**Java 8:**

**3 - What is Object-Level Locking and Class-Level Locking**

**Object Level Locking:**

Every object is Java has a unique lock. Whenever we use synchronized keyword, then only lock concept comes into the picture. Whenever a Thread wants to execute a synchronized method of object, it has to get the lock of an object. Once it’s get the lock of an object then it can execute any synchronized method of the object. Below are the examples:

**Example1 (Synchronized Method):**

public void synchronized method1() {  
 //Synchronized method  
}

**Example2 (Synchronized Block):**

Public void method1()  
{  
 synchronized(this) //here we are using this keyword to make a object level lock  
 {  
 System.out.println(Thread.currentThread().getName());  
 }  
}

**Class Level Locking:**

If a Thread wants to execute a static synchronized method, then the thread requires a Class level lock. Once the Thread gets the Class level lock, then it can execute any static synchronized method of the class. Below are the examples:

**Example1 (static Synchronized Method):**

public static void synchronized method2() {  
 //Synchronized method  
}

**Example2 (Synchronized Block):**

Public void method2()  
{  
 synchronized (ClassName.class) //here we using ClassName.classkeyword to make a class level lock  
 {  
 System.out.println(Thread.currentThread().getName());  
 }  
}

**5 - What is double-checked locking:**

**class** Singleton {

**private** **static** **volatile** Singleton *\_instance* = **null**;

**private** Singleton() { }

**public** **static** Singleton getInstance()

{

**if**(*\_instance* == **null**) // 1st Check

{

**synchronized**(Singleton.**class**)

{

**if**(*\_instance* == **null**) //2nd Check

{

*\_instance* = **new** Singleton();

}

}

}

**return** *\_instance*;

}

}

The purpose of Double checked locking was to avoid excessive synchronization and hence it relies on non-synchronized access of \_instance field at the 1st check point. But it is harmful.

Suppose one thread, **Thread-1** is inside synchronized block and it's creating Singleton instance and assigning reference to \_instance variable. In the mean time, Thread scheduler stops the Thread-1. Now, a second thread, **Thread-2** enters and come to 1st check point which is not synchronized, now there is a possibility that it can see *half-initialized* \_instnace field and return that to the client, leading to subtle bugs in your program.  
  
This issue was fixed by introducing of **volatile** variable in Java 1.5. According to this rule, write to a volatile field will happen before any read, which stops the possibility of seeing half initialized instance of Singleton class.

**4. Java 8 related changes for Interface:**

Prior to java 8, [interface in java](https://beginnersbook.com/2013/05/java-interface/) can only have abstract methods. All the methods of interfaces are public & abstract by default. From Java 8 we can define default and static methods in an interface. Classes implementing such Interface can override its default methods (If Class implementing multiple interfaces and those have the same default methods with the same signature then it creates ambiguity in the class as to which of the method actually needs to call when the methods has been called, to solved this ambiguity we need to implement that common method inside the class. ). Static methods we can not override in the Class who is implanting the Interface having the static method.

interface MyInterface{

default void newMethod(){   
System.out.println("Newly added default method");   
 }

static void anotherNewMethod(){  
 System.out.println("Newly added static method");  
 }

void existingMethod(String str);   
}

public class MyClass implements MyInterface{   
 public void existingMethod(String str){   
System.out.println("String is: "+str);

}

public static void main(String args[]) {   
 MyClass c = new MyClass();

//calling the default method of interface  
c.newMethod();

//calling the static method of interface  
MyInterface.anotherNewMethod();

//calling the abstract method of interface  
c.existingMethod("This is an existing method");

}   
}

**6 - What are the disadvantages of normal singleton design pattern implementation and ways to overcome the same**

**Singleton Design Pattern:**

To create singleton class, we need static member of class, private constructor and static factory method.

* **Static member:** It gets memory only once because of static, itcontains the instance of the Singleton class.
* **Private constructor:** It will prevent to instantiate the Singleton class from outside the class.
* **Static factory method:** This provides the global point of access to the Singleton object and returns the instance to the caller.

class Singleton{

private static Singleton obj=new Singleton (); //Early, instance will be created at load time

private Singleton (){}

public static Singleton getSingleton(){

return obj;

}   
}

Lazy intantiation can be achieved using synchronization block in the method.

class Singleton {   
 private static Singleton obj;   
 private Singleton(){}

public static Singleton getSingleton(){

if (obj == null){  
 synchronized(Singleton.class){   
 if (obj == null){   
obj = new Singleton();//instance will be created at request time (Lazy intantiation)

}   
}

}

return obj;   
 }   
}

## **Disadvantage of Normal Singleton Design Pattern and ways to overcome it:**

1. Since it’s been created at global level and we are avoiding it passing around the application. It makes code smell (means the fundamental of design principles, code quality, kind of weaknesses of the design that may showing slowing down the development and application might be at risk of bugs or failures.)
2. They violate the single responsibility principle (Every class or module have the responsibility of single part of the functionality provided by the software. And those responsibilities should be entirely encapsulated into the class and it provide the services for what it is responsible for.)
3. They create or makes the application more tightly coupled. An ideal application is always should be loosely coupled, where in all independent modules should interact each other seamlessly during future development of the application.

