to check the equality.
== operator can not be overridden.	equals() method if not present and Object.equals() method is utilized, otherwise it can be overridden.

2. Checked and unchecked exceptions

Checkedexceptions	Unchecked exceptions
Checked exception is handled during compile time and it gives the compilation error if it is not caught and handled during compile time	In case of the unchecked exception, a compiler does not mandate to handle. The compiler ignores during compile time.
Example: FileNotFoundException, IOException etc.	Example: ArrayIndexoutOfBoundException

3. Difference Between List And Set

List	Set
1) inseration order is maintained	1)inseration order is not maintained
2)duplicate elements are allowed	2)duplicateelements are not allowed
3)contains classes	3)contains classes as well as interface
4) null insertion is allowed	4) null insertion is allowed only once
4) 3 main classes in List	4)Classes and Interface
i)ArrayList	i)Hashset (Class)
ii)LinkedList	a)LinkedHashSet(Class)
iii)Vector	ii)Navigable Set (Interface)
	a)Sorted set (Interface)

b)Tree Set

4. Difference Between ArrayList And LinkedList

ArrayList	LinkedList
1)Datastructure : growable array or resizable array	1)Datastructure : Doubly linked list
2)Array list implements Random Access interface so we can access random value with same speed.	2)LinkedList implements serializable interface and clonable interface but not random access interface.
3)if our frequent operation is inseration or deletion in the middle of arraylist then it worst choice beacuse it is require several shifting operation.	3)best choice for if our frequent operation is inseration and deletion . worst choice if our frequence operation is reterival.

5. Difference Between ArrayList and Vector

ArrayList	Vector
1)ArrayList is not synchronized	1)Vector is **synchronized**.
2) Array list and vector implements Random Access interface	2) vector implements Serializable, Clonable and Random Access interface.

Hash Map	**Hash Table**
No method is synchronized	method is synchronized
Multiple threads can operate at the time	At a time only one thread/ user is allowed
Threads are not required to wait and hence	Threads are required to wait and hence

performance is high	performance is low
Null is allowed for only once in key and multiple time s in value	Null is not allowed in key-value
Introduced in 1.2 version .	Introduced in 1.0 version and it is legacy class
It is not legacy classes	

HashMap	**LinkedHashMap**
Insertion order is not preserved	Insertion order is preserved
Underlying hash map is hash table	Underlying data structure is hashtable + linked list
1.2 version	1.4 version

Where we use Collection part in selenium

Selenium Interview Questions:

Selenium Questions:

1. Difference in driver.get() and driver.navigate().to()

get()	navigate().to()
get() method is used to open the specific url in browser.	**navigate().to()** method is used to open specific url and also to navigate from one url to another url.
It will wait till all the elements in webpage are loaded.	It will not wait till all the elements in webpage are loaded.
Syntax: *driverObjName.***get("***website url***");**	Syntax:*driverObjName***.navigate().to("***url***");**

2. Navigation methods in selenium

Navigation is interface in selenium.

Some methods are as follows:

- **navigate().to():**

  It is used to open a URL **or** to navigate from one URL to other URL.

  Syntax:

  *driverObjName***.navigate().to("***url***");** **------ Void**

- **navigate().back():**

  Used to navigate back in browser.

  Syntax:

  *driverObjName***.navigate().back ();** **------ Void**

- **navigate().forward():**

  Used to navigate forward in browser.

  Syntax:

  *driverObjName***.navigate().forward ();** **------ Void**

- **navigate().refresh():**

  Used to refresh the browser page.

  Syntax:

  driverObjName.**navigate().refresh ();   ------ Void**

3. Difference between Implicit wait and Explicit wait.

Implicit Wait	Explicit Wait
Implicit Wait time is applied to all the elements in the script / webpage.	Explicit Wait time is applied only to those elements which are intended by us.
In Implicit Wait, we need not specify "ExpectedConditions" on the element to be located.	In Explicit Wait, we need to specify "ExpectedConditions" on the element to be located.
It is recommended to use when the elements are located with the time frame specified in Selenium implicit wait.	It is recommended to use when the elements are taking long time to load and also for verifying the property of the element like(visibilityOfElementLocated, elementToBeClickable,elementToBeSelected).

4. Difference between getText() and getAttribute()

getText()	getAttribute()
getText() method is used to capture the visible text present on the particular element.	getAttribute() method is used to capture the inner HTML attribute value of that particular element.
We did not pass any parameter in this method.	In this method we need to pass the parameter as attribute key.

Syntax: *WebEleName*.**getText();**	Syntax: *WebEleName*.**getAttribute("***attribute Key***");**

5. How to extract css property value

We can extract Css value by using getCssValue() method.

We have to provide property name parameter.

Syntax:

WebEleName.**getCssValue("***Property name***");**

The **return** type of this method is **String**. So we can store it in variable and print in o/p.

6. Difference between driver.close() and driver.quit()

driver.close()	driver.quit()
close() method shall close the browser window which is in focus.	**quit()** method closes all the browser windows that are opened in current execution.
close() method closes the active WebDriver instance.	**quit()** method closes all the active WebDriver instances.
Syntax:*driverObjName*.**close();**	**Syntax:***driverObjName*.**quit();**

7. How to handle dropdown elements with select tag and without select tag.

We can handle drop down in selenium **with the help of Select class.**

First we have to **create object of Select class by passing drop down WebElement instance.**

Then by using that object we can handle the dropdown.

- **To select option** from drop down we can use three methods
  - **selectByIndex()** ------ insert **index integer**.
  - **selectByValue()** ------ insert **value in String**.
  - **selectByVisibleText()** ------ insert **visible text in String.**

- **To capture the selected option** we use **getFirstSelectedOption()** method the return type of this method is **WebElement.**

  Then by using **getText()** method we capture the selected option, the return type of this method is **String.** So we can store the captured option in String and then we can print it.

