General:

- Tell me about yourself?
- What is your Role and Responsibilities?
- Architecture of your current application?
- Which Automation framework you are using in your organization can you tell me the folder structure and flow of your framework?

Java:

- Why string is immutable in java?
- what is static in java?
- what is final in java?
- can we create the object for the abstract classes?
- can we create constructor of abstract class?
- can constructor be overloaded. Explain why?
- can main method be overloaded?
- can we override static method?
- can i execute multiple catch blocks without try will it give me compile time error?
- if we declare the main method as private what will happen?
- how to check whether the array is empty and null?
- what is the use of constructor in java?
- what is hashmap? Can we store objects in hashmap and how to retrieve them?
- access modifiers in java and its scope?
- what is meant by thread?
- what is singleton class in java?
- what is the difference between static binding and dynamic binding?
- what is polymorphism?
- how and when to use interface?
- where do you use polymorphism in java?
- design pattern in java.
- how to prevent the override method in java?
- what is the difference between list and set?
- why do we use finally and how it differs from the final keyword?
- Can we use multiple catches? When can we use multiple catches?
- Different between POI and JXL jar?
- Why is the main method static?
- What is the use of static variables?
- Can we instantiate an interface?
- Can we override constructor?
- What is the system.out.println? and use of it?

- How will you access default and protected class?
- Why Object creation not possible in Abstract classes?
- Is Hashmap thread safe?
- What is static, How to set value of static variable
- Can we overload private methods?
- Is it possible to extend Final Class?
- Is it possible to overload main method?
- Is it possible to initialize a variable present in an Interface?
- What would happen, if multiple inheritance is possible, in Java?
- Explain Exceptions hierarchy in java?
- Explain Set and Map in Java?
- Explain about Inheritance.
- Difference between overloading and overriding?
- Difference Encapsulation and Abstraction?
- Difference between throw and throws?
- What All of the classes in the Java Collection Framework have?
- Will Java provide default constructor by own? How
- Difference between Arraylist and Linked List, In which situation they are used?
- Difference between List<String> list = new ArrayList<String>() and ArrayList<String> list = new ArrayList<String>();
- Difference between HashMap and Multimap?
- In which situation the method should be static and when non-static?
- How does HashMap is implemented using key value pair?
- Suppose you have class and abstract class in class there is a user defined constructor and main method which one will get executed first?
- What do you mean by POJO why we use POJO?
- class A have 3 method, class B have 2 method, class B inherited class A, how do you call method of class A by creating object of class B?
- What is this keyword in java?
- What is finally and where do we use it?
- What is Autoboxing and unboxing?
- What is serialization and deserialization?
- What is call by reference and call by value?
- Primitives and Non-Primitives datatypes in Java? String is primitive or non-primitive?
- What is the method overloading?
- Why is it important to override HashCode() when you override equals()?
- Difference between final, finally, finalize?
- Difference between abstract and interface?
- Difference between String Builder and String Buffer?
- How to define dynamic array?

- Can we create the object for an interface?
- Can we overload static method?
- Can we write non-abstract methods in Interface?
- Can we execute a java program without main method?
- What are the classes available in a list interface?
- Difference between Hash Map and Hash Set?
- Where did you use HashMap in your project and also oops concepts in your Automation Framework?
- What is the difference between a checked and unchecked exceptions?
- Difference between Array and ArrayList?
- Difference between ArrayList and LinkedList?
- Can main method be overridden?
- Can we call a non-static variable in static method?
- How to achieve serialization and deserialization?
- What is an abstract modifier?
- Do you know oops concepts and in framework where and how you have implemented it?
- Can we declare a private class?
- What is difference between == and equals?
- How string is immutable?
- Where strings get stored and where reference gets stored?
- Can you please explain with the reference of memory location that how string is immutable?
- If not want to use string class then what can be used?
- Difference between string and stringbuffer.
- I have a table and want to store all table data then which collection should be use and why?
- How to declare list
- Where used set in automation?
- How to store multiple values in one reference?
- If i want that my class should not be extended and instance cannot be created by other class then how to declare class?
- How to find missing implementation in cucumber?
- Method overloading and method overriding? Where used in framework?
- **Static keyword**

Selenium:

- What are the challenges you have faced during testing?
- What strategies you followed while building a selenium framework from scratch?
- Where do you perform the singleton design pattern? If you don't use it, then do you have an idea about this?
- Difference between Implicit, Explicit and Fluent waits in Selenium?
- Pros and cons of Implicit wait and Explicit wait.
- Why we prefer explicit wait instead of fluent wait? What are the disadvantages of fluent wait?
- Without implicit wait selenium script will work or not?
- What is default polling time in explicit wait and in implicit wait?
- Explain about synchronization in selenium?
- Which concept they have implemented in explicit and fluent wait?
- Explain abstraction and interface respect of selenium with some example
- Difference between Factory design and Singleton framework?
- What is page object and page factory model?
- Have you used interface in your framework other than selenium interfaces?
- How do you achieve inheritance in your framework?
- What is Webdriver, Name methods which do not have the implementation?
- What are the methods present in the webdriver interface?
- What's the fastest locator in Selenium?
- What does :: (doubles colon) in sibling xpaths represent?
- Explain. "Driver.manage.window.maximize" (talk about option interface here)
- What is difference between get() and navigate().to() in Selenium?
- How would you check the broken links, in the webpage?
- Difference between submit() and click() in Selenium?
- Difference between absolute XPath (/) and relative XPath (//)
- Difference between findelement and findelements?
- Difference between frames and iframes?
- Return type of findelement and findelements?
- What error will be thrown when no element found for findelement and findelements?
- State some exception which you have faced in your framework? (Don't mention only selenium explain. Explain java exception also)
- Types of Exceptions and how to handle stale element exception?
- What are the interface used in selenium?
- Where do you used inheritance in selenium?
- How do you initialize web elements in POM? What error or exception will come if not initiated?
- If both wait method that is implicit and explicit is mentioned in the script, then which one is work? is it good practice to mention both in good?

• What is the difference between close and quit in selenium?
• How do you handle Alert in Selenium?
• In a web page, there are several Pop-up, but we don't when the pop-up will appear, in this case how we will handle the Pop-up using Selenium WebDriver (Java)
• How to handle file upload when type attribute does not file for upload web element.
• How to cover character keyboard operation from the context menu utilizing user-defined keyword?
• Consider this snippet WebDriver driver=new Chromedriver(); what does the above code snippet mean?
• Where can "Dynamic Polymorphism" in Selenium WebDriver be observed?
• What is the difference between "/" and "//" in XPath?
• If proper Xpath, CssSelector and ID are not available, how do you identify an object?
• Attributes of CSS Selector?
• Which is most faster xpath or css?
• How to get n-th element using XPath and CSS?
• Consider you are only allowed to use css locator, how will you find the parent/grandparent of a web element?
• Will driver.findelements() throw an exception ?
• What is returned by driver().manage() ?
• In selenium, if you want to access the element that has the text "This element has an ID that changes every time the page is loaded" in it, then which of the following will you use?
• On page Object Model Framework (POM), how do you initialize the elements of a page to be used in the runner class? (name of the PageObjects class is ""SignupPage.java"" and the dirver object name is ""driver"").
• Get the values from the dropdown and print them in Ascending order
• Using TreeSet to find elements command
• Does takes screenshot is interface or class
• Selenium uses lots of third parties jars for scripting. Then why do we still go for selenium?
• Why do we have to use build() and perform() with the action object
• Can we use perform() along in scripting without build()
• What is difference between build() and perform() in selenium?
• Return type of getwindowhandle() and getwindowhandles()?
• Window Handling in Selenium -Switching from another window to Parent window
• If the button is disabled? how to check -using getattribute()
• Explain method overloading with selenium and some example
• How do you read excel in the script? (very careful while answering. the counter-question will come as per your answer)

• Do you use the property file in your framework? If yes, then which java concept gets utilize here? (Java Collection)
• On a web page, there are several Pop-up, but we don't when the pop-up will appear, in this case how we will handle the Pop- up using Selenium WebDriver (Java)
• Started automation test suite execution and few test cases are failed in a test run. How can you execute only failed test cases at once (with one click) what design pattern do we use when we trigger different browsers?
• What are approached to handle dynamic WebElement?
• Click last option in the dropdown (Last drop-down changes dynamically)
• How to calculate links on a page? (Answer with HTML tag)
• Write the code to read the value from the excel sheet.
• What is Page Factory in POM Design pattern?
• Suppose you have 10 pages in your application then how to achieve POM. What you will do?
• Annotation used in Page Object Model?
• Ways to find broken links in Selenium?
• How to handle frame in Selenium?
• How to handle Alerts in Selenium?
• Different types of Navigation Commands?
• Difference between assert and verify?
• How to download a file using Selenium?
• How do you manage a set of Data Tables in Selenium?
• How do you automate localization testing -diff language in UI?
• How to avoid NoSuchElementException without using try/catch block and with try/catch block?
• How to handle web tables whose values change dynamically?
• How to check whether web element is enabled or Disabled without using isEnabled method?
• Why is CSS locator faster than Xpath?
• Even though CSS is faster than Xpath ,why do 95% of the companies use XPath ?
• Error is throwing as Element not found but when I go and check that element is available in the web page? The element is not hidden so no need to use Java script executor? How do you solve this?
• How do you execute using headless mode?
• In Selenium, how to get text value from text-box if gettext() is not working?
• If we are using correct locator but still getting element not found error then how you will resolve this error?
• In Page object model once you create loginpage.java class what is the first thing you start with writing initially. How are you initiating writing something into a page class?

• What if Windows popup occurs during test execution and due to that can't execute automated tests, how u will resolve this error?
• Different ways to handle hidden elements?
• What is the difference between click() function in webelement interface and click() function in Actions class?
• Is it possible to change the behavior of a test at runtime?
• Describe how to handle the below items using selenium -iframe -windows -table -Alerts
• How to click right click of mouse?
• How to scroll down a page?
• What will driver.getWindowHandels() return?
• How will you automate Windows based application?
• In cucumber, in which class you have gluecode, how many classes for gluecode and what was the program line limit for class?

Java Program:

• Swap string without 3rd variable?
• WAP to find 2nd highest number in an array
• Duplicates in a String?
• How to find the length of the string without using length?
• Largest Number in an Array?
• Reverse string without using reverse function
• Write code to print the Fibonacci series?
• Write code to print only the even numbers from an array.
• Write code to find special character, number, capital and small letter in a given string.
• Write code to check if a number is palindrome?
• Write a Java code to identify, if the pair of strings are an Anagram or not?

TestNG:

• What is the importance of the testng framework?
• Why we use TestNG in your framework
• What is the purpose of testing XML
• Explain the purpose of listeners? is it the selenium concept of TestNG?
• Case Scenario: How to run the same method 100 times in TestNG with the same data?
• What is the reporting tool in your framework? and why?
• Some questions in TestNG XML?
• What are different testng annotations?

- How can you configure tests in testng?
- What is @dataprovider?
- Difference between @Factory and @DataProvider?
- @Factory explain with real time example?
- Test Order in TestNG?
- How to add/remove test cases in Testng.xml?
- Explain the difference between beforemethod, beforetest, and beforeclass
- List out the testng annotation hierarchy order?
- How you achieve parallel execution using testng?
- Out of 50 testcases, how you will run only the failed testcases?
- How can you take the screenshot for the failed testcases?
- How can you run the same tests for 10 times?
- Types of Listeners?
- TestNG: Parallel executions, Grouping?
- Difference between after suite and before suite?
- What is the use of testng.xml?
- How many suits can be there in testNG , what if I run all the suits?
- Syntax to perform parallel testing in TestNG and what do you write in <suite tag> also what do you mention in double quotes like parallel = " "
- In testNG, do we have multiple suite in one XML file and what If I want to run all suits?
- What is invocation count in testng?
- Cucumber tags and annotations?
- What is background in Cucumber?
- Difference between Scenario and Scenario Outline?
- Write skeleton of test runner?
- Explain retry analyzer?
- Cucumber tags? And how to run different combinations of tags when multiple tags are present
- Difference between hooks and tags?

Maven:

- Lifecycle of Maven
- Use of Maven surfire plugin. If yes, where and why?
- What is the use of pom.xml?
- CI / CD tools
- What is Jenkins?
- How will you handle dependencies in Maven at run time?

• Today we have executed some tests using maven, but tomorrow when you see that someone deleted all dependencies from .pom file then in that case will you be able to execute tests or not.
• Consider you have to write test/suite for different environments(qa, preproduction, production) and pass different set of data for each environment. How will you do it using maven file(Pom.xml)?
• Can you give some basic commands used in maven project?
• How will you configure Jenkins job?
• What are two components Jenkins is integrated with?
• How you will schedule the deployments?
• What is the purpose of version control tool?
• What are the git commands you have used?
• What is difference between group id and artefact id?
• How and when is Jenkins is used in your Automation?