Using Reflection, Serialization and Cloning still we can create another instance of the Singleton class. Below is the way how can we create another instance of Singleton class and how to overcome this issue.

Using Reflection we can still create another instance of singleton class

import java.lang.reflect.Constructor;

Constructor[] constructors =

Singleton.class.getDeclaredConstructors();

for (Constructor constructor : constructors)

{

// Below code will destroy the singleton pattern

constructor.setAccessible(true);

instance2 = (Singleton) constructor.newInstance();

break;

}

If we check the hascode of the instances e.g. instance2.hashCode() and earlier instance1.hasCode both will be different, hence the another instance is created for singleton class. To overcome this, we can use Enum because java ensures internally that enum value is instantiated only once. Since java Enums are globally accessible, they can be used for singletons.

e.g.

//Java program for Enum type singleton

public enum GFG

{

INSTANCE;

}

Using Serialization we can still create another instance of singleton class

Singleton instance1 = Singleton.instance;

ObjectOutput out = new ObjectOutputStream(new FileOutputStream("file.text"));

out.writeObject(instance1);

out.close();

// deserailize from file to object

ObjectInput in

= new ObjectInputStream(new FileInputStream("file.text"));

Singleton instance2 = (Singleton) in.readObject();

To overcome this issue, we have to implement readResolve() method in the Singleton class as below:

class Singleton implements Serializable

{

// public instance initialized when loading the class

public static Singleton instance = new Singleton();

private Singleton()

{

// private constructor

}

// implement readResolve method

protected Object readResolve()

{

return instance;

}

}

When the object is read, it's going to replace the desterilized instance with the existing instance of Singleton class using readResolved() method.

Using cloning also another object of the Singleton can be created and to overcome this issue we have to override the clone() method inside the Singleton class and explicitly thro an exception there as below:

@Override

  protected Object clone() throws CloneNotSupportedException

  {

    throw new CloneNotSupportedException();

  }

**Selenium:**

**9. Is it possible to automate captcha and bar-code? Please explain?**

Reading Captcha and Barcode using any Automation Tool or Software is an illegal thing on the Internet.  
The main purpose of using Captcha is to identify the user should be human and not the automated script, in order to restrict the automated user login or accessing the application features over an internet.

Way to Solve:

Using API of external services (Death by Captcha) we still can solve the CAPTCH (). CAPTCHA as an acronym for "Completely Automated Public Turing test to tell Computers and Humans Apart"

With Death by Captcha you can solve any CAPTCHA. All you need to do is implement their API and pass then our CAPTCHAs and then will return the text.

**2 - How to wait and check for presence of an element on a page, every second(give code)? For Eg.FluentWait**

ExpectedCondition e = new ExpectedCondition<Boolean>() {

public Boolean apply(WebDriver driver) {

return driver.findElements(By.cssSelector("div[id\*='div']")).size() >0;

}

};

WebDriverWait wait = new FluentWait<WebDriver>(driver)

.withTimeout(15, TimeUnit.SECONDS)

.pollingEvery(1, TimeUnit.SECONDS)

.ignoring(NoSuchElementException.class);

wait.until(e);

**6 - How to create a new fireFox profile and set different capabilities for the same?**

**12 - Creating the firefox profile by setting up the capabilities to set the download folder (with some properties to be set for the acceptance of security certificate)**

**DesiredCapabilities Class and its use:**

* It is a class in org.openqa.selenium.remote.DesiredCapabilities package.
* It gives facility to set the properties of browser. Such as to set BrowserName, Platform, Version of Browser.
* Mostly DesiredCapabilities class used when do we used Selenium Grid.
* We have to execute mutiple TestCases on multiple Systems with different browser with Different version and Different Operating System.

**Example:**

WebDriver driver;  
String baseUrl , nodeUrl;  
baseUrl = "https://www.facebook.com";  
nodeUrl = "http://192.168.10.21:5568/wd/hub";

DesiredCapabilities capability = DesiredCapabilities.firefox();  
capability.setBrowserName("firefox");  
capability.setPlatform(Platform.WIN8\_1);

driver = new RemoteWebDriver(new URL(nodeUrl),capability);  
driver.get( baseUrl);  
driver.manage().window().maximize();

**11 - How to handle changing element locator values/dynamic locator values? For Example, using Preceding-sibling, Descendant, Ancestor etc**

If we have an element on the webpage with is having only one property define and that is dynamic, constantly changing, then in such scenario we need to check about the preceding-sibling/following-sbiling which are not changing in the DOM structure then with the help of those we can find our desired element.

e.g.