- **To count the options in drop down** we use **getOptions()** method the return type of this method is **List of WebElement**. i.e. **List<WebElement>**

  Then by using **size()** method we can count the options. The return type of this method is **integer**, so we can store the count number in int and then we can print the count of options present in drop down.

- **To print all options** in drop down we use **getOptions()** method the return type of this method is **List of WebElement**. i.e. **List<WebElement>**

  Then by using **Enhance for loop** or **for loop** we can print all options in dropdown.

To handle **dropdown without Select class** we can use **Actions class method click()** and also we can use **WebElement method click()**.

8. How to handle frames and windows?

Frames:

Frames are used to divide browser window i.e. webpage into multiple sections, where each sections can load a separate HTML page.

There are top frame and child frame present in a webpage.

Whenever we access that particular webpage thenthe focus is on top frame. We are on top frame then we have to switch to child frame.

We cannot switch child frame to child frame. First we have to switch to top frame then we can switch to other child frame.

We can **switch to child** frame by using three ways,

- **By using frame index id**

  Syntax:

  driverObjName.**switchTo().frame(***intindex***);**

- **By using frame name**

  Syntax:

  driverObjName.**switchTo().frame("***name***");**

- **By using frame WebElement**

  Syntax:

  driverObjName.**switchTo().frame("***WebElement Instance***");**

Return type of frame is **WebDriver**.

If we want to switch to top frame then we use defaultContent() method.

Return type of this method is **WebDriver**.

Syntax:

driverObjName.**switchTo().defaultContent();**

9. how to handle multiple Windows:

We can handle **tabs/ window** by using **getWindowHandle()** method.
Return type getWindowHandle() method is **String**.

If we want to handle **multiple tabs or window** then we have to use **getWindowHandles()** method.
Return type of getWindowHandles() method **Set<String>.**

Then by using if statement we can switch to another tab. This method is only helpful for 2 tabs.
For multiple tab handling we convert set of string into array list and then by using the index position we can switch to any tab / window.
(*then tell the syntax of if statement and array list as in methods.*)

10. How to switch back to parent window?

We can handle tabs/window by using getWindowHandle() method.
Return type getWindowHandle() method is String. So we store it in String variable and **by using driver.switchTo().window()** method and **passing the string variable** parameter we can switch to parent window.

In 2nd way we convert set of string into array list and then by using the index position we can switch to any tab / window. So for **parent** window the **index** is always **zero (0)**. So by **passing this zero index in driver.switchTo().window()**we can switch to the parent window.

11. How to perform page scroll in selenium

12. How to click or send value to an element without using element.click() and element.sendKeys()?

We can click or send value to an element by using Actions class methods.
Which are click() method, sendKeys() method, keyDown() method and keyUp() method.

In 2nd way by using JavaScript executor.

13. Difference between Absolute xpath and Relativexpath.

Absolute xpath	Relative xpath
Absolute xpath uses complete path from root to the desired element.	Relative xpath is a customized xpath or user created xpath.
Absolute xpath starts with "/" single forward slash.	Relative xpath Starts with "//" double forward slash.
Absolute xpath identifies element fastly than relative xpath.	Relative xpath can identify element slowly than Absolute xpath.
Absolute xpath have more chances of failure as any changes in any other element before that element can change the Absolute xpath.	The failure chances of a well written relative xpath is very less.
Eg. /html/head/body/form/table/tbody/tr/th	Eg. //table/tbody/tr/th

14. Explain few Xpath functions

Relative xpath functions

- Relative xpath with **single attribute**:

  Syntax:

  *//tagName***[@***Attribute***=** '*Value*'**]**

  tagName: present in html code. Starting text of path of that particular element.

  Attribute: it is the key value pair in html code. Like name, class, id, etc.

  Value: it is the value of specific attribute key.

- Relative xpath with **2 attribute:**

  Syntax:

  *//tagName***[@***Attribute1***=** '*Value1*'**][@***Attribute2***=** '*Value2*'**]**

  *//tagName***[@***Attribute1***=** '*Value1*' **and @***Attribute2***=** '*Value2*'**]** (*and operator*)

  *//tagName***[@***Attribute1***=** '*Value1*' **or @***Attribute2***=** '*Value2*'**]**(*or operator*)

- Relative xpath with **text() method:**

  Syntax:

  //*tagName* **[text()= '***Visible text value***']**

- Relative xpath with **contains() method with attribute:**

  Syntax:

  //*tagName* **[contains(@***Attribute***, '***Value***')]**

  //*tagName* **[contains(text() , '***Visible text value***')]**

- Relative xpath using **starts-with() method with attribute:**

  Syntax:

  //*tagName* **[starts-with(@***Attribute***, '***Value***')]**

  //*tagName* **[starts-with(text() , '***Visible text value***')]**

- Relative xpath using **following method:**

  Syntax:

  //*tagName***[@***Attribute***= '***Value***']//following ::** *tagName***[@***Attribute***= '***Value***']**

  //*tagName***[@***Attribute***= '***Value***']//following ::** *tagName***[***1***]**

- Relative xpath using **preceding method:**

  Syntax:

  //*tagName***[@***Attribute***= '***Value***']//preceding ::** *tagName***[@***Attribute***= '***Value***']**

  //*tagName***[@***Attribute***= '***Value***']//preceding ::** *tagName***[***1***]**

We can also use text() method in following and preceding method

15. Mention few selenium exceptions

There are several Exceptions in selenium such as;

NoSuchElementException:

This is commonly seen exception as a subclass of *NotFoundException* class.

The exception occurs when WebDriver is unable to find and locate elements.