Webservices:

• Difference between REST and SOAPUI.
• Method in REST.
• Difference between PUT and PATCH call
• How to integrate postman to project?
• How will you handle dynamic payloads in API?
• How do you capture specific responses value and pass to another request?
• What challenges you faced in API testing?
• What is difference between Authorization and Authentication?
• What are the API status codes, you have come across?

• What is difference between OAuth1.0 and OAuth2.O ,When and where do you use and how. Can you write a sample code?

• How you get the response from one api and send to another api?

Functional Testing:

• What is Test Plan?
• Explain the bug life cycle?
• Difference between smoke and sanity tests?
• Difference between regression and retesting?
• Difference between functional and regression testing?
• Difference between severity and priority?
• Difference between Test Plan and Test Strategy?
• Difference between boundary value analysis and equivalence partitioning?
• Difference between white box and black box testing?

• If we are having 1000 test cases, what type of testing carried for automation testing? Can we write a method that returns two or more values? If then, how?
• Bug Life Cycle?
• Bug Triage?
• What is exploratory testing?
• What is adhoc testing?
• What is build acceptance testing?
• What is difference between Validation and Verification?
• Explain severity and priority and High severity with low priority, low severity and high priority?
• Given the test cases having priority of -1,0,1,2 tell me the sequence of execution.
• Explain inner join and outer join in SQL?
• Difference between DELETE,DROP & TRUNCATE?
• What are test design techniques?
• What is deferred bug?
• How you will decide what tests to automate?
• When you decide to stop the testing?
• If your test suite takes 1 and half to run, what you will do to reduce the time?
• How many test cases in your regression test suite? How much time it will take to execute?
• How to cover character keyboard operation from the context menu utilizing user-defined keyword?
• What is most important things to define in a bug?
• Can tell about any achievements you have done in automation?
• Can you tell me the difference between bdd and tdd?
• What is a test plan and what are the steps to create a test plan?
• What are the inbound and outbound for testing?
• What is smoke, regression and sanity testing?
• What is the next step to be taken, if developer rejects the Open defect?
• Explain the test metrices?
• How would you priorities Tests?
• How do you perform Automation Code Review/ Walk Through in your project?
• There are 250 manual test cases , how will you segregate (on what basis) the regression , sanity , smoke suite?
• When Regression ,Sanity , smoke test scripts are executed ?
• How will you decide that this test case is feasible or good candidate for automation?

Agile:

• What is an agile methodology?
• What is the scrum and who is your scrum master?

• Ceremonies followed in Agile methodology?
• Retrospective meeting?
• Describe Scrum ceremony?
• When do you automate in current sprint or next sprint?
• Explain velocity in sprint.
• In tight sprint schedule, if a new requirement is added, how will you handle this situation?
• What is backlog in Scrum methodology?
• You have 30 + sprints in your release how will you design your test scripts and run them?

Managerial round:

• How you will be an asset to the team?
• Why you are looking for a change?
• How soon you can join?
• Suppose if we give manual testing for six months or one year what will you do?
• How interested are you in learning new technologies?
• Failures in your work life.
• As Lead, how do you define quality of product before releases?
• Suppose you are the leam QA and 1 new member join your team and at the same time you have a deadline to meet in next 2 or 3 days so how will you involve that new member in team so that you can utilise him/her to meet deadlines?
• As a QA, where do you see yourself after 3 years?
• What are your strengths and Weakness?
• What are some best practices you learnt and how much difference it made in testing career? Explain before and after situations
• After you have run a full regression test, and find new regression bugs, which bugs would you prioritize. Bugs that suggest that functionality has regressed, or bugs that appear in new features?

Why String is immutable in Java?

String is immutable in Java to ensure that its value remains constant throughout the program's execution. Once a string object is created, its content cannot be modified, which means that any operation that modifies the string creates a new string object. This design decision has several benefits:

1. Security: String immutability prevents any accidental or malicious change to the value of the string. Since strings are often used to store sensitive information such as passwords, making them immutable adds an extra layer of security.

2. Thread safety: Immutable objects in Java, including strings, are inherently thread-safe. This means that multiple threads can access the string object, and there is no risk of data corruption or inconsistency.

3. Caching: String literals are cached by the Java Virtual Machine (JVM), which means that if two String objects contain the same value, they actually point to the same location in memory. This reduces memory usage and improves performance.

Example:

String str1 = ""Hello"";
String str2 = ""Hello"";

Both str1 and str2 point to the same location in memory because they have the same value. If we modify str1 to ""Hello World"", a new object will be created in memory, and str1 will point to that object, leaving the original object containing ""Hello"" unchanged. This ensures that the value of the string remains constant throughout the program's execution.

########## MANAS JHA ##########

What is static in java?

Static in Java is a keyword used to define a variable or method that belongs to a class rather than an instance of the class. It means that the variable or method can be accessed without creating an object of the class.

For example, let's say we have a class called ""Car"" with a static variable ""numberOfCars"". This variable will hold the total number of cars that have been created, regardless of which instance of the ""Car"" class is accessing it.

Here is an example of how to declare a static variable in Java:

```
public class Car {
static int numberOfCars;
}
```

To access the static variable, we can simply use the class name followed by the variable name, like this:

```
Car.numberOfCars += 1;
```

This increases the total number of cars every time a new instance of the ""Car"" class is created , without needing to create a new object for that variable.

########## MANAS JHA ##########

What is final in java?

In Java, ""final"" is a keyword that can be used to declare variables, methods, and classes. When applied to a variable, it means that the value of the variable cannot be changed once it has been initialized. When applied to a method, it means that the method cannot be overridden in a subclass. And when applied to a class, it means that the class cannot be subclassed.

Example:

1. Declaring a final variable:

final int MAX_NUMBER = 100;

Once the variable MAX_NUMBER is assigned the value 100, it cannot be changed.

2. Declaring a final method:

public final void display() {
System.out.println(""This method cannot be overridden."");
}

The ""final"" keyword before the method signature means that this method cannot be overridden by a subclass.

3. Declaring a final class:

public final class MyClass {
// class body
}

The ""final"" keyword before the class name means that this class cannot be subclassed.

What is this keyword word in java?

The ""this"" keyword in Java refers to the current instance of the class in which it is used. It is often used to differentiate between instance variables and local variables that have the same name. One example of using ""this"" in Java is as follows:

public class Person {
private String name;

public Person(String name) {
this.name = name;
}

// without using this keyword
public Person1(String abcd) {
name = abcd;
}

}

In this example, ""this.name"" refers to the instance variable ""name"" of the Person class, while ""name"" alone refers to the local variable passed in as a parameter.

Can we create the object for the abstract classes?

No, we cannot create an object for an abstract class, as abstract classes are incomplete classes and are intended to be extended and implemented by concrete subclasses. The purpose of abstract classes is to provide a common interface and behaviour among its subclasses. However, we can create an object of the concrete subclass which extends the abstract class. Let's take an example of the abstract class:

```
abstract class Animal {
abstract void makeSound();
}

class Dog extends Animal {
void makeSound() {
System.out.println(""Bark"");
}
}

public class Main {
public static void main(String[] args) {
Dog myDog = new Dog();
myDog.makeSound();
}
}
```


In this example, `Animal` is an abstract class with an abstract method `makeSound()`. The `Dog` class extends the `Animal` class and defines the implementation of the `makeSound()` method. We create an object of `Dog` class and call its `makeSound()` method to get the output `'Bark'`.

Can we create constructor of abstract class?

Yes, we can create a constructor of an abstract class. However, the constructor of an abstract class can only be called from a concrete subclass which extends the abstract class.

For example, consider the following abstract class:

```
public abstract class Shape {
protected int x, y;
public Shape(int x, int y) {
this.x = x;
this.y = y;
}
public abstract void draw();
```

}

Here, we have defined a constructor that takes two integer parameters x and y. This constructor is called from the concrete subclass which extends the Shape class.

public class Circle extends Shape {
private int radius;
public Circle(int x, int y, int radius) {
super(x, y);
this.radius = radius;
}
public void draw() {
// code to draw a circle
}
}

In this example, the Circle class extends the Shape class and calls the constructor of the Shape class using the super() keyword.

One more example


```
package practiceSet1;


abstract class AbstractClass {
    // Abstract class constructor
    public AbstractClass() {
        System.out.println("Constructor of AbstractClass");
    }

    // Abstract method (to be implemented by subclasses)
    public abstract void abstractMethod();
}

class ConcreteClass extends AbstractClass {
    // Concrete subclass constructor
    public ConcreteClass() {
        System.out.println("Constructor of ConcreteClass");
    }

    // Implementation of the abstract method
    @Override
    public void abstractMethod() {
        System.out.println("Implementation of abstractMethod in ConcreteClass");
    }
}

public class ConstructorofAbstractclass {
    public static void main(String[] args) {
        ConcreteClass obj = new ConcreteClass();
        obj.abstractMethod();
    }
}
```

Can constructor be overloaded. Explain why?

Yes, constructors can be overloaded in Java. Constructor overloading is the technique of having more than one constructor with varying parameter lists in a Java class. This enables constructors to accept different argument combinations to initialize different values for an object.

For example, consider the following Java class with overloaded constructors:

```
public class ExampleClass {
private int num1, num2;

public ExampleClass() {
num1 = 0;
num2 = 0;
}

public ExampleClass(int x) {
num1 = x;
num2 = 0;
}

public ExampleClass(int x, int y) {
num1 = x;
num2 = y;
}
}
```

In the above example, we have three constructors with different parameter lists - no arguments, one integer argument, and two integer arguments. Each constructor initializes the member variables of the class in a different way.

Constructor overloading provides flexibility in object initialization and makes the code more readable and maintainable.

Can main method be overloaded?

Yes, the main method can be overloaded in Java. Overloading means having multiple methods with the same name but different parameters.

Example:

```
public class MainMethodOverload {
public static void main(String[] args) {
System.out.println(""Hello World!"");
}

public static void main(String arg1, String arg2) {
```

```
System.out.println(""Arguments: "" + arg1 + "", "" + arg2);
}
}
```

In this example, we have two main methods with different parameters. One takes an array of Strings as an argument and the other takes two individual String arguments. When the program is executed, it will print ""Hello World!"" from the first main method. However, if we pass two String arguments when running the program (`java MainMethodOverload argument1 argument2`), it will execute the second main method that prints the arguments instead.

Can we override static method?

No, it is not possible to override a static method in Java as static methods belong to the class and not the object. Therefore, the method can only be accessed through the class name and not via objects. However, you can declare a static method with the same signature in the subclass, and it will shadow the inherited static method in the parent class.

Example:


```
// Parent class with a static method

public class Parent {
public static void display() {
System.out.println(""Parent's static method"");
}
}

// Child class with the same static method signature

public class Child extends Parent {
public static void display() {
System.out.println(""Child's static method"");
}
}

// Accessing the parent class static method

Parent.display(); // Output: Parent's static method

// Accessing the child class static method

Child.display(); // Output: Child's static method
```

Explantion -2

Overriding a static method:

Overriding is the concept of providing a new implementation for a method in a subclass, which has the same name, return type, and parameters as the method in the superclass. However, it's not possible to override a static method because static methods belong to the class and not to the instances of the class. Therefore, when a subclass defines a static method with the same name, return type, and parameters as the static method in the superclass, it's not considered as overriding, but rather it's considered as hiding the static method in the superclass.

Here is an example that demonstrates this concept:

```
class Superclass {
   static void printMessage() {
      System.out.println("Hello, I am a static method in the Superclass");
   }
}

class Subclass extends Superclass {
   static void printMessage() {
      System.out.println("Hello, I am a static method in the Subclass");
   }
}

public class Main {
   public static void main(String[] args) {
      Superclass.printMessage(); // Output: Hello, I am a static method in the Superclass
      Subclass.printMessage();   // Output: Hello, I am a static method in the Subclass
      Superclass obj = new Subclass();
      obj.printMessage();       // Output: Hello, I am a static method in the Superclass
   }
}
```

In the above example, we have a Superclass with a static method printMessage(). We also have a Subclass that extends the Superclass and defines a static method printMessage() with the same name, return type, and parameters. When we call printMessage() on the Superclass and the Subclass, it prints the corresponding message as expected. However, when we create an instance of the Subclass and call printMessage() on it, it calls the static method in the Superclass instead of the one in the Subclass. This is because static methods belong to the class and not to the instances of the class.

Overloading a static method:
Overloading is the concept of providing multiple methods in a class with the same name but different parameters. It's possible to overload a static method because the method signature (name and parameters) determines which method to call at compile-time.