<div id='a1'>  
 <div id='b1'>  
 <div id='xyz'>  
 <div id='b2'>  
</div>

if the div tag having id 'xyz' its constantly changing but the div having ids b1 and b2 are not changing then we can write xpath like **//div[@id='b1']/following-sibling::div** or **//div[@id='b2']/preceding-sibling::div**

Similarly where the element we are finding with the property, we will get multiple element in such case we can go for parent or ancestor to find the unique path structure for our element i.e. element might found multiple with the give property of the element but we can check the parent and ancestor unique property and using that we can identify our element uniquely using xpath like below:

//input[@id='q']/parent::div[@name = 'xyz']

//input[@id='q']/ancestor::div[@id='test123']

In ancestor we will get all grandparents to parent hierarchies.

Same way we can use descendant as well as below, with this we will get many div descendants among which we can find the desired element.

In descendant we will get the descendant hierarchies under the top node (div[@id='\_Alw']) only

//div[@id='\_Alw']/descendant::div[@id='mv-noti-error']/span

**8 - How to operate on windows based pop-up? Is it feasible via Selenium? For Eg: Using AutoIT, Robot Class etc**

Using Selenium we cannot able to handle windows based popup, but with the help of Java Robot class and AutoIT tool we still can able to handle it in Selenium. Selenium is only capable working on the browsers not on the windows popup or applications. There is also WinAppDriver (Microsoft open source) tool which we can use in coordination with Selenium to automate the windows based application.

Example1. When we open a website there is a popup comes for login credentials to enter and that is a windows popup using robot we can do as below:

We need to import the Robot class from java.awt.Robot.

Robot robot = new Robot();

robot.keyPress(KeyEvent.VK\_T);  
 robot.keyRelease(KeyEvent.VK\_T);  
 robot.keyPress(KeyEvent.VK\_E);  
 robot.keyRelease(KeyEvent.VK\_E);  
 robot.keyPress(KeyEvent.VK\_S);  
 robot.keyRelease(KeyEvent.VK\_S);  
 robot.keyPress(KeyEvent.VK\_T);  
 robot.keyRelease(KeyEvent.VK\_T);  
 robot.keyPress(KeyEvent.VK\_TAB);  
 robot.keyRelease(KeyEvent.VK\_TAB);

robot.keyPress(KeyEvent.VK\_T);  
 robot.keyRelease(KeyEvent.VK\_T);  
 robot.keyPress(KeyEvent.VK\_E);  
 robot.keyRelease(KeyEvent.VK\_E);  
 robot.keyPress(KeyEvent.VK\_S);  
 robot.keyRelease(KeyEvent.VK\_S);  
 robot.keyPress(KeyEvent.VK\_T);  
 robot.keyRelease(KeyEvent.VK\_T);  
 robot.keyPress(KeyEvent.VK\_TAB);

robot.keyPress(KeyEvent.VK\_ENTER);

Example2: For uploading the document on the portal we use to get the windows popup to select the file from our system, using AutoIT we can write a few line of script to provide the file (path of the file) to upload.

/\* It will wait for 8 seconds to appear File Upload dialog.  
 Used Title property of File upload dialog window. \*/

WinWait("Open","",3)

/\* Set control focus to File name Input box of File Upload dialog.  
 Used Class property of File upload dialog window and Class+Instance property for File name Input box. \*/

ControlFocus("[CLASS:#32770]","","Edit1")  
 Sleep(1000)

/\* Set the name of file In File name Edit1 field.  
 "Test.txt" file Is located In AutoIT folder of E drive. So we have to provide full path like E:\AutoIT\Test.txt. \*/

ControlSetText("[CLASS:#32770]", "", "Edit1", "E:\Testing Study\AutoITTest\Resume.docx") Sleep(1000)

; Click on the Open button of File Upload dialog.  
  
 ControlClick("[CLASS:#32770]", "","Button1");

**3. Explain Firefox driver internal architecture:**

WebDriver is an interface provided by Selenium. As we know that interfaces in Java are the collection of constants and abstract methods(methods without any implementation). The WebDriver interface serves as a contract that each browser specific implementation like FireFoxDriver (Chorme, Safari etc.) must follow. The WebDriver interface declares methods like get(), navigate(), close(), switchTo(), getWindowHandles(), getWindowHandles() etc. and the developers of the browser specific drivers implement these methods to get the stuff automated.

Below is the architecture of FirefoxDriver:

public class FirefoxDriver extends RemoteWebDriver {...}

public class RemoteWebDriver implements WebDriver, JavascriptExecutor,  
 FindsById, FindsByClassName, FindsByLinkText, FindsByName,  
 FindsByCssSelector, FindsByTagName, FindsByXPath,  
 HasInputDevices, HasCapabilities, Interactive, TakesScreenshot {....}

Developers of the Firefox browser team which develops the driver for Firefox (FirefoxDriver class) has to implement the unimplemented methods of WebDriver and other interfaces to get the stuff automated. With this architecture the selenium automation can the help of browser native support to automate the web application and Selenium automation directly worked with the browser just like the end user use to manually test the application.

Below is the Selenium Architecture diagram to show how the FirefoxDriver has its architecture defined based on Selenium.