NoSuchFrameException:

When WebDriver is trying to switch to an invalid frame, NoSuchFrameException under *NotFoundException* class is thrown.

NoSuchWindowException:

NoSuchWindowException comes under *NotFoundException* class. This is thrown when WebDriver tries to switch to an invalid window.

NoAlertPresentException

NoAlertPresentException under *NotFoundException* is thrown when WebDriver tries to switch to an alert, which is not available.

StaleElementReferenceException:

This exception says that a web element is no longer present in the web page.

TimeoutException:

The TimeoutException occurs when the command that is currently in execution and does not completed within the expected time frame.

ElementNotSelectableException:

This exception comes under *InvalidElementStateException* class. ElementNotSelectableException indicates that the web element is present in the web page but cannot be selected.

ElementNotVisibleException:

This exception comes under *InvalidElementStateException* class. ElementNotVisibleExceptionoccurs when WebElement is present but it not visible.

NoSuchAttributeException:

NoSuchAttributeException occurs when the attribute of the element could not be located.

16. Difference between single slash and double slash.

single slash	double slash
Single slash is used for absolute xpath.	Double slash is used for relative xpath.
A single slash '/' anywhere in xpath signifies to look for the element immediately inside its parent element.	A double slash '//' signifies to look for any child or any grand-child (descendant) element inside the parent element.

17. What are the scenarios that cannot be automated using selenium?

There are many things that cannot be done using Selenium WebDriver. Few of them which I can list down are as follows:

Bitmap comparison is not possible using Selenium WebDriver.

Automating Captcha is not possible using Selenium WebDriver.

We **cannot read bar code** using Selenium WebDriver.

We **cannot automate OTP** submissionusing Selenium WebDriver.

There are more things that cannot be automated using Selenium WebDriver.

18. How to perform right click on selenium

For performing right click operation we have to use Actions class method.

By using contextClick() method we can perform right click operation.

The return type of contextClick() method is Actions class.

This method is overloaded in Actions class.

First we have to create object of Actions class by passing the WebDriver instance.

Then by using this object and contextClick() method we can perform right click operation.

Syntax:

Actions *objAct* **= new Actions(***WebDriverinstance***);**

*objAct***.contextClick(***WebElementinstance***).build().perform();**

19. Action class and it's methods

By using Actions class we can perform keyboard and mouse events.

First we have to create an object of Actions class by passing the WebDriver instance.

Then by using this object and WebElement instance we can perform mouse and keyboard events.

Actions *objActName* **= new Actions(***WebDriverinstance i.e. driverName***);**

WebElement *wbName* **=***driverName***.findElement(By.***id***(""));**

Mouse Events:

- **click():**

  Clicks on the middle of the given element.

  Return type of click() method is Actions class.

  This method is overloaded in Actions class.

  Syntax:

  *objActName***.click(***wbName***).build().perform();**

- **doubleClick():**

  Double click on the middle of the given element.

  Return type of doubleClick() method is Actions class.

  This method is overloaded in Actions class.

  Syntax:

  *objActName***.doubleClick(***wbName***).build().perform();**

- **contextClick():**

  Right click on the middle of the given element.

  Return type of contextClick() method is Actions class.

  This method is overloaded in Actions class.

  Syntax:

  *objActName***.contextClick(***wbName***).build().perform();**

- **clickAndHold():**

  Clicks on the middle of the element and holds up to release.

  Return type of clickAndHold() method is Actions class.

  This method is overloaded in Actions class.

  Syntax:

  *objActName***.clickAndHold(***wbName target***).build().perform();**

- **dragAndDrop():**

  Drags the source element and drops at targeted element.

Return type of dragAndDrop() method is Actions class.

This method is overloaded in Actions class.

Syntax:

objActName.**dragAndDrop(***wbName source, wbName target***).build().perform();**

- **moveToElement():**

  Moves the mouse middle of the given element. Mouse over event.

  Return type of moveToElement() method is Actions class.This method is overloaded

  Syntax:

  objActName.**moveToElement(***wbName target***) .build().perform();**

- **release():**

  Release the pressed mouse button on at current mouse location.

  Return type of release() method is Actions class. This method is overloaded

  Syntax:

  objActName.**release().build().perform();**

Keyboard event:

- **keyDown():**

  It is used to press the given key.

  Return type of keyDown() method is Actions class.This method is overloaded.

  Syntax:

  objActName.**keyDown(Keys.***KEYNAME***) .build().perform();**

- **keyUp():**

  It is used to release the given key.

  Return type of keyUp() method is Actions class.This method is overloaded.

  Syntax:

  objActName.**keyUp(Keys.***KEYNAME***) .build().perform();**

- **sendKeys():**

  It is used to enter data in the active element.

  Return type of keyUp() method is Actions class.This method is overloaded.

  Syntax:

  objActName.**sendKeys("***Value to send***") .build().perform();**

20. Which faster driver in selenium

HTML UnitDriver is the most light weight and fastest implementation browser of WebDriver. It is based on HtmlUnit.

It is known as **Headless Browser Driver**.

It is same as Chrome, IE or Firefox driver but it does not have GUI so one cannot see the test execution on screen.

21. Can we create an object of the WebDriver() in selenium?

We cannot create an object of WebDriver because it is an Interface.

WebDriver, WebElement, Timeouts, Alert, Options.

22. What is StaleElementReferenceException and how to handle it?

This exception says that a web element is no longer present in the web page. This exception is not the same as ElementNotVisibleException. StaleElementReferenceException is thrown when an object for a particular web element was created in the program without any problem and however; this element is no longer present in the window. This can happen if there was a

- Navigation to another page
- DOM has refreshed
- A frame or window switch

Example:

```
WebElement firstName = driver.findElement(By.id("firstname"));
driver.switchTo().window(Child_Window);
element.sendKeys("Aaron");
```

In the code above, object firstName was created and then the window was switched. Then, WebDriver tries to type 'Aaron' in the form field.