Here is an example that demonstrates this concept:

```
class MyClass {
   static void printMessage() {
      System.out.println("Hello, I am a static method with no parameters");
   }

   static void printMessage(String message) {
      System.out.println("Hello, I am a static method with a message: " + message);
```

```
    }
}

public class Main {
    public static void main(String[] args) {
        MyClass.printMessage();             // Output: Hello, I am a static method with no parameters
        MyClass.printMessage("Hello, World!"); // Output: Hello, I am a static method with a message: Hello, World!
    }
}
```

In the above example, we have a MyClass with two static methods printMessage(), one with no parameters and the other with a String parameter. When we call printMessage() with no arguments, it calls the first method with no parameters. When we call printMessage() with a String argument, it calls the second method with the String parameter. This works because the method signature (name and parameters) determines which method to call at compile-time.

What is finally and where do we use it?

Finally is a code block in Java that is associated with try-catch statements. It is used to execute the code block irrespective of whether an exception is thrown or not.

Syntax:

```
try {
  //Code block that may throw an exception
}
catch (ExceptionType e) {
  //Code block to handle the exception
}
finally {
  //Code block to be executed irrespective of whether an exception is thrown or not
}
```

In the above syntax, the try block contains the code that may throw an exception. The catch block catches the exception and handles it. The finally block contains the code that will be executed after the try block, irrespective of whether an exception was thrown or not.

The finally block is often used to perform tasks like closing resources, such as files or database connections, that were opened in the try block. This ensures that the resources are always released, even if an exception occurs during the execution of the try block.

Example:

```
try {
  //Code block that may throw an exception
  FileInputStream file = new FileInputStream(""file.txt"");
  //...
}
catch (FileNotFoundException e) {
  //Code block to handle the exception
```

```
  System.out.println(""File not found."");
}
finally {
  //Code block to be executed irrespective of whether an exception is thrown or not
  file.close();
}
```

In the above example, the try block opens a file input stream, which may throw a FileNotFoundException. The catch block handles this exception by printing an appropriate message. The finally block closes the file input stream, ensuring that the resource is always released.

What is Autoboxing and unboxing?

Autoboxing and unboxing are features in Java that allow automatic conversion between primitive data types and their corresponding wrapper classes.

Autoboxing refers to the automatic conversion of primitive data types to their corresponding wrapper class objects. For example, when an int value is assigned to an Integer object, autoboxing will automatically take place.

Example:

```
int num = 10;
Integer integerObj = num; // autoboxing
```

Unboxing refers to the automatic conversion of wrapper class objects to their corresponding primitive data types. For example, when an Integer object is assigned to an int variable, unboxing will automatically take place.

Example:

```
Integer integerObj = new Integer(10);
int num = integerObj; // unboxing
```

Autoboxing and unboxing can make code more concise and readable, but they can also have a performance impact if used excessively in high-performance applications.

What is serialization and deserialization?

Serialization is the process of **converting an object into a stream of bytes** so that it can be stored in a file or transmitted over a network. The byte stream created is platform independent so the object can be reconstructed on any other Java platform.

Java provides a mechanism called Object Serialization to serialize and deserialize objects. An object can be serialized by implementing the java.io.Serializable interface. This interface has no methods and is used to identify the class implementing it as being serializable.

Here is an example code for serialization:

```
import java.io.*;

class Student implements java.io.Serializable {
  public String name;
  public int rollNo;
}

public class SerializeDemo {

  public static void main(String [] args) {
    Student student = new Student();
    student.name = ""John Doe"";
    student.rollNo = 12345;

    try {
      FileOutputStream fileOut = new FileOutputStream(""student.ser"");
      ObjectOutputStream out = new ObjectOutputStream(fileOut);
      out.writeObject(student);
      out.close();
      fileOut.close();
      System.out.println(""Serialized data is saved in student.ser"");
    } catch (IOException i) {
      i.printStackTrace();
    }
  }
}
```

Deserialization is the process **of converting a serialized object back into an object in memory.** Java provides ObjectInputStream class to deserialize an object. Here is an example code for deserialization:

```
import java.io.*;

class Student implements java.io.Serializable {
  public String name;
  public int rollNo;
}

public class DeSerializeDemo {

  public static void main(String [] args) {
    Student student = null;

    try {
      FileInputStream fileIn = new FileInputStream(""student.ser"");
      ObjectInputStream in = new ObjectInputStream(fileIn);
      student = (Student) in.readObject();
      in.close();
      fileIn.close();
    } catch (IOException i) {
      i.printStackTrace();
```

```
        return;
      } catch (ClassNotFoundException c) {
        System.out.println(""Student class not found"");
        c.printStackTrace();
        return;
      }

      System.out.println(""Deserialized Student..."");
      System.out.println(""Name: "" + student.name);
      System.out.println(""Roll no: "" + student.rollNo);
    }
}
```

Explanation 2

Serialization in Java is the process of converting an object into a byte stream so that it can be easily stored in a file, sent over a network, or persisted in a database. Deserialization is the reverse process, where the byte stream is converted back into an object. Java provides the `java.io.Serializable` interface, and objects of a class that implements this interface can be serialized.

```
import java.io.*;

// A class that implements Serializable
class Student implements Serializable {
    private static final long serialVersionUID = 1L; // For version control
    private String name;
    private int rollNumber;

    // Constructor
    public Student(String name, int rollNumber) {
        this.name = name;
        this.rollNumber = rollNumber;
    }

    // Getter methods
    public String getName() {
        return name;
    }

    public int getRollNumber() {
        return rollNumber;
    }
}

public class SerializationExample {

    public static void main(String[] args) {
        // Creating an object to be serialized
        Student student = new Student("John Doe", 12345);
```

```
        // Serialization
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("student.ser"))) {
            oos.writeObject(student);
            System.out.println("Serialization completed. Object saved to student.ser");
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Deserialization
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("student.ser"))) {
            Student deserializedStudent = (Student) ois.readObject();
            System.out.println("Deserialization completed. Object details:");
            System.out.println("Name: " + deserializedStudent.getName());
            System.out.println("Roll Number: " + deserializedStudent.getRollNumber());
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```


In this example:

1. The Student class implements the Serializable interface.
2. An object of the Student class is created and serialized to a file named "student.ser" using ObjectOutputStream.
3. The same object is deserialized from the file using ObjectInputStream.
4. The details of the deserialized object are then printed.
5. Make sure to handle exceptions appropriately in a production environment. Also, note the use of serialVersionUID to control the version of the class during deserialization. This helps prevent issues when the class structure changes between serialization and deserialization.

What is call by reference and call by value?

In Java, when we pass a variable to a method, it is either passed by value or passed by reference.

Call by value means that a copy of the value of the variable is passed to the method, and any modifications made to the variable inside the method do not affect the original variable. Here's an example:

```
public static void changeValue(int x) {
  x = 10;
}

public static void main(String[] args) {
  int num = 5;
  changeValue(num);
  System.out.println(num); // Output: 5
}
```

In this example, we pass the variable `num` to the `changeValue` method, which changes the value of `x` to 10. However, when we print out `num` in the main method, it still has the value of 5, because the `changeValue` method was only modifying a copy of the value.

Call by reference means that a reference to the original variable is passed to the method, and any modifications made to the variable inside the method affect the original variable as well. **However, Java does not support call by reference.** Instead, it simulates call by reference using object references. Here's an example:

```
public static void changeValue(int[] arr) {
  arr[0] = 10;
}

public static void main(String[] args) {
  int[] nums = {5, 6, 7};
  changeValue(nums);
  System.out.println(nums[0]); // Output: 10
}
```


In this example, we pass the array `nums` to the `changeValue` method, which changes the value of the first element in the array to 10. Since `nums` is an object reference to the array, and the `changeValue` method also has a reference to the same array, the change made inside the method affects the original variable.

Primitives and Non-Primitives datatypes in Java? String is primitive or non-primitive?

Primitives are the basic data types that are built into the Java programming language, while non-primitives are objects that are derived from classes. The eight primitive data types in Java are:

1. boolean
2. byte
3. char
4. short
5. int
6. long
7. float
8. double

String is a non-primitive data type in Java. It is an object that represents a sequence of characters.

Here's an example of declaring and initializing primitive and non-primitive variables in Java:

```
java
// declaring and initializing primitive variables
boolean isTrue = true;
byte byteVal = 10;
char charVal = 'A';
```

```
short shortVal = 100;
int intVal = 1000;
long longVal = 1000000L;
float floatVal = 3.14f;
double doubleVal = 3.14159;

// declaring and initializing non-primitive variables
String strVal = ""Hello, World!"";
```

Note that the variables of primitive data types are initialized to default values (e.g. 0 for numeric types, false for boolean), while non-primitive variables are initialized to null by default.

What is the method overloading?

Method overloading is a feature in Java where a class can have multiple methods with the same name but different parameters. These methods can differ in the number or type of parameters they accept. When a method is called, the compiler matches the arguments to the appropriate method based on the number and type of parameters.

For example, suppose we have a class called Calculator with the following two methods:


```
public int add(int x, int y) {
   return x + y;
}

public double add(double x, double y) {
   return x + y;
}
```


Both methods are named add and accept two parameters, but the first method accepts two integers while the second method accepts two doubles. If we call the add method with two integers, the first method will be called. If we call it with two doubles, the second method will be called.

```
Calculator calculator = new Calculator();
int sum = calculator.add(2, 3);      // Calls the first add method
double sumDouble = calculator.add(2.5, 3.5); // Calls the second add method
```

In this example, the add method is overloaded because it has two different versions that accept different types of parameters.

Why is it important to override HashCode() when you override equals()?

It is important to override hashCode() when you override equals() because both methods are used to compare objects in Java. The equals() method is used to compare the actual contents of two objects, whereas the hashCode() method is used to compare the hash code value of two objects. If you override the equals()

method, but don't override hashCode(), then the comparison of two objects will not work as expected, because the hash codes of the two objects may not be equal.

Example Java code:

```
public class Person {
   private String name;
   private int age;

   public Person(String name, int age) {
      this.name = name;
      this.age = age;
   }

   //Override equals method
   @Override
   public boolean equals(Object obj) {
      if (this == obj) {
         return true;
      }
      if (!(obj instanceof Person)) {
         return false;
      }
      Person other = (Person) obj;
      return name.equals(other.name) && age == other.age;
   }

   //Override hashCode method
   @Override
   public int hashCode() {
      return Objects.hash(name, age);
   }
}
```


In the above example, the Person class has been defined with two properties, name and age. The equals() method has been overridden to compare two Person objects based on their name and age properties. The hashCode() method has been overridden to return the hash code value of the name and age properties using the Objects.hash() method, which returns a hash value based on the hash code values of the input objects. This ensures that two Person objects that are equal will have the same hash code value.

Difference between abstract and interface?

Abstract classes and interfaces are used in Java for abstraction and abstraction is a fundamental concept in object-oriented programming. Both the abstract class and interface can contain abstract methods that do not have a body or implementation in the class/interface itself.

Abstract class:
- An abstract class is a class that cannot be instantiated. It provides a common implementation for its subclasses, which extends the abstract class.

- It can have both abstract and non-abstract methods.
- It can have instance variables, static variables, constants, etc.
- An abstract class can also have a constructor, which is called when a subclass is created.
- Subclasses of an abstract class must implement all its abstract methods.

Example code for abstract class:

```
abstract class Shape {
  String color;

  Shape(String color) {
    this.color = color;
  }

  //abstract method
  abstract double area();

  //non-abstract method
  void getColor() {
    System.out.println(""The color of the shape is "" + color);
  }
}

class Circle extends Shape {
  double radius;

  Circle(String color, double radius) {
    super(color);
    this.radius = radius;
  }

  //implementation of abstract method
  double area() {
    return Math.PI * radius * radius;
  }
}

class Test {
  public static void main(String[] args) {
    Shape s = new Circle(""Red"", 3);// type casting
    s.getColor();
    System.out.println(""The area of circle is "" + s.area());
  }
}
```

Interface:
- An interface is a collection of abstract methods that are implemented by its implementing class.
- It cannot implement any method nor declare any variable or constructor.
- All of its methods are public abstract by default.

- An interface can extend other interfaces.
- A class can implement multiple interfaces.

Example code for interface:

```
interface Shape {
  double area();
}

class Circle implements Shape {
  double radius;

  Circle(double radius) {
    this.radius = radius;
  }
                                    MANAS JHA
  //implementation of interface method
  public double area() {
    return Math.PI * radius * radius;
  }                                                    MANAS JHA
}
 MANAS JHA
class Rectangle implements Shape {
  double length, width;