In this case StaleElementReferenceException is thrown.

Avoiding and Handling:

Confirm that we are trying to do the action in the correct window. To avoid issues due to DOM refresh, we can use Dynamic Xpath.

Example:

Say 'id' of a username field is 'username_1' and the xpath will be //*[@id='firstname_1?].

When you open the page again the 'id' might change say to ''firstname _11'.

In this case, the test will fail because the WebDriver could not find the element.

and StaleElementReferenceException will be thrown.

In this case, we can use a dynamic xpath like,

```
    try
    {
driver.findElement(By.xpath("//*[contains(@id,firstname')]")).sendKeys("Aaron");
    }
    catch(StaleElementReferenceException e)
    {
        Catch statement
    }
```

In the example above dynamic XPATH is used and if the exception is still found, it is caught.

23. What is use WebDriver driver=new ChromeDriver();

By using WebDriver driver= new ChromeDriver() we actually open the blank chrome browser.

By giving reference of WebDriver we can call all implemented methods of WebDriver interface.

Here we achieve upcasting.

24. How to lunch headless script

We can launch headless browser / script with the help of ChromeOptions class.

First we have to create object of ChromeOptions class.

Then we use addArguments() method and under the argument we pass "--headless"

Then we pass the ChromeOptions object name in ChromeDriver constructor.

We write above script first.

Syntax:

System.setProperty("webdriver.chrome.driver","Path of chrome driver");

ChromeOptions*objName* **=new ChromeOptions();**

opt.addArguments("--headless");

WebDriver *driverName* **= new ChromeDriver(***objName***);**

25. What the different interface in selenium.

There are several interface present in the Selenium. Some of them are as follows,

SearchContext is the parent interface of all interface present in Selenium.

WebDriver,

WebElement,

OutputType

Navigation

Options

Timeouts

Window

TargetLocator

TakesScreenshot

26. Difference between findElement() and findElements()

findElement()	findElements()
findElement() is used to find specific one element in webpage.	findElements() is used to find elements having similar tag or locator in webpage.
The return type of findElement() is **WebElement**.	The return type of findElements() is List of WebElement. i.e. **List<WebElement>**
Throws exception NoSuchElementException if there are no elements matching the locator strategy.	Returns an empty list if there are no web elements matching the locator strategy.

27. Difference between build() and perform()

build()	perform()
build() method is used to combine multiple actions in single statement.	**perform()** method is used to execute each and every statement / action.
return type of build() method is **Action class**	return type of perform() method is **void**

28. What is Locator and its type?

Locator is an address that identifies a web element uniquely within the web page.

Locator are the HTML properties of element.

Element locator are common for all the browsers.

Selenium web driver use 8 different locator

id

name

className

tagName

linkText

partialLinkText

cssSelector

xpath

29. WebDriver interface method

abstract method

get()	void
getTitle	String
getCurrentUrl	String
getPageSource()	String
close()	void
quit()	void

Navigation interface it is sub interface of web driver

navigate().to()	void
navigate().back()	void
navigate().forward()	void
navigate().refresh()	void

Window class:

maximize()	void
minimize()	void
setSize()	void

Dimension class

cons with 2 int arument/parameterwidth and height

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

findElement() WebElement ---- method overloading

30. WebElement interface method

isDisplayed()	boolean
isEnabled()	boolean
isSelected()	boolean
click()	void
sendKeys()	void
getText()	String
getAttribute()	String
clear()	void

31. Count how many link in web page?

Link on any web page are given anchor tag name i.e. **“a”**.

By using **tag name** locator as **“a”**and **findElements()** method we can find the total links on webpage the return type of findElements() method is **list of WebElement.**

Then by using **size()** method we can get the count of links, the return type of **size()** method is **integer**. So we can store it in integer and print it in output console.

Then by using enhance for loop we can print all links in output.

```
List<WebElement>ls=driver.findElements(By.tagName("a"));
inta=ls.size();
```

32. How to handle Alert popup in selenium?

We can handle alert popup in selenium with the help of Alert interface.

First we have to switch focus from main window to the alert popup so we use,

driver.**switchTo().alert();** method, return type of this method is Alert interface.

Syntax:

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

In Alert interface there are 4 methods are available.

1. accept():

When we want to **click on ok** button on alert popup we use **accept()** method.

Return type of accept() is void.

Syntax:

altName.**accept();**

2. dismiss():

When we want to **click on cancel** button on popup we use **dismiss()** method.

Return type of dismiss () is void.

Syntax:

altName.**dismiss();**

3. getText():

If we want to **capture visible text**present onpopup we use **getText()** method.

Return type of getText() is String.

Syntax:

altName.**getText();**

4. sendKeys():

If we want to **enter any data** in popup text box we use **sendKeys()** method.

Return type of sendKeys() is Void.

Syntax: *altName*.**sendKeys ();**

33. How to handle Static table?

Static tables are present in web page.

We perform following operations on static table;

1) Capture the value:

To capture the particular value we can use **findElement()** method with **getText()** method, return type of getText() is String, so we can store it in String and we can print it in output console.

Syntax:

String*a*=*driver*.**findElement(By.***xpath*(**"***//td[text()='Amit']***"**)).getText();**

For following methods first we use **findElement()** method **to find the particular table**, the return type of findElement() is WebElement. So we store it in WebElement instance and then we use that instance.

Syntax:

WebElement *wbTable*=*driver*.**findElement(By.***id*(**"***Employee***"**));**

2) Count rows in table

To count total no of rows in that particular table, first we use **findElements()** method and ***tagNamelocator*** generally the **tag name for rows is "tr"**, the return type of findElements() method is List of WebElement. Then we use **size()** method the return type of size() is integer so we can store it in integer and print it in output console.