  Rectangle(double length, double width) {
    this.length = length;          MANAS JHA
    this.width = width;
  }

  //implementation of interface method                              MANAS JHA
  public double area() {
    return length * width;
  }  MANAS JHA
}

class Test {
  public static void main(String[] args) {
    Shape s1 = new Circle(3);
    System.out.println(""The area of circle is "" + s1.area());
    Shape s2 = new Rectangle(4, 5);
    System.out.println(""The area of rectangle is "" + s2.area());
  }
}
```


Difference between String Builder and String Buffer?

In Java, Strings are immutable objects, which means that once a String object is created, its value cannot be changed. However, if we need to modify a String object, we can use either String Builder or String Buffer. Both of these classes provide a similar functionality of appending, inserting, and deleting Strings, but there is a difference between them.

String Builder is a non-synchronized and more efficient version of String Buffer, which means that it can be faster in most cases. They both provide similar methods, but StringBuffer is synchronized which makes it slower in multi-threaded environments.

Here is an example Java code that demonstrates the use of String Builder and String Buffer:

```
// using StringBuilder
StringBuilder sb = new StringBuilder(""Hello"");
sb.append("" World"");
String str1 = sb.toString();

// using StringBuffer
StringBuffer sbf = new StringBuffer(""Hello"");
sbf.append("" World"");
String str2 = sbf.toString();
```


In the above code, we create a StringBuilder and StringBuffer object and append the value "" World"" to the existing ""Hello"" string. Finally, we convert the StringBuilder and StringBuffer objects to strings using the toString() method.

Note that while both StringBuilder and StringBuffer can be used interchangeably in most cases, it's important to use StringBuffer in multi-threaded environments as it is synchronized and thread-safe.

How to define dynamic array?

To define a dynamic array in Java, you can use the ArrayList class. Here's an example of how to create an ArrayList:

```
import java.util.ArrayList;

public class DynamicArrayExample {

  public static void main(String[] args) {

    // create an ArrayList of integers
    ArrayList<Integer> dynamicArray = new ArrayList<Integer>();

    // add elements to the array
    dynamicArray.add(1);
    dynamicArray.add(2);
    dynamicArray.add(3);
```

```
   // print the elements of the array
   System.out.println(dynamicArray);

   // get the size of the array
   int size = dynamicArray.size();
   System.out.println(""The size of the array is: "" + size);

   // access an element of the array
   int element = dynamicArray.get(0);
   System.out.println(""The first element of the array is: "" + element);

   // remove an element of the array
   dynamicArray.remove(1);
   System.out.println(""The elements of the array after removing an element: "" + dynamicArray);
  }

}
```


This code creates an ArrayList of integers, adds elements to the array, prints the elements of the array, gets the size of the array, accesses an element of the array, and removes an element from the array.

Can we create the object for an interface?

No, we cannot create an object for an interface in Java. An interface is just a collection of method signatures and constants, but it does not include any implementation code. However, we can create an object for a class that implements an interface. For example:


```
public interface MyInterface {
  void doSomething();
}

public class MyClass implements MyInterface {
  public void doSomething() {
    // implementation code goes here
  }
}

// we can create an object for MyClass, which implements MyInterface
MyInterface obj = new MyClass();
```


In Java, you can call the default method of an interface by creating an instance of a class that implements the interface and then calling the method on that instance.

For example, consider the following interface with a default method:

```
public interface MyInterface {
   default void myDefaultMethod() {
```

```
        System.out.println("This is a default method.");
    }
}
```

To call the default method myDefaultMethod(), you can create a class that implements the MyInterface interface and then call the method on an instance of that class, like this:

```
public class MyClass implements MyInterface {
    // implementation of any abstract methods in MyInterface
}

// create an instance of MyClass
MyClass myObject = new MyClass();

// call the default method on the instance of MyClass
myObject.myDefaultMethod();
```

When you call myDefaultMethod() on myObject, the implementation of the default method in MyInterface will be called and the output "This is a default method." will be printed to the console.

Note that if a class implements multiple interfaces that have default methods with the same name, the class must override the default method and provide its own implementation. If the class wants to call the original implementation of the default method, it can do so using the syntax InterfaceName.super.methodName().

Can we overload static method?

Yes, we can overload a static method in Java.
Here is an example:


```
public class Example {
  public static void print() {
    System.out.println(""No parameters"");
  }

  public static void print(int num) {
    System.out.println(""Parameter: "" + num);
  }

  public static void main(String[] args) {
    print();
    print(5);
  }
}
```


In this example, we have two static methods named `print`. One takes no parameters, while the other takes an `int` parameter. We can call either method from the `main` method, and they will behave differently based on their parameter list.

Can we override default method ?

Yes, it is possible to override a default method in an interface in Java. When a class implements an interface that has a default method, the class can provide its own implementation for the method, effectively overriding the default implementation provided by the interface.

Here's an example that demonstrates how to override a default method in an interface:

```
public interface MyInterface {
  default void myDefaultMethod() {
    System.out.println("This is the default implementation of myDefaultMethod() in MyInterface.");
  }
}

public class MyClass implements MyInterface {
  @Override
  public void myDefaultMethod() {
    System.out.println("This is a custom implementation of myDefaultMethod() in MyClass.");
  }
}
```


In this example, MyInterface has a default method myDefaultMethod() that prints a message to the console. MyClass implements MyInterface and overrides myDefaultMethod() with its own implementation that also prints a message to the console.

When you create an instance of MyClass and call myDefaultMethod(), the custom implementation in MyClass will be called instead of the default implementation in MyInterface.

```
MyClass myObject = new MyClass();
myObject.myDefaultMethod(); // This is a custom implementation of myDefaultMethod() in MyClass.
```

Note that if a class overrides a default method in an interface, it is still possible to call the original default implementation from the interface using the syntax InterfaceName.super.methodName().

Can we write non-abstract methods in Interface?

Yes, we can write non-abstract methods in an Interface using the default and static keywords. From Java 8, we can define a default implementation for methods in an interface. These methods are referred to as default methods. Similarly, the static keyword is used to define a static method in an interface that can be called without creating an instance of the interface.

Here is an example of an interface with default and static methods:

```
public interface SampleInterface {

  void abstractMethod();

  default void defaultMethod() {
    System.out.println(""This is a default method in the interface"");
  }

  static void staticMethod() {
    System.out.println(""This is a static method in the interface"");
  }
```

```
}
```

In this example, the interface has an abstract method, a default method, and a static method. The abstract method does not have an implementation, while the default and static methods have their own implementation.

Can we execute a java program without main method?

No, we cannot execute a Java program without a main method. The main method is the entry point of the Java program and without it, the program cannot be executed.

However, there are ways to indirectly execute code without a main method by using alternative entry points like static initializers, constructors, or static blocks. But these methods are not recommended as they violate the standard Java programming conventions and can make the code difficult to understand and maintain.

Here is an example of how to execute code without a main method using a static initializer:


```
public class NoMainMethodExample {
  static {
    System.out.println(""This code is executed without a main method"");
  }
}
```


In this example, the static initializer block is used as an alternative entry point to execute code without the main method. When the class is loaded into the JVM, the static block is executed first before any other code in the class.

What are the classes available in a list interface?

The classes available in the List interface are:

1. ArrayList
2. LinkedList
3. Vector (deprecated)
4. Stack (deprecated)

Here's an example of how to create an ArrayList:

```
List<String> list = new ArrayList<>();
```

And here's an example of how to create a LinkedList:

List<String> list = new LinkedList<>();

Can I execute multiple catch blocks without try will it give me compile time error?

No, it is not possible to execute multiple catch blocks without a try block. This will lead to a syntax error during compilation as the catch statement is meant to handle exceptions thrown by the try block. Here's an example of how to use try-catch block in Java:

```
try {
//some code that potentially throws an exception
} catch (ExceptionType1 e) {
//code to handle ExceptionType1
} catch (ExceptionType2 e) {
//code to handle ExceptionType2
} finally {
//code that is executed regardless of whether an exception was thrown or not
}
```

In this example, the try block contains the code that may throw an exception. If an exception of type ExceptionType1 is thrown, the first catch block will execute to handle it. If instead, an exception of type ExceptionType2 is thrown, the second catch block will execute. The finally block will always execute regardless of whether an exception was thrown or not.

If we declare the main method as private what will happen?

If we declare the main method as private, it won't be accessible from outside the class, which means that the entry point for the application will not be found and the program will not execute. It will result in a compile-time error.

For example, consider the following code snippet:

```
private static void main(String[] args) {
System.out.println(""Hello, World!"");
}
```

This code won't compile because the main method is declared as private. To fix this, we need to declare it as public.

Here's the corrected version of the code:

```
public static void main(String[] args) {
System.out.println(""Hello, World!"");
}
```


How to check whether the array is empty and null?

To check if the array is empty, we need to check if the length of the array is 0. We can achieve this by using the length property of the array. Here's an example:

```
int[] arr = new int[0]; // initializing an empty array

if(arr.length == 0) {
System.out.println(""Array is empty"");
}
```

To check if the array is null, we can simply check if the array reference itself is null. Here's an example:

```
int[] arr = null; // initializing a null array reference

if(arr == null) {
System.out.println(""Array is null"");
}
```


What is the use of constructor in java?

A constructor in Java is a special method that is used to initialize the object of a class. It is called when an object is created using the 'new' keyword. The main use of a constructor is to initialize the object's instance variables and set up its initial state.

For example, consider the following class:

```
public class Person {
private String name;
private int age;

public Person(String name, int age) {
this.name = name;
this.age = age;
}
}
```

Here we have defined a constructor that takes in two parameters, 'name' and 'age'. When an object of the Person class is created, the constructor will be called and the 'name' and 'age' fields will be initialized with the values passed in.

We can create an object of the Person class as follows:

```
Person p = new Person(""John"", 30);
```

Here the constructor is called with the arguments ""John"" and 30, and the fields 'name' and 'age' are initialized with these values.

In summary, constructors are used to initialize the variables and objects of a class when they are created, and they ensure that the object is in a valid state before it is used.

What is Hashmap? Can we store objects in Hashmap and how to retrieve them?

Hashmap is a data structure in Java that helps store and access key-value pairs. It provides constant time complexity O(1) for both retrieving and storing values.

Yes, we can store objects in Hashmap. The objects can be stored as values and can be retrieved using their corresponding keys. To retrieve an object from the Hashmap, we need to first get the value associated with the key.

Example:

In this example, we will use a Hashmap to store employee information. Here, the keys will be the employee IDs and the values will be custom `Employee` objects.

```
import java.util.HashMap;

public class Employee {
private String name;
private int age;

public Employee(String name, int age) {
this.name = name;
this.age = age;
}

public String getName() {
return this.name;
}

public int getAge() {
return this.age;
}
}

public class HashmapExample {
public static void main(String[] args) {
HashMap<Integer, Employee> employees = new HashMap<Integer, Employee>();

employees.put(1, new Employee(""John"", 35));
employees.put(2, new Employee(""Doe"", 40));
employees.put(3, new Employee(""Jane"", 28));

Employee emp1 = employees.get(1);
System.out.println(""Employee name: "" + emp1.getName());
System.out.println(""Employee age: "" + emp1.getAge());
}
}
```

Output:

Employee name: John
Employee age: 35

In the above example, we store three employees in a Hashmap using their IDs as keys. We then retrieve the employee with ID 1 using the `get()` method and access its properties using the `getName()` and `getAge()` methods.

Where did you use HashMap in your project and also oops concepts in your Automation Framework?

HashMap is a useful data structure that can be used in Selenium automation projects to store and retrieve data in a key-value format.

Here are some examples of how HashMap can be used in Selenium automation:

Storing WebElements: You can use a HashMap to store WebElements by assigning a unique key to each WebElement, such as the WebElement's ID or a custom identifier. This allows you to easily retrieve the WebElement later by using the key as a reference. For example:

```
HashMap<String, WebElement> elementMap = new HashMap<String, WebElement>();
elementMap.put("searchBox", driver.findElement(By.id("search")));
elementMap.get("searchBox").sendKeys("Selenium");
```

Storing test data: You can use a HashMap to store test data such as usernames, passwords, URLs, etc. by assigning a unique key to each data item. This makes it easy to retrieve the data later in your test scripts. For example:

```
HashMap<String, String> testDataMap = new HashMap<String, String>();
testDataMap.put("username", "testuser");
testDataMap.put("password", "testpass");
driver.findElement(By.id("username")).sendKeys(testDataMap.get("username"));
driver.findElement(By.id("password")).sendKeys(testDataMap.get("password"));
```

Storing configuration settings: You can use a HashMap to store configuration settings for your automation project, such as browser type, timeouts, etc. This allows you to easily modify the settings in one place and have the changes reflected throughout your test scripts. For example:

```
HashMap<String, String> configMap = new HashMap<String, String>();
configMap.put("browserType", "Chrome");
configMap.put("implicitWait", "10");
WebDriver driver = new ChromeDriver();
driver.manage().timeouts().implicitlyWait(Integer.parseInt(configMap.get("implicitWait")),
TimeUnit.SECONDS);
```

In summary, HashMap can be used in Selenium automation projects to store and retrieve data in a key-value format, making it easy to organize and access data in your test scripts.

Access modifiers in java and its scope?

Access modifiers are used to define the visibility of class, methods, variables, and constructors in Java. There are four types of access modifiers in Java:

1. Private: Private access modifier restricts the visibility of variables, methods, and constructors within the same class.

Example:

```
public class Example {
private int value;

private void setValue(int newValue) {
value = newValue;
}
}
```


2. Default: If no access modifier is specified then it is considered as default access modifier. This restricts the visibility of variables, methods, and constructors within the same package.

Example:

```
class Example {
int value;

void setValue(int newValue) {
value = newValue;
}
}
```


3. Protected: Protected access modifier restricts the visibility of variables, methods, and constructors within the same package and subclasses in other packages.

Example:

```
public class Example {
protected int value;

protected void setValue(int newValue) {
value = newValue;
}
}
```

4. Public: Public access modifier has full visibility. It can be accessed from anywhere in the project.

Example:

```
public class Example {
public int value;

public void setValue(int newValue) {
value = newValue;
}
}
```

########## MANAS JHA ##########

What is meant by Thread?

In Java, a Thread refers to a lightweight sub-process in a program that enables multiple activities to take place concurrently within a single program. Threads are used to perform parallel processing and can be used for executing time-consuming tasks. Java provides built-in support for multithreading with the java.lang.Thread class.

Example of creating and starting a thread in Java:

```
class MyThread extends Thread {
public void run() {
System.out.println(""Thread is running"");
}
}

public class Main {
public static void main(String args[]) {
MyThread myThread = new MyThread(); // create a new instance of the thread
myThread.start(); // start the thread
}
}
```

In this example, we create a class called MyThread that extends the Thread class provided by Java. We then override the run() method to specify the code that should be executed when the thread starts. In the main method, we create an instance of MyThread and start it by calling the start() method. The output will be ""Thread is running"".

########## MANAS JHA ##########

What is singleton class in java?

A singleton class in Java is a design pattern which restricts the of a class to only one instance and provides a global point of access to that instance. It involves creating a class with a private constructor and a static method that returns the instance of the class.

Here's an example of a singleton class in Java:

```
public class Singleton {
private static Singleton instance = null;
```

```
private Singleton() { // private constructor
}

public static Singleton getInstance() { // static method to retrieve instance
if(instance == null) {
instance = new Singleton();
}
return instance;
}
}
```

In the above example, the class has a private constructor, ensuring that it cannot be instantiated directly. The getInstance() method is used to retrieve the instance of the class and creates one if it doesn't exist already. The static keyword ensures that the method is available on the class level without the need for an instance of the class.

What is the difference between static binding and dynamic binding?

Static binding and dynamic binding are two concepts that are often used in programming languages, including Java. The main difference between static binding and dynamic binding is the time at which the binding occurs.

Static binding refers to the process of linking a function call to the function definition at the compile-time. It occurs at the time of compilation and is used for method overloading. When you overload a method in Java, the method call is resolved at compile-time. The decision which overloaded method to call is based on the parameters passed during the method call.

Example of Static Binding in Java:

```
public class Animal{
public void sound(){
System.out.println(""Animal is making a sound"");
}
}
```



```
public class Dog extends Animal{
public void sound(){
System.out.println(""Dog is barking"");
}
}

public class Main{
public static void main(String[] args){
Animal animal = new Animal();
animal.sound(); // Animal is making a sound

Animal dog = new Dog();
dog.sound(); // Dog is barking - this is dynamic binding
}
}
```

Dynamic binding, also known as late binding, refers to the process of linking a function call to the function definition at runtime. It occurs at the time of execution and is used for method overriding. When you override a method in Java, the method call is resolved at runtime based on the actual class of the object on which the method is called.

Example of Dynamic Binding in Java:

```
public class Animal{
public void sound(){
System.out.println(""Animal is making a sound"");
}
}

public class Dog extends Animal{
public void sound(){
System.out.println(""Dog is barking"");
}
}

public class Cat extends Animal{
public void sound(){
System.out.println(""Cat is meowing"");
}
}

public class Main{
public static void main(String[] args){
Animal animal = new Animal();
animal.sound(); // Animal is making a sound

Animal dog = new Dog();
dog.sound(); // Dog is barking - this is dynamic binding

Animal cat = new Cat();
cat.sound(); // Cat is meowing - this is dynamic binding
}
}
```

In summary, static binding is used for method overloading, while dynamic binding is used for method overriding.

########## MANAS JHA ##########

What is polymorphism?

Polymorphism is a concept in object-oriented programming that allows objects of different types to be treated as if they are of the same type. This means that a single interface or method can be used to represent multiple different classes. There are two types of polymorphism: compile-time polymorphism, which is achieved through method overloading, and runtime polymorphism, which is achieved through method overriding.

For example, let's say we have a Animal class and Lion and Dog subclasses that inherit from it. We can create an array of Animal objects that can hold both Lion and Dog objects, and call the same method on all of them:

```
Animal[] zoo = new Animal[2];
zoo[0] = new Lion();
zoo[1] = new Dog();
for (Animal animal : zoo) {
animal.makeSound();
}
```

In this example, the makeSound() method is overridden in the Lion and Dog subclasses, so it will output different sounds depending on the class of the object that is calling it. This is an example of runtime polymorphism.

How and when to use interface?

Interfaces in Java are used to define a standard set of methods that a class must implement. It is used when you want to provide a contract for the implementation of a certain functionality. This is especially useful in large-scale applications where multiple classes may need to implement the same set of methods.

Some of the common scenarios where interface is used are:

1. Multiple inheritances - Since Java doesn't support multiple inheritance, an interface can be used to provide the functionality of multiple inheritance.

2. Loose coupling - Interfaces make use of contract programming which in turn enables loose coupling between different parts of the program.

3. Frameworks - Interfaces are very useful when creating frameworks, libraries or APIs. By defining an interface, developers can ensure that users can easily integrate with their code.

Here is a simple example of how to use interfaces:


```
public interface Animal {
public void eat();
public void sleep();
}

public class Dog implements Animal {
public void eat() {
System.out.println(""Dog is eating"");
}
public void sleep() {
System.out.println(""Dog is sleeping"");
}
}
```

In the example above, we have an interface called 'Animal' that defines the methods 'eat' and 'sleep'. We have also created a class called 'Dog' that implements the 'Animal' interface. The 'Dog' class then overrides the methods defined in the interface and provides its own implementation.

By using interfaces, we have ensured that any class that implements the 'Animal' interface must provide an implementation of the 'eat' and 'sleep' methods. This provides a contract for other developers to follow and ensures that different parts of the program can easily integrate.

Where do you use polymorphism in java?

Polymorphism is the concept of using a single method or class to represent multiple different forms or types. In Java, we use polymorphism in various ways, such as:

1. Method overloading - Defining multiple methods with the same name but different parameters. For example:


```
public int add(int a, int b) {
return a + b;
}
```



```
public int add(int a, int b, int c) {
return a + b + c;
}
```

2. Method overriding - Overriding the implementation of an inherited superclass method in a subclass. For example:


```
class Animal {
void makeSound() {
System.out.println(""Animal is making a sound"");
}
}
```



```
class Dog extends Animal {
void makeSound() {
System.out.println(""Bark! Bark!"");
}
}
```

3. Object typecasting - Converting an object of one class to another class by explicitly stating the type of object. For example:

```
Animal animal = new Dog();
Dog dog = (Dog) animal;
```

In this case, we use polymorphism to treat the Dog object as an Animal object, but also to access the specific methods and properties of the Dog class.

Why do we use finally and how it differs from the final keyword?

Finally is a block of code that is used to execute a set of statements regardless of whether an exception is thrown or not in a try block. It is mainly used to release resources acquired in a try block. On the other hand, the final keyword is used to declare a constant variable in Java. Once a variable is declared as final, its value cannot be changed.

Here is an example of using the finally block:

```
try {
//code block
} catch (Exception e) {
//exception handling
} finally {
//code that needs to be executed regardless of whether an exception occurs or not.
}
```

And here is an example of using the final keyword:

```
final double pi = 3.14;
//pi value cannot be changed because it's declared as final.
```


Can we use multiple catches? When can we use multiple catches?

Yes, we can use multiple catch blocks in Java. Multiple catch blocks are used to handle different types of exceptions that may occur in a try block.

We can use multiple catch blocks when we need to handle different types of exceptions separately. For example, if we are reading data from a file, we may need to handle FileNotFoundException if the file is not found, and IOException if there is an error while reading the data from the file.

Here is an example code that demonstrates the use of multiple catch blocks in Java:


```
try {
   // code that may throw exceptions
} catch (FileNotFoundException e) {
   // handle FileNotFoundException here
} catch (IOException e) {
   // handle IOException here
} catch (Exception e) {
   // handle any other exception here
}
```


In this example, the first catch block will handle FileNotFoundException, the second catch block will handle IOException, and the third catch block will handle any other exception that may occur. The order of the catch blocks matters, and we should always catch the more specific exceptions before the more general ones.

Different between POI and JXL jar?

POI and JXL are two popular Java libraries used for reading/writing Excel files.

POI:

Apache POI (Poor Obfuscation Implementation) is a Java library used for working with Microsoft Office documents such as Excel, Word, and PowerPoint. It provides APIs to read and write Office documents with Java programs. POI supports a variety of file formats, including XLS, XLSX, DOC, DOCX, PPTX, and many others.

To use POI for reading/writing Excel files, you need to add the following Maven dependency to your project:

```
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi</artifactId>
  <version>4.1.2</version>
</dependency>
```


Here is an example code for reading an Excel file using POI:


```
try {
  FileInputStream file = new FileInputStream(new File(""data.xlsx""));
  Workbook workbook = new XSSFWorkbook(file);
  Sheet sheet = workbook.getSheetAt(0);
  Iterator<Row> rowIterator = sheet.iterator();
  while (rowIterator.hasNext()) {
    Row row = rowIterator.next();
    Iterator<Cell> cellIterator = row.cellIterator();
    while (cellIterator.hasNext()) {
      Cell cell = cellIterator.next();
      switch (cell.getCellType()) {
        case Cell.CELL_TYPE_NUMERIC:
          System.out.print(cell.getNumericCellValue() + ""t"");
          break;
        case Cell.CELL_TYPE_STRING:
          System.out.print(cell.getStringCellValue() + ""t"");
          break;
      }
    }
    System.out.println("""");
  }
  file.close();
} catch (Exception e) {
  e.printStackTrace();
}
```


JXL:

JXL is a Java library used for reading/writing Excel files in the XLS format. It provides APIs to create, read, and modify Excel files with Java programs. JXL is lightweight and easy to use but does not provide support for newer file formats such as XLSX.

To use JXL for reading/writing Excel files, you need to add the following Maven dependency to your project:

```
<dependency>
  <groupId>net.sourceforge.jexcelapi</groupId>
  <artifactId>jxl</artifactId>
  <version>2.6.12</version>
</dependency>
```

Here is an example code for reading an Excel file using JXL:


```
try {
  Workbook workbook = Workbook.getWorkbook(new File(""data.xls""));
  Sheet sheet = workbook.getSheet(0);
  for (int i = 0; i < sheet.getRows(); i++) {
    for (int j = 0; j < sheet.getColumns(); j++) {
      Cell cell = sheet.getCell(j, i);
      System.out.print(cell.getContents() + ""t"");
    }
    System.out.println("""");
  }
  workbook.close();
} catch (Exception e) {
  e.printStackTrace();
}
```


In summary, POI and JXL are both useful Java libraries for working with Excel files. POI supports a wider range of file formats and provides more features than JXL, while JXL is lightweight and easier to use but only supports XLS files.

Why is the main method static?

The main method in Java is declared as static because it needs to be executed by the Java Virtual Machine (JVM) before any object of the class is created. It is the entry point for the execution of the program, and it is invoked by the JVM without creating an instance of the class.

Here is an example of a basic main method in Java:

```
java
public class Main {
   public static void main(String[] args) {
      // Program execution starts from here
      System.out.println(""Hello, world!"");
```