Syntax:

```
List<WebElement>lsRow =wbTable.findElements(By.tagName("tr"));
introwCount =lsRow.size();
System.out.println(rowCount);
```

3) Count column in table:

To count total no of column in that particular table, first we use **findElements()** method and ***tagNamelocator*** generally the **tag name for column is "th"**, the return type of findElements() method is List of WebElement. Then we use **size()** method the return type of size() is integer so we can store it in integer and print it in output console.

Syntax:

```
List<WebElement> lsCol =wbTable.findElements(By.tagName("th"));
intcolCount =lsCol.size();
System.out.println(colCount);
```

4) Count records in table

To count total no of records in that particular table, first we use **findElements()** method and ***tagNamelocator*** generally the **tag name for records is "td"**, the return type of findElements() method is List of WebElement. Then we use **size()** method the return type of size() is integer so we can store it in integer and print it in output console.

Syntax:

```
List<WebElement>lsData =wbTable.findElements(By.tagName("th"));
intdataCount =lsData.size();
System.out.println(dataCount);
```

34. What is Synchronization?

It is process of matching two or more activities/process in time.

during the test execution test tool give instruction one by one with same speed but AUT takes less time forsome steps execution and more time for some step execution in order to keep them in sync then Synchronizationis required.

Types of Synchronization

1) Unconditional Synchronization:

In Unconditional Synchronization we specify timeout value, we will make tool to wait certain amount of time and then process.

When we give unconditional sync then we forcefully pause the execution process for given time. After time completion the process resumes.

Ex. **Thread.sleep(***timeinmiliseconds***);**

2) Conditional Synchronization:

Conditional synchronization does not used for all commands/ statement in the program.

It works only for **findElement()** and **findElements()** method only.

There are two types in conditional sync

i) ImplicitlyWait:

The implicitlyWait in selenium WebDriver is used to tell web driver to wait certain amount of timebefore throws a**NoSuchElementException**.

Once we set the time thenWebDriver will wait for element before throwing the exception.

It also known as global wait, it is applicable for all the element in web page

Return type of implicit wait is TimeOuts interface

Method overloading

Syntax:

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

ii) ExplictWait:

The explicit wait in selenium is used to tell web driver to wait certain condition (ExpectedConditions) or maximum time exceed before throwing an exception.

It is apply for single element/ object in webpage.

Once we declare explicit wait we have use expectedConditions .

Method overloading.

First we have create object of WebDriverWait by passing WebDriver instance and time in seconds

Syntax:

WebDriverWait*wait* **=new WebDriverwait(***driver*,*30***);**

```
wait.until(ExpectedCondition.visibilityOfElement(By.id("")));
```

Return type of until is WebElement.

3) Fluent wait:

It is used to change the repetition cycle of finding element.

By default the cycle is repeated in 255 milliseconds

syntax:

```
Wait wait =new FluentWait(driver)
        .withTimeout(Duration.ofSeconds(20))
        .pollingEvery(Duration.ofSeconds(2))
        .ignoring(Exception.class)
```

1)text box operation:

i)check displayed status	isDisplayed();
ii)check enabled status	isEnabled();
iii)enter the value in text box	sendKeys("data");
iv)capture the entered value in text box	getAttribute('value');
v)clear the value in text box	clear();

2)operation on button

i)check displayed status	isDisplayed();
ii)check enabled status	isEnabled();
iii)return the button value	getAttribute('locator');
iv)click on button	click();

3)operation on link

i)check displayed status	isDisplayed();
ii)check enabled status	isEnabled();
iii)capture the link name	getText();
iv)click on link	click();

4) operation on radio button

1) checkfemale displayed status	true	isDisplayed()	boolean
2)check female enabled status	true	isEnabled()	boolean
3) check female selected status	false	isSelected()	boolean
4) click female		click()	void
5) check female selected status	true	isSelected()	boolean
6) click on male radio button		click()	void
5) check female selected status	false	isSelected()	boolean

5) operation on drop down

>check displayed status of dp

>check enabled status

> select drop down value

>count drop down value

> check selected value

> check specific value is present in drop down or not?

6) operation on images:

there are 3 types of image

1) general image

>check displayed

2) Image button

>>check displayed

> enabled status

> click

> capture the image button value

3) Image link

> displayed status

>enabled status

> click on image link

7) capture error test area or error message

> return the error message	-- getText()	String

8) Operation on table

> get cell count

> Row count

> get cell value

> check specific value is present in cell

table	table
table heading	th
row	tr
table data	td

9) capture error test area or error message

> return the error message --getText() String

10) Operations on browser:

1) open a browser

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

2) open a url

driverObjName.**get(“***website url***”);**

3) navigate from one url to another url

driverObjName.**navigate().to(“***url***”);**

4) navigate back

driverObjName.**navigate().back ();**

5) navigate forward

driverObjName.**navigate().forward ();**

6) refresh the browser

driverObjName.**navigate().refresh ();**

7) minimize

driverObjName.**manage().window().minimize();**

8) maximize

driverObjName.**manage().window().maximize();**

9) Full screen

driverObjName.**manage().window().fullScreen();**

10) Set size or to change the size of browser

driverObjName.**manage().window().setSize(Dimension** *targetSize***);**

11) close current window

driverObjName.**close();**

12) close all the tab

driverObjName.**quit();**

Method overloaded in selenium

- page load time out
- implicit wait
- frame
- findELement
- find Elements
- getScreenshotAs
- keyDown in actions class
- sendKeys in actions class

Constructors

- actions
- select
- dimension

TestNG: (Next Generation)

TestNG is an open source tesing framework where NG stands for Next Generation, It is desgined to simplify the broad range of testing needs from unit testing (developer) to system testing. TestNG is inspired from junit (java platform) and Nunit (c# platform) but it has its own features and functionality that makes it a more powerful.Using testNG we can create, group, prioritize and execute the test cases, and we can also generate the test reports.

What are the **advantages** of TestNG?

- It is an open-source framework, hence it is easy to use and it is easy to configure.
- Using TestNG we can systematically create the test cases.
- It gives lots of annotations which in turn makes the test case creation easy.
- Using TestNG, priorities of the tests and the execution sequence can be defined.
- Grouping is possible using TestNG.
- It generates HTML reports (Selenium Webdriver cannot generate the test reports alone, it helps SW to achieve this).
- Data parameterization is possible using TestNG.
- Parallel execution and cross browser execution is possible using TestNG.
- Readily supports integration with tools like eclipse, maven etc.
- Supports data driven testing.

Note:

- Using testNG we can create test cases, group test cases, prioritize test cases, execute the test case and generate the report.
- **Main method is not used** in testNG program
- TestNG **only executes the methods that contains@test annotation**if we don't write **@test** annotation then method are not going to be executed.
- TestNG test **cases are executed in alphabetical** order. If u want **maintain execution flow then we use priority** attribute. We give priority to the test cases.

- **By default priority** for test case **is zero**.
- We can **write negative priority** to test cases.
- We **can write same priority** to multiple test cases.

Some Methods and Attributes in TestNG:

➢ How to **run** the test method?

We **run the test** method by giving **@Test** annotation above the method block.

Syntax:

@Test

Method {body}

➢ How to give **priority** to test case?

We give **priority** to test cases by passing **priority=number** attribute.

Syntax:

@Test(priority=*Number***)**

➢ To **skip the execution** of the test case.

We **can skip specific test case** in TestNG with the help of **enabled=false** attribute**.**

Syntax:

@Test(enabled=false)

➢ **dependsOnMethods**Attribute:

It is used for execution sequence.

When depends on method executes with pass status then and only then it executes the dependent method.

If the depends on method executes with fail status then it will skip the execution of the dependent method.

It is also called as hard dependency.

Syntax:

@Test(dependsOnMethods={"*methodName***"})**

- **alwaysRun** Attribute**:**

  It is always used with **dependsOnMethod**

  It is used to execute the test whether the dependsOnMethod pass or fail.

  It is also called as soft dependency.

  Syntax:

  @Test(dependsOnMethods= {"*methodName***"},alwaysRun=true)**

- **How to execute same test cases multiple times?**

  We execute same test cases multiple times with the help of **invocationCount** attribute

  Syntax:

  @Test(invocationCount= *Number***)**

Questions

1. Annotations in testing

There are 3 sections of annotation

1) Pre-condition annotation

2) Test annotation

3) Post-condition annotation.

i) Pre-condition annotation:

These are the annotation that are **executed before the test**.

@BeforeSuite

@BeforeTest

It is pre-condition for **all the classes in xml File**.

@BeforeClass

It is pre-condition **for all test cases** in the particular class

@BeforeMethod

It is pre-condition **for each and every test case** in the particular class

ii) Test annotation:

This is the annotation which is only **mentioned before the test cases**.

@Test

iii) Post condition annotation:

These are the annotation that are **executed after the test cases.**

@AfterMethod

It is post condition **for each and every test cases** in the particular class

@AfterClass

It is post condition **for all test cases** in the particular class

@AfterTest

It is post condition for **all the classes in xml file.**

@AfterSuite

2. Difference between @BeforeClass and @BeforeMethod

@BeforeMethod- Pre-condition for every Test case in a Class/Program

@AfterMethod - Post-condition for every Test case in a Class/Program

@BeforeClass - Pre-condition for All Test cases in a Class/Program

@AfterClass - Post-condition for All Test cases in a Class/Program

@BeforeTest:- It is is pre condition for all the classes in xml File

@AfterTest- It is post condition for all the classes in xml file

3. Soft assert VS hard assert

or

diff between verify and assert

Hard Assert:

hard assert throws an exception an immdiately when an assert statement fail and test suite go to the next test cases. If there is any code in current test case after assert statement then it will not execute that code/ statement.

Syntax:- Assert.assertEquals("Actula result","expected Result");

Soft Assert:

> soft assert is a customized error handle provided by testNG. It collects errors during execution.

> soft assert does not throws an exception when assert statement fail and would continue with next step after assert statement.

> If we want throws exception then we have to use **assertAll()** method

First we have to create object of soft assert then we have to use assertAll method after assert statement to throw an exception.

SoftAssert soft=new SoftAssert();

soft.assertEquals("actual result","expected Result");

soft.assertAll();

4. How to pass parameters to test methods?

There are **two ways** by which we can achieve parameterization in TestNG.

1) With the help of **Parameters annotation** and **TestNG XML** file.

2) With the help of **DataProvider** annotation.

5. Use of DataProvider Annotation.

The data driven concept is achieved by **@DataProvider** annotation in TestNG.

With the help of DataProvider annotation we pass the parameters to test methods.

It has only one **attribute 'name'**. If you have to specify the name attribute.

The Data Provider's name will be same as the corresponding method name.

Syntax:

```
@Test(dataProvider="login")
public voidcheckLogin(Stringusername,Stringpassword)
{
        Statement.(username);
        Statement.( password);
}

@DataProvider(name="login")
public Object[][]getData()
{
        Object [] []data=new Object [3][2];
        returndata;
}
```

6. Use of priority, dependsOnMethod, dependsOnGroups, alwaysRun, groups.

Priority:

We give priority to test cases to maintain the correct execution flow of the test cases.