```
    }
}
```

In the above example, the main method is declared as static, which means it can be called without creating an instance of the Main class. The JVM calls this method at the start of the program to begin execution.

If the main method were not declared as static, the JVM would need to create an instance of the class before it could execute the method. This would unnecessarily increase the complexity of the program's execution and could cause issues with class loading and initialization.

What is the use of static variables?

Static variables are variables that belong to a class instead of an instance of the class. They are commonly used to hold values that are shared across all instances of a class. Some common use cases for static variables include:

1. Holding constant values that are used across the entire program.
2. Keeping track of the number of objects created from a class.
3. Storing configuration values that are used by all instances of a class.
4. Sharing data between different parts of a program.

Here is an example of how to declare and use a static variable in Java:

```
public class MyClass {
  static int myStaticVariable = 0;

  public void myMethod() {
    myStaticVariable++;
    // ... do something with myStaticVariable ...
  }
}
```


In this example, the static variable ""myStaticVariable"" is declared at the class level, meaning it is shared across all instances of the class. The ""myMethod"" method increments the value of the static variable each time it is called, allowing all instances of the class to access and modify the same value.

Can we instantiate an interface?

No, we cannot instantiate an interface in Java as interfaces cannot be instantiated directly.

However, we can create a concrete class that implements the interface and then instantiate the concrete class.

For example, let's say we have the following interface:

```
public interface MyInterface {
  public void myMethod();
}
```

We can create a concrete class that implements the interface:

```
public class MyClass implements MyInterface {
  public void myMethod() {
    System.out.println(""MyMethod is called from MyClass"");
  }

  // Other methods and fields
}
```

Now, we can instantiate the concrete class:


```
MyInterface obj = new MyClass();
obj.myMethod(); // Output: MyMethod is called from MyClass
```


In this example, we first created a concrete class `MyClass` that implements the interface `MyInterface`. We then instantiated `MyClass` and assigned it to the interface variable `obj`. Finally, we called the `myMethod()` method through the `obj` variable, and the output was printed to the console.

Can we override constructor?

No, constructors cannot be overridden in Java.

However, a subclass can call a superclass constructor using the `super()` keyword, which allows it to use the constructor of the superclass. This can be used to modify or enhance the functionality of the superclass.

For example:


```
public class Superclass {
  public Superclass(int num) {
    // constructor code
  }
}

public class Subclass extends Superclass {
  public Subclass(int num) {
    super(num); // calls the constructor of the superclass
    // subclass constructor code
  }
}
```

In this example, the `Subclass` extends the `Superclass` and calls its constructor using `super(num)`. This allows the `Subclass` to perform additional operations before or after the construction of the `Superclass`.

########## MANAS JHA ##########

What is the system.out.println? and use of it?

System.out.println is a method in the Java programming language used to print text or values to the console output. This method allows us to display the values of variables, debug our code, and provide feedback to users.

Here is an example of how to use System.out.println in Java:

```
public class Main {
  public static void main(String[] args) {
    String name = ""John"";
    int age = 25;
    double height = 1.75;

    // Print out variable values using System.out.println
    System.out.println(""Name: "" + name);
    System.out.println(""Age: "" + age);
    System.out.println(""Height: "" + height);
  }
}
```

In the above example, we have declared three variables (name, age, and height) and use System.out.println to print out their values to the console output. This will produce the following output:

```
Name: John
Age: 25
Height: 1.75
```

MANAS JHA

########## MANAS JHA ##########

How to prevent the override method in Java?

To prevent override method in Java, we can use the final keyword. The final keyword can be used in two ways:

1. By making the method final: This will prevent any subclass from overriding the method.

Example:

```
public class ParentClass {
public final void display(){
System.out.println(""This is a final method."");
}
}
```

2. By making the class final: This will prevent any subclass from extending the class and overriding the methods.

Example:

```
public final class ParentClass {
public void display(){
System.out.println(""This is a final class."");
}
}
```

Both of these approaches ensure that the method cannot be overridden by any subclass, hence preventing the override method in Java.

What is the difference between list and set?

List and Set are two different types of collections in Java. The main difference between them is the way they store elements.

List is an ordered collection that allows duplicate elements. Elements are stored in the same order as they are added to the collection. Lists are typically used when order of elements is important.

Example:


```
List<String> colors = new ArrayList<>();
colors.add(""red"");
colors.add(""green"");
colors.add(""blue"");
colors.add(""red"");
System.out.println(colors); // prints [red, green, blue, red]
```


Set is an unordered collection that does not allow duplicate elements. Elements are stored in an undefined order. Sets are typically used when uniqueness of elements is important.

Example:

```
Set<Integer> numbers = new HashSet<>();
numbers.add(1);
numbers.add(2);
numbers.add(3);
numbers.add(2);
System.out.println(numbers); // prints [1, 2, 3]
```

Code for adding elements to a Set and checking if an element is present:

```
Set<String> fruits = new HashSet<>();
fruits.add(""apple"");
```

```
fruits.add(""banana"");
fruits.add("range"");

// check if an element is present in the set
if (fruits.contains(""apple"")) {
System.out.println(""Fruits set contains apple!"");
}

// add an element to the set
fruits.add(""kiwi"");
```


Design pattern in JAVA.

A design pattern is a reusable solution to a commonly occurring problem in software design. In Java, there are various design patterns that can be used to improve the quality of code, flexibility, and maintainability of the software. Some of the famous design patterns in Java include:

1. **Singleton Pattern:** This pattern ensures that a class has only one instance and provides a global point of access to that instance.

Example code:

```
public class SingletonDemo {

private static SingletonDemo instance = new SingletonDemo();

private SingletonDemo(){}

public static SingletonDemo getInstance(){
return instance;
}

public void showMessage(){
System.out.println(""Hello World!"");
}
}
```

2. **Observer Pattern:** This pattern defines a one-to-many relationship between objects so that when one object changes state, all its dependents are notified and updated automatically.

Example code:

```
public class ObserverDemo {

public static void main(String[] args) {

Subject subject = new Subject();

new HexaDecimalObserver(subject);
```

```
new OctalObserver(subject);
new BinaryObserver(subject);

System.out.println(""First state change: 15"");
subject.setState(15);
System.out.println(""Second state change: 10"");
subject.setState(10);
}
}
```

3. **Factory Pattern:** This pattern provides a way to create objects without exposing the creation logic to the client and refers to the newly created object through a common interface.

Example code:

```
interface Animal{
void makeSound();
}

class Cat implements Animal{
public void makeSound(){
System.out.println(""Meow"");
}
}

class Dog implements Animal{
public void makeSound(){
System.out.println(""Bark"");
}
}

class AnimalFactory{

public Animal getAnimal(String animalType){
if(animalType == null){
return null;
}
if(animalType.equalsIgnoreCase(""CAT"")){
return new Cat();
} else if(animalType.equalsIgnoreCase(""DOG"")){
return new Dog();
}
return null;
}
}

public class FactoryDemo{

public static void main(String[] args) {
AnimalFactory animalFactory = new AnimalFactory();
```

```
Animal cat = animalFactory.getAnimal(""CAT"");
cat.makeSound();

Animal dog = animalFactory.getAnimal(""DOG"");
dog.makeSound();
}
}
```


How does HashMap is implemented using key value pair?

HashMap is implemented using key-value pair by hashing the key to get the bucket index where the value is stored. In Java, the HashMap implementation uses an array of linked lists to store the key-value pairs, where each index of the array represents a bucket. The key is hashed using hash function to get the bucket index and the value is stored in the linked list at that index. If two different keys hash to the same index, then both keys will be stored in the same linked list.

Code example:

```
HashMap<Integer, String> hashmap = new HashMap<Integer, String>();
hashmap.put(1, "ne"");
hashmap.put(2, ""Two"");
hashmap.put(3, ""Three"");

String value = hashmap.get(2);
System.out.println(value);
```


The output of this code will be ""Two"", which is the value stored under the key 2.

class A have 3 method, class B have 2 method, class B inherited class A, how do you call method of class A by creating object of class B?

To call a method of class A from class B, you need to create an object of class B and call the method of class A using super keyword. Here's an example code in Java:


```
class A{
public void method1(){
System.out.println(""Method 1 of class A"");
}
public void method2(){
System.out.println(""Method 2 of class A"");
}
}

class B extends A{
public void method3(){
System.out.println(""Method 3 of class B"");
}
public void method4(){
System.out.println(""Method 4 of class B"");
```

```
}
public void callSuperMethod(){
super.method1(); // calling method1 of class A
super.method2(); // calling method2 of class A
}
}

public class Main{
public static void main(String[] args){
B b = new B();
b.callSuperMethod(); // calling methods of class A from class B
}
}
```

In the above example, we have two classes A and B. Class B inherits class A. Class A has two methods - method1 and method2. Class B has two additional methods - method3 and method4. In the method callSuperMethod of class B, we are calling the methods of class A using the super keyword. When we run the main method, it will call the method callSuperMethod of class B which in turn will call the methods of class A. Output of the program will be:

```
Method 1 of class A
Method 2 of class A
```

How will you access default and protected class?

To access a default class or a protected class from a different package, we can do the following:
1. Create an instance of the class in a class that resides in the same package as the default or protected class. This can be done because classes in the same package can access each other's default and protected members.
2. Use inheritance to extend the default or protected class in a class that resides in a different package. This can be done because subclasses can access protected members of their superclasses.
Here's an example of accessing a default class from a different package:

```
package package1;
class MyClass {
  void myMethod() {
    System.out.println(""This is a default method."");
  }
}
package package2;
import package1.*;

class AnotherClass {
  public static void main(String[] args) {
    MyClass obj = new MyClass();
    obj.myMethod(); // Accessing default method from package1
  }
}
```

And here's an example of accessing a protected class from a different package:

```
package package1;
```

```
class MyClass {
  protected void myMethod() {
    System.out.println(""This is a protected method."");
  }
}
package package2;
import package1.*;
class AnotherClass extends MyClass {
  public static void main(String[] args) {
    AnotherClass obj = new AnotherClass();
    obj.myMethod(); // Accessing protected method from package1
  }
}
```


Why Object creation not possible in Abstract classes?

In Java, an abstract class is a class that is declared with the ""abstract"" keyword. It cannot be instantiated (meaning, you cannot create an object of an abstract class directly) because one or more methods are missing their implementation.

The basic purpose of an abstract class is to serve as a blueprint or template for other classes to extend it and provide their own implementation. If we could create an object of an abstract class, it would not make sense, as it would not have all its methods' implementations.

Here is an example:

```
abstract class Animal {
  private String name;

  public Animal(String name) {
    this.name = name;
  }

  public String getName() {
    return this.name;
  }

  public abstract void makeSound(); // Method without implementation
}

class Dog extends Animal {
  public Dog(String name) {
    super(name);
  }

  public void makeSound() {
    System.out.println(""Woof!"");
  }
}

public class Main {
  public static void main(String[] args) {
    Animal dog = new Dog(""Rookie"");
```

```
    System.out.println(dog.getName()); // Output: ""Rookie""
    dog.makeSound(); // Output: ""Woof!""
  }
}
```

In this example, the class Animal is an abstract class, and it has a method called makeSound() without its implementation. The Dog class, which extends the Animal class, provides its own implementation of makeSound().

When we create an object of Dog class, it has inherited the getName() method from the Animal class, and it implements the makeSound() method itself.

If we try to create an object of Animal directly, like this: `Animal animal = new Animal();`, the compiler would give an error because we cannot instantiate an abstract class directly.

Is Hashmap thread safe?

No, HashMap is not thread-safe by default. It means that if two threads access the same HashMap object concurrently, it may result in unexpected behavior such as lost updates, inconsistent state or even an exception.

However, there are thread-safe alternatives to HashMap in Java such as ConcurrentHashMap and Hashtable which can be used in multi-threaded environments. Here is an example of how to use ConcurrentHashMap:

```
ConcurrentHashMap<String, Integer> map = new ConcurrentHashMap<>();
// perform concurrent read and write
Thread t1 = new Thread(() -> {
   for (int i = 0; i < 1000; i++) {
      map.put(""key"" + i, i);
   }
});
Thread t2 = new Thread(() -> {
   for (String key : map.keySet()) {
      System.out.println(map.get(key));
   }
});
t1.start();
t2.start();
// wait for threads to finish
t1.join();
t2.join();
```

In the above code, we have used ConcurrentHashMap to perform concurrent read and write operations without any synchronization issues.

What is static, How to set value of static variable

Static is a keyword in Java that is used to define a class member that belongs to the class and not the instance of the class. A static variable is a variable that is associated with the class, rather than with any particular instance of that class.

To set the value of a static variable, you can use the class name to reference the variable, followed by the assignment operator and the value:

```
public class MyClass {
  static int myStaticVariable;
  public static void main(String[] args) {
    MyClass.myStaticVariable = 10; // Set the value of myStaticVariable to 10
    System.out.println(MyClass.myStaticVariable); // Print the value of myStaticVariable
  }
}
```

In this example, we define a static variable called `myStaticVariable` inside the class `MyClass`. We then set the value of `myStaticVariable` to 10 using the class name `MyClass` followed by the dot operator and the variable name. Finally, we print the value of `myStaticVariable` using the class name and the dot operator.
Note that you can also set the value of a static variable inside a static initialization block:

```
public class MyClass {
  static int myStaticVariable;
  static {
    myStaticVariable = 10;
  }
}
```


In this example, we define a static initialization block that sets the value of `myStaticVariable` to 10. This block will be executed once when the class is loaded into memory.

Can we overload private methods?

Yes, we can overload private methods in Java but, you can access these from the same class.