If we do not provide priority to the test cases then TestNG will execute the test cases on the basis of alphabetical order, which is not suitable for test execution sequence.

By giving priority we give sequence of execution to the test cases.

dependsOnMethods:

If we want to execute the test of particular method only after the pass test of another method then we use dependsOnMethods attribute.

So it will execute that test only if the other method pass the test otherwise it skips that particular test.

It is also called as hard dependency.

alwaysRun:

It is always used with dependsOnMethod**.**

It is used to execute the test whether the dependsOnMethod pass or fail.

It is also called as soft dependency.

groups:

It is used for group the test cases, so the test cases in one particular group can be executed at a time.

By using groups attribute it is very easy to perform execution of some test cases which we want to execute rather than executing the whole test cases.

dependsOnGroups

If we want to execute the test of particular method after the test of any particular group we use dependsOnGroups attribute.

7. Can we mention negative priority

- **Yes**, we can mention negative priority to test cases.

8. What the by default priority for test in TestNG

- The default priority for test in TestNG is **zero (0).**

9. TestNG xml structure

In TestNG xml structure starts with suite tag,then test tag,then classes tag, under classes tag there is class or multiple classes are present,

Under the class one or more methods are present

We can change the suit name and test name according to the project by using xml structure.

The xml structure is as follows:

```
<suite name="project name">
<test name="regression testing">
<classes>
        <class name="packagename.classname" />
</classes>
</test>
</suite>
```

10. How to group test cases

We can create group of test cases by using **groups** attribute.

Syntax:

@Test(groups = {"*groupname*"})

We have to provide the group name for each test case of that particular group.

When we call that particular group, the methods having same group name are called only. So we need to be careful while giving group name.

We call the group test cases by using xml file structure.

We have to **add <groups> tag after <test>** tag.

We have to **close <groups> tag just before <classes>**tag.

Under <groups>tag we write **<run>** tag.

Under run tag we write **<includes name= "*groupname*"/>** tag and we close include tag the run tab.

We can **use excludes** tag to execute all test cases **excluding the given group**.

Structure:

```
<suite name="projectname">
<test name="testcasename">
        <groups>
                <run>
                        <include name="regression"/>                    --(can write exclude here)
                </run>
        </groups>
<classes>
        <class name="packagename.classname"/>
        <(we can write here multiple classes to run the test as above syntax)>
</classes>
</test>
</suite>
```

11. Use of IRetryAnalyzer class

We execute fail test cases using IRetryAnalyzer interface

It contains only method we override this method.

Syntax:

```
public class ExecuteFailTestCaseAgain implements IRetryAnalyzer{

    intcounter=0;
    intmaxRetry=4;

    @Override
    public boolean retry(ITestResult result) {

        if(counter<maxRetry)
        {
            counter++;
            return true;
        }

        return false;
    }

}
```

12. How to run failed test cases

There are 2 ways we can execute fail test cases again and again

1) By using IRetryAnalyzer interface.
2) By using IAnnotationTransformer interface.
3) XML file

13. How to perform parallel testing

We can perform parallel testing by two ways.

1) By passing parallel = "classes"in xml file and multiple classes:

So we have to pass parallel = "classes" in suit tag in xml file.

Then we have to create multiple classes with same tests but different browser names.

And then we have to pass these multiple classes in xml file under the classes tag.

And then we execute the test cases from xml then the execution of all classes under the classes tag takes place parallel.

```
<suite name="Suite" parallel="classes">
    <test thread-count="5" name="Test">
        <classes>
            <class name="Tutorial9.EdgeBrowser"/>
            <class name="Tutorial9.ChromeBrowser"/>
        </classes>
```

2) By passing parameters:

So first we pass parallel = “classes” in suit tag in xml file.

Then under test tag we write parameter tag and we write the name and value of in this tag.And then classes and class tag.

We write the code from test tag open to test tag close multiple times i.e. as per our requirement for parallel test, with different parameter values.

Then in actual test script we provide Parameter annotation and give the name and we write some else if block which can open different browsers in the script.

Then we execute the test from xml file.

Then it will execute the parallel test cases as per the value of parameter and else if block condition.

Syntax:

```
<suite name="Suite" parallel="tests">
    <test thread-count="5" name="Test in chrome browser">
        <parameter name="browserType" value="chrome" />
        <classes>
            <class name="Tutorial9.ParallelDemo" />
        </classes>
    </test><!-- Test -->

    <test thread-count="5" name="Test in edge browser">
        <parameter name="browserType" value="edge" />
        <classes>
            <class name="Tutorial9.ParallelDemo" />
        </classes>
    </test><!-- Test -->
</suite><!-- Suite -->
```

```
@Parameters("browserType")
@BeforeClass
public void setUp(String browserType) {

        if(browserType.equals("chrome"))
        {
        WebDriverManager.chromedriver().setup();
        driver =new ChromeDriver();
        }
        else if(browserType.equals("edge"))
        {
                WebDriverManager.edgedriver().setup();
                driver=new EdgeDriver();
        }
```

14. What is execution sequence of testNG annotation?

The execution sequence is as like,

@BeforeSuite

@BeforeTest

@BeforeClass

@BeforeMethod

@Test

@AfterMethod

@AfterClass

@AfterTest

@AfterSuite

15. Can we write same priority for multiple test cases?

- **Yes**, we can mention same priority to multiple test cases. Then the test case having same priority are executed in alphabetical order.

16. How to create ExcelReader class?

First we create ExcelReader class.

Under this class we create

- **Constructor with 1 String argument:**

  By using this we **check the file** by using **File Class**

  Then we **load the file** in class by using **FileInputStream**.

  Then we **load the workbook** using **XSSFWorkbook** (xlsx) or **HSSFWorkbook** (xls).

- **Non static method with String return type and 3 int arguments:**

  Here 3 arguments are

  Here we first **load the sheet**by using **getSheetAt(***sheetindex***)** method.

  Then we use **getRow(***rowindex***).getCell(***cellindex***).getStringCellValue()** method to capture the particular cell data.

- **Non static method with int return type and 1 int argument.**

  Here we first load the sheet by using **getSheetAt()** method.

  Then we use **getLastRowNum(***sheetindex***)** method to count the total rows in that particular row. It gives w.r.t. the index position i.e. last row index so we add 1 in that number to get actual present total rows.

- Non static method with int return type and 1 int argument.

  Here we first load the sheet by using **getSheetAt()** method.

  Then we use **getRow(***0***).getLastCellNum(***sheetindex***)** method to count the columns present in that particular sheet.

Actual Syntax of ExcelReader class:

```
publicclass ExcelReader extends BaseClass
{

	private FileInputStream fis;
	private XSSFWorkbook workbook;
	private XSSFSheet sheet;

	public ExcelReader(String path)			------	(Constructor with 1 String argument)
	{
		File f= new File (path);
		try
		{
			fis = new FileInputStream(f);
			workbook= new XSSFWorkbook(fis);
		}
		catch (Exception e)
		{
			e.printStackTrace();
		}

	}

	public String getData(intsheetIndex, introwIndex, intcolumnIndex)
	{					(Non static method with String return type and 3 int arguments)
	sheet=workbook.getSheetAt(sheetIndex);
	String data=sheet.getRow(rowIndex).getCell(columnIndex).getStringCellValue();
	returndata;
	}

	publicint getRowCount(intsheetIndex)
	{					(Non static method with int return type and 1 int arguments)
		sheet=workbook.getSheetAt(sheetIndex);
```

```
        introwCount=sheet.getLastRowNum();
        rowCount=rowCount+1;
        returnrowCount;
    }

    publicint getColumnCount(intsheetIndex)
    {                                  (Non static method with int return type and 1 int arguments)
        sheet=workbook.getSheetAt(sheetIndex);
        intcolumnCount= sheet.getRow(0).getLastCellNum();
        returncolumnCount;
    }


}
```

17. Use of ISuiteListener Methods

The ISuiteListener works on the suite level.

Additionally, it listens to the event of the start of a suite execution and end of the suite execution.

ISuiteListener then runs the methods only before the start of the suite and at the end.

It contains two methods:

1) **onStart:** This method invokes before the test suite execution starts.

Syntax: void onStart(ISuite suite);

2) **onFinish**: This method invokes after the test suite execution ends.

Syntax: void onFinish(ISuite suite);

Data driven framework(data driven testing):

- We achieve data driven testing by using **@DataProvider annotation**.
- Basically under data driven testing we have to read the data from excel sheet
- If we want to work with excel sheet then **we need Apache POI dependency**.
- Basically in my hybrid framework under the utility/Utils package we have create ExcelReader class to read the data from excel sheet. It is reusable class.

Under the class:

1)First we create constructor with 1 string argument

First we have to check file is present or not? We check file with the help of File class by passing file path.

Syntax: **File** *f* **= new File("***path of file***");**

File class is present in java.io package

2)Then we have to load file by using FileInputStream class by passing file class object name

Syntax:**FileInputStream***fis* **=new FileInputStream(***f***);**

3)Then we have to load workbook

Basicallythere are 2 types of excel sheet

1).xlsx

2) .xls

If we want to work **with .xlsx** then we have to create object of **XSSFWorkbook.**

We create object of XSSFWorkbook by passing FileInputStream instance

Syntax:**XSSFWorkbook** *workbook* **=new XSSFWorkbook(***fis***);**

If we want to work **with .xls** then we have to create object of **HSSFWorkbook.**

We create object of HSSFWorkbook by passing FileInputStream instance

Syntax:**HSSFWorkbook** *workbook* **=new HSSFWorkbook(***fis***);**

4) Then we focus on specific sheet and we read individual data from excel sheet.

We have created method like

Non static method with String return and 3 int argument

We have to focus specific sheet

There are 2 ways we can focus on sheet

1) By using sheet index

2) By using sheet name

Syntax:*workbook***.getSheetAt(***Sheet index***)**;

Return type of getSheetAt() method is XSSFSheet.

We have to read individual data by using XSSFSheet instance name.

We have to pass sheet row number and sheet cell number then we have to capture sheet data.

Syntax:*sheet***.getRow(***row index***).getCell(***column index***).getStringCellValue()**

Return type of getStringCellValue() is string.

Then again we are creating one more method to count how many rows in sheet

Then we have to **getLastRowNum();**

Return type of this method is int.

Then again we are creating one more method to count how many column in sheet

Then we have to **getRow(0).getLastCellNum();**

Return type of this method is int.

Suppose I give you 2000 test cases and you want to execute these test cases within two days, and only you can execute these test cases. How will you do that?

>>> Executing 1000 test cases a day is a daunting task for any tester. Firstly, I assume, the tester is nearing the end of testing. The deadline is reached. Conventional testing states that test cases should be segregated based on Severity and Priority. Business severity test cases should be executed first and then the priority cases. Also, automating the test cases using regression testing will help a lot and without which it could be practically impractical to complete executing 1000 test cases a day.

>>> 1. You can prioritize the test cases based on their severity and priority.

2. By this time you will be having some idea like if you execute one particular test case it may cover most of the business flows. Identify those test cases and execute.

3. Think from customer point of view like what is more important for client

4. Discuss with the developer like which are the areas where there will be possibility of system crashing or error because they will be knowing each and every part of the code well. Also based on the knowledge like which are the modules in which they have done last minute changes or frequent changes or the areas in which the requirement was not clearor developed in a hurry. So based on these factors you can decide.