```
public class Calculator {
  private int addition(int a , int b){
    int result = a+b;
    return result;
  }
  private int addition(int a , int b, int c){
    int result = a+b+c;
    return result;
  }
  public static void main(String args[]){
    Calculator obj = new Calculator();
    System.out.println(obj.addition(12, 13));
    System.out.println(obj.addition(12, 13, 15));
  }
}
```


Is it possible to extend Final Class?

No, it is not possible to extend a final class in Java as the final keyword in the class declaration indicates that the class cannot be subclassed. If an attempt is made to extend a final class, a compile-time error will occur.
Here's an example code snippet to demonstrate this:

```
final class FinalClass {
   // class implementation
}
// Attempt to extend FinalClass
class SubClass extends FinalClass {
   // Error: Cannot extend final class 'FinalClass'
}
```


Is it possible to overload main method?

Yes, it is possible to overload the main method in Java. This means that you can define multiple main methods within the same class, but they must have different signatures in order to be distinguished by the Java compiler.
Here's an example of overloading the main method in Java:
java

```
public class Main {

  public static void main(String[] args) {
    System.out.println(""This is the first main method."");
  }

  public static void main(String arg1, String arg2) {
    System.out.println(""This is the second main method."");
  }
}
```

In this example, we have defined two main methods within the same class, but with different parameter lists. The first main method takes an array of strings as its argument, which is the standard signature for the main method in Java. The second main method takes two string arguments.
When we run this program, we can specify which main method to call by passing the appropriate arguments on the command line:

```
java Main          // Calls the first main method.
java Main foo bar  // Calls the second main method.
```

So, to answer the question, yes, you can overload the main method in Java.

Is it possible to initialize a variable present in an Interface?

No, it is not possible to initialize a variable present in an Interface in Java.
Interfaces in Java can only contain constant variables that are implicitly final and static. These variables can only be initialized at the time of declaration and their values cannot be changed later.
For example:

```
interface MyInterface {
  int MY_CONSTANT = 42;
}
```

Here, the variable MY_CONSTANT is a constant and can be accessed by implementing classes as MyInterface.MY_CONSTANT. However, this variable cannot be initialized or changed by any implementing class.

What would happen, if multiple inheritance is possible, in Java?

As a language, Java does not support multiple inheritance of classes. Multiple inheritance is a feature of some other programming languages, where a class can inherit from multiple classes at the same time. However, Java does support multiple inheritance of interfaces, where a class can implement multiple interfaces.

If multiple inheritance of classes were possible in Java, it could lead to complications such as the diamond problem. The diamond problem occurs when a class inherits from two classes that have a common ancestor, leading to ambiguity in method resolution.

To avoid such complications, Java enforces a single inheritance model of classes, where each class can have only one direct superclass. This helps ensure a clear and understandable hierarchy of class relationships.

Explain Exceptions hierarchy in java?

In Java, an exception is an event that occurs during the execution of a program, which disrupts the normal flow of the program's instructions. Java's exception hierarchy provides a predefined set of classes to identify and handle various types of runtime errors. The hierarchy is as follows:

1. Throwable: It is the superclass of all exception classes in Java. It is at the top of the hierarchy, and it has two main subclasses: Error and Exception.

2. Error: It represents catastrophic errors that are difficult, or impossible for the program to recover from. Examples include OutOfMemoryError and StackOverflowError.

3. Exception: It represents conditions that are recoverable by the program. It has many subclasses, depending on the type of exception.

4. **RuntimeException:** It is a subclass of Exception and represents errors that occur during the runtime execution of a program. They are unchecked exceptions, meaning that the compiler does not enforce a catch or declare requirement on them. Examples include NullPointerException, ArrayIndexOutOfBoundsException, and ArithmeticException.

5. **Checked Exceptions:** These are exceptions that must be caught or declared in the method signature of the method that throws them. They must be handled by the programmer, as they are not automatically caught by the JVM. Examples include IOException, ClassNotFoundException, and SQLException.

6. Unchecked Exceptions: These are exceptions that can be caught, but don't have to be. Examples include NullPointerException, IllegalStateException, and IllegalArgumentException.

Here's an example Java code to demonstrate exception handling:

```
public class ExceptionDemo {
   public static void main(String[] args) {
      try {
         // code that may throw an exception
         int x = 10/0;
      } catch (ArithmeticException e) {
         // exception caught and handled here
         System.out.println(""Error: "" + e.getMessage());
      }
      System.out.println(""Program executed successfully."");
   }
```

}

In the above code, we are trying to divide an integer by zero, which will cause an ArithmeticException. We catch this exception in the catch block and handle it by printing an error message. Finally, we print a message indicating that the program executed successfully.

Explain Set and Map in Java?

Set and Map are two important interfaces in the Collection framework of Java. The key difference between Set and Map is that Set stores only unique elements whereas Map stores key-value pairs.

1. **Set:** Set is an interface that extends the Collection interface. It is used to store unique elements only, i.e., no duplicates are allowed. There are several implementations of the Set interface such as HashSet, TreeSet, and LinkedHashSet.

Example code:

```
// Creating a HashSet and adding elements to it
Set<String> set = new HashSet<String>();
set.add(""Java"");
set.add(""Python"");
set.add(""C++"");
set.add(""Ruby"");
// Iterating over the set and printing its elements
Iterator<String> itr = set.iterator();
while(itr.hasNext()){
   System.out.println(itr.next());
}
```

Output:

```
Java
Python
C++
Ruby
```

2. **Map:** Map is an interface that stores key-value pairs. It is used to associate values with keys, where each key is unique. There are several implementations of the Map interface such as HashMap, TreeMap, LinkedHashMap, and ConcurrentHashMap.

Example code:

```
// Creating a HashMap and adding key-value pairs to it
Map<String, Integer> map = new HashMap<String, Integer>();
map.put(""Java"", 1);
map.put(""Python"", 2);
map.put(""C++"", 3);
map.put(""Ruby"", 4);
// Iterating over the map and printing its key-value pairs
for (Map.Entry<String, Integer> entry : map.entrySet()) {
   String key = entry.getKey();
   Integer value = entry.getValue();
   System.out.println(key + "" -> "" + value);
}
```

Output:

```
Java -> 1
Python -> 2
C++ -> 3
Ruby -> 4
```

Explain about Inheritance.

Inheritance is a mechanism in object-oriented programming where a class can inherit properties and behavior from another class, which is called the superclass. The class that inherits from the superclass is called the subclass.

To implement inheritance in Java, the keyword ""extends"" is used to indicate that a class inherits from another class. For example, consider the following code:

```
public class Animal {
   public void eat() {
      System.out.println(""Animal is eating"");
   }
}
public class Dog extends Animal {
   public void bark() {
      System.out.println(""Dog is barking"");
   }
}
```

In this code, the class ""Dog"" inherits from the class ""Animal"". This means that the subclass ""Dog"" can access the methods and properties of the superclass ""Animal"", including the ""eat"" method.

To create an instance of the ""Dog"" class and call its methods, the following code can be used:

```
Dog myDog = new Dog();
myDog.eat();   // Output: ""Animal is eating""
myDog.bark();  // Output: ""Dog is barking""
```

This shows how inheritance allows us to reuse code and create a hierarchy of classes that share common properties and behavior.

Difference between overloading and overriding?

Overloading and overriding are two important concepts in Java programming.

Overloading refers to the ability to define multiple methods in a class with the same name but different parameters. This enables programmers to create methods that can perform similar tasks but act on different data types or with different parameter lists.

Here's an example:

java

```
public class MyClass {
   public void myMethod(int x) {
      // Do something with an integer value
   }

   public void myMethod(String str) {
      // Do something with a string value
   }
}
```

In this code, we have defined two methods in the `MyClass` class with the same name (`myMethod`), but different parameters (an integer value and a string value). This allows us to use the same method name for similar functionality but with different data types.

Overriding, on the other hand, refers to the ability to define a method in a child class that has the same name and signature as a method in the parent class. This method in the child class ""overrides"" the method in the parent class, providing a new implementation of that method.
Here's an example:
java

```
public class Animal {
   public void makeSound() {
      System.out.println(""Animal is making a sound"");
   }
}
public class Dog extends Animal {
   @Override
   public void makeSound() {
      System.out.println(""Dog is barking"");
   }
}
```

In this code, we have defined two classes: `Animal` and `Dog`. The `Animal` class has a method `makeSound` that prints a generic message, while the `Dog` class overrides this method with a new implementation that specifically says that a dog is barking.
To summarize, the key difference between overloading and overriding is that overloading is used to define multiple methods in a class with the same name but different parameters, while overriding is used to provide a new implementation of a method in a child class that has the same name and signature as a method in the parent class.

Difference Encapsulation and Abstraction?

Encapsulation and Abstraction are two fundamental Object-Oriented Programming (OOP) concepts in Java. The main difference between these two is that Encapsulation refers to the principle of hiding the implementation details, while Abstraction focuses on providing an abstract view of an entity.
Here is a code example to illustrate the difference between Encapsulation and Abstraction in Java:
Encapsulation Example:


```
public class BankAccount {
   private double balance;
   private int accountNumber;
   private String ownerName;

   //getter and setter methods
   public double getBalance() {
      return balance;
   }
   public void setBalance(double balance) {
      this.balance = balance;
   }
   public int getAccountNumber() {
      return accountNumber;
   }
   public void setAccountNumber(int accountNumber) {
      this.accountNumber = accountNumber;
   }
   public String getOwnerName() {
```

```
    return ownerName;
  }
  public void setOwnerName(String ownerName) {
    this.ownerName = ownerName;
  }
}
```

In the above example, we can see that Encapsulation is achieved by marking the fields (balance, accountNumber, and ownerName) as private. The getter and setter methods are used to get and set the values of these fields from outside the class.

Abstraction Example:

```
public abstract class Shape {
  protected int x;
  protected int y;

  //constructor
  public Shape(int x, int y) {
    this.x = x;
    this.y = y;
  }

  //abstract method
  public abstract void draw();
}
```

In the above example, we can see that Abstraction is achieved by marking the class as abstract and defining an abstract method (draw()) without providing any implementation. This indicates that any class that extends the Shape class must implement the draw() method.

To summarize, Encapsulation is used to hide implementation details, while Abstraction is used to provide a simplified view of an entity. Both concepts are important in Object-Oriented Programming and are used extensively in Java.

########################## MANAS JHA ##########################

Difference between throw and throws?

In Java, 'throw' and 'throws' are two different keywords used in handling exceptions.

1. **'throw'** keyword is used to throw an exception explicitly.

Syntax:

```
throw new Exception(""Error Message"");
```

2. **'throws'** keyword is used in method signature to indicate that the method may throw one or more exceptions.

Syntax:

```
public void method_name() throws Exception1, Exception2, ... {
  // method code here
}
```

In other words, 'throws' is used to delegate the responsibility of handling exception to the caller of the method, while 'throw' is used to explicitly throw an exception.

Example:

```
public void divide(int a, int b) throws ArithmeticException {
  if(b == 0) {
    throw new ArithmeticException(""Division by zero is not allowed."");
  }
  int result = a / b;
```

```
    System.out.println(""Result: "" + result);
}
```

In this example, 'throws' is used in method signature to indicate that the method 'divide' may throw an 'ArithmeticException'. Inside the method, 'throw' is used to throw an exception if the divisor is zero.

What All of the classes in the Java Collection Framework have?

All of the classes in the Java Collection Framework have the following common methods:
- add(E element): adds the specified element to the collection.
- remove(Object o): removes the specified element from the collection.
- size(): returns the number of elements in the collection.
- isEmpty(): returns true if the collection is empty.
- contains(Object o): returns true if the collection contains the specified element.
- clear(): removes all the elements from the collection.
- toArray(): returns an array containing all the elements in the collection.

In addition, some of the classes have additional methods specific to their collection type. For example, the List interface includes get(int index), set(int index, E element), and add(int index, E element) methods for accessing elements by index, and the Set interface includes methods for testing set equality such as equals(Object o) and hashCode().

Here is a sample code showing the implementation of the common methods in a Collection interface:

```
public interface Collection<E> {

    public boolean add(E element);

    public boolean remove(Object o);

    public int size();

    public boolean isEmpty();

    public boolean contains(Object o);

    public Object[] toArray();

    public void clear();
}
```

Note: This code only includes the method signatures and not the actual implementation of each method. The implementation will depend on the specific collection class that implements the interface.

Will Java provide default constructor by own? How

Yes, Java provides a default constructor if you have not defined any constructor explicitly. This default constructor is created by the compiler and has no parameters or code within it.

Here is an example:

```
public class MyClass {
  private int number;
  private String name;
```

```
  // This is the default constructor created by Java
  public MyClass() {
  }

  // Other constructors you define
  public MyClass(int number, String name) {
    this.number = number;
    this.name = name;
  }

  // Other methods of the class
}
```

So, if you create an object of `MyClass` like this:
MyClass myObject = new MyClass();
then the default constructor created by Java will be called.

Difference between Arraylist and Linked List, In which situation they are used?

ArrayList and LinkedList are data structures in Java with different implementations and usage scenarios. The main differences between them are:

1. Implementation: ArrayList is implemented as a resizable dynamic array, while LinkedList is implemented as a doubly linked list.
2. Performance: ArrayList can be faster than LinkedList for random access and iteration, while LinkedList is more efficient than ArrayList for insertions and deletions at the beginning or in the middle.
3. Memory usage: ArrayList uses more memory than LinkedList, as it needs to allocate a certain amount of space for its elements, while LinkedList only needs to allocate memory for each element and the pointers.

Here are examples of how to create and use ArrayList and LinkedList in Java:

```
ArrayList<String> arrayList = new ArrayList<>();
// add elements to the end of the list
arrayList.add(""element1"");
arrayList.add(""element2"");
arrayList.add(""element3"");
// access elements by index
String element = arrayList.get(1);
// remove elements by index
arrayList.remove(2);
// iterate over elements
for (String e : arrayList) {
  System.out.println(e);
}
LinkedList<String> linkedList = new LinkedList<>();
// add elements to the end of the list
linkedList.add(""element1"");
linkedList.add(""element2"");
linkedList.add(""element3"");
// add elements to the beginning of the list
linkedList.addFirst(""element0"");
// access elements by index
String element = linkedList.get(1);
// remove elements by index
```

```
linkedList.remove(2);
// iterate over elements
for (String e : linkedList) {
  System.out.println(e);
}
```

In general, ArrayList is more suitable for scenarios where random access and iteration over the elements are frequent, while LinkedList is better for scenarios where insertions and deletions at the beginning or in the middle are common. However, the actual performance may depend on the specific use case and the amount of data involved.

Difference between List<String> list = new ArrayList<String>() and ArrayList<String> list = new ArrayList<String>();

Both List<String> list = new ArrayList<String>() and ArrayList<String> list = new ArrayList<String>() are ways of creating an ArrayList object that can store strings, but there is a difference between them.

List<String> list = new ArrayList<String>()

This line of code creates an object of class ArrayList that is stored in a variable of type List<String>. This approach is called ""**programming to the interface**"".

By creating the ArrayList using the List interface, it enables a more flexible implementation. For example, if we decide later on we want to use a LinkedList instead of an ArrayList, we can just change that one line of code and the rest of our code base remains the same.

ArrayList<String> list = new ArrayList<String>()

This second line creates an object of class ArrayList that is stored in a variable of type ArrayList<String>. This approach is called ""programming to the implementation"".

By creating the ArrayList using the implementation of ArrayList, it provides us access to all the methods, including any specific methods that may not exist in the List interface.

In general, it is better to program to the interface than the implementation for the better maintainance and scalability of code.

Difference between HashMap and Multimap?

HashMap is a collection in Java that stores key-value pairs. It is used to store and retrieve elements based on keys. A key can have only one associated value. On the other hand, Multimap is a collection in Java that allows multiple values to be associated with a single key.

Here is an example code showing the difference between HashMap and Multimap in Java:

HashMap example:

```
import java.util.HashMap;
public class HashMapExample {
  public static void main(String[] args) {
    HashMap<Integer, String> map = new HashMap<Integer, String>();

    // adding elements to HashMap
    map.put(1, ""John"");
    map.put(2, ""Mike"");
    map.put(3, ""Lisa"");

    // retrieving the value associated with key 1
```

```
    System.out.println(""Value for key 1 is "" + map.get(1));
  }
}
```

Output:
Value for key 1 is John

Multimap example:

```
import com.google.common.collect.ArrayListMultimap;
import com.google.common.collect.Multimap;
public class MultimapExample {
  public static void main(String[] args) {
    Multimap<Integer, String> map = ArrayListMultimap.create();

    // adding elements to Multimap
    map.put(1, ""John"");
    map.put(2, ""Mike"");
    map.put(3, ""Lisa"");
    map.put(1, ""David"");
    map.put(2, ""Emily"");

    // retrieving values associated with key 1
    System.out.println(""Values for key 1 are "" + map.get(1));
  }
}
```


Output:
Values for key 1 are [John, David]

As you can see from the examples, HashMap allows only one value to be associated with a key, while Multimap allows multiple values to be associated with a single key. To use Multimap in Java, **you need to add the Guava library to your project.**

In which situation the method should be static and when non-static?

In Java, methods can be declared as static or non-static.

A static method belongs to a class and can be called without creating an instance of the class. Non-static methods, on the other hand, belong to an instance of a class and can only be called on that instance.

Here are some scenarios in which a method should be declared static:

1. Utility methods: Methods that perform general-purpose tasks, such as sorting an array or formatting a date, can be declared static because they do not require any specific instance variables.
2. Constants: If a method returns a constant value that does not depend on any instance variables, it can be declared as static.
3. Factory methods: Methods that create objects of a class can be declared static so that they can be called without creating an instance of the class.
4. Main method: The main method is the entry point of a Java program and needs to be declared as static.

On the other hand, non-static methods are useful in the following scenarios:

1. Instance-specific methods: Methods that operate on data specific to each instance of a class should be declared non-static.
2. Accessor methods: Getters and setters that allow access to instance variables should be non-static.
3. Constructors: Constructors create new instances of a class and cannot be declared static.

Here is an example code snippet demonstrating static and non-static methods:

```
public class MyClass {
```

```
    private int myNumber;
    public static void myStaticMethod() {
        // Static method implementation
    }
    public void myNonStaticMethod() {
        // Non-static method implementation
    }
    public int getMyNumber() {
        return this.myNumber;
    }
    public void setMyNumber(int myNumber) {
        this.myNumber = myNumber;
    }
}
// Calling a static method:
MyClass.myStaticMethod();
// Creating an instance and calling a non-static method:
MyClass obj = new MyClass();
obj.myNonStaticMethod();
// Accessing instance variables through accessor methods:
obj.setMyNumber(42);
int number = obj.getMyNumber();
```


Suppose you have class and abstract class in class there is a user defined constructor and main method which one will get executed first?

The user-defined constructor will get executed first before the main method in Java.
Here is an example code to demonstrate it:

```
abstract class AbstractExample {
    public AbstractExample() {
        System.out.println(""Abstract constructor has been called"");
    }
    public abstract void abstractMethod();
}
public class Example extends AbstractExample {
    public Example() {
        super();
        System.out.println(""User-defined constructor has been called"");
    }
    public static void main(String[] args) {
        Example example = new Example();
        example.abstractMethod();
    }
    @Override
    public void abstractMethod() {
        System.out.println(""Abstract method implementation"");
    }
}
```

In the above code, we have an abstract class `AbstractExample` with a user-defined constructor and an abstract method. The `Example` class extends the `AbstractExample` class and has its own user-defined constructor and `main` method.
When we create an instance of the `Example` class in the `main` method, the constructor of the `Example` class is called first, which in turn calls the constructor of the `AbstractExample` class. Finally, the `abstractMethod` is called and its implementation is printed to the console.
Therefore, we can conclude that the user-defined constructor is executed first in Java.

What do you mean by POJO why we use POJO?

POJO stands for Plain Old Java Object. It is a Java class that contains only private instance variables, constructors, getters, and setters without any business logic, framework-specific code, or other dependencies. POJOs provide a simple and flexible way to encapsulate data and represent objects in a system.
POJOs are widely used in Java development, especially in enterprise applications, because they are easy to create, understand, and maintain. They enable developers to focus on business logic, rather than worrying about framework-specific complexities or dependencies. POJOs also promote loose coupling, as they can be used independently of any framework or container.
Here is an example of a POJO class in Java:

```
public class Customer {
   private int id;
   private String name;
   private String address;
   public Customer() {
   }
   public Customer(int id, String name, String address) {
      this.id = id;
      this.name = name;
      this.address = address;
   }
   public int getId() {
      return id;
   }
   public void setId(int id) {
      this.id = id;
   }
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public String getAddress() {
      return address;
   }
   public void setAddress(String address) {
      this.address = address;
   }
}
```

In this class, we have declared three private instance variables and created constructors, getters, and setters to access and modify the values of these variables. This class can be used to represent customers in a system, without any framework-specific code or dependencies.

What is the difference between a checked and unchecked exceptions?

In Java, there are two types of exceptions: checked exceptions and unchecked exceptions.
Checked exceptions are checked at compile-time. This means that the Java compiler will check to see if the code handles or declares the checked exception. If the checked exception is not handled or declared, the code will not compile.
Here is an example of a checked exception:

```
public void readFile() throws FileNotFoundException {
   File file = new File(""file.txt"");
   Scanner scanner = new Scanner(file);
}
```


In this example, the `FileNotFoundException` is a checked exception. The code throws this exception because the file ""file.txt"" may not exist. The method declares this exception, but it does not handle it. If this method is called, the calling method must either handle the exception with a `try-catch` block or declare the exception with the `throws` keyword.

Unchecked exceptions, also known as runtime exceptions, are not checked at compile-time. This means that the Java compiler does not require the code to handle or declare unchecked exceptions. These exceptions are typically caused by programming errors, such as dividing by zero or trying to access an array index that is out-of-bounds.

Here is an example of an unchecked exception:

```
public void divide(int a, int b) {
   int result = a / b;
}
```


In this example, if `b` is equal to zero, the code will throw an `ArithmeticException`. This is an unchecked exception that does not need to be handled or declared.
In summary, the main difference between checked and unchecked exceptions is that checked exceptions are checked at compile-time, while unchecked exceptions are not checked at compile-time.

Difference between Array and ArrayList?

In Java, an array is a fixed-size container that can store a fixed number of elements of the same data type, while an ArrayList is a dynamically resizable container that can store any number of elements of any data type.
For example, to declare an array of integers with a size of 5, you would use this code:

```
int[] myArray = new int[5];
```

To add elements to the array, you would need to specify the index position, like this:

```
myArray[0] = 1;
myArray[1] = 2;
myArray[2] = 3;
myArray[3] = 4;
myArray[4] = 5;
```

An ArrayList, on the other hand, can be declared like this:

```
ArrayList<Integer> myList = new ArrayList<Integer>();
```

To add elements to the ArrayList, you can use the `.add()` method:

```
myList.add(1);
```

```
myList.add(2);
myList.add(3);
myList.add(4);
myList.add(5);
```

One of the main differences between arrays and ArrayLists is that arrays are fixed in size, while ArrayLists can change in size dynamically. This means that if you need to add more elements to an array, you would need to create a new array with a larger size and copy the elements from the old array to the new array. With an ArrayList, you can simply use the `.add()` method to add more elements.

Another difference is that arrays can store primitive data types (such as int or char), while **ArrayLists can only store objects.** To store a primitive data type in an ArrayList, you would need to use a wrapper class (such as Integer or Character).

Can we call a non-static variable in static method?

No, we cannot call a non-static variable in a static method directly. We need to create an instance of the class and use that instance to access the non-static variable. Here is an example:

```
public class Example {
  int nonStaticVar = 10; // non-static variable
  static void staticMethod() {
    Example obj = new Example(); // create an instance of the class
    System.out.println(obj.nonStaticVar); // access the non-static variable using the instance
  }
}
```

In this example, we have a non-static variable `nonStaticVar` and a static method `staticMethod`. We cannot access `nonStaticVar` directly in `staticMethod`. So, we create an instance of the class using `Example obj = new Example()` and then access `nonStaticVar` using `obj.nonStaticVar`.

What is an abstract modifier?

In Java, the abstract modifier is used to indicate that a class, method or variable is incomplete or not fully defined.

An abstract class in Java is a class that cannot be instantiated and is intended to be subclassed by other classes. An abstract method is a method that is declared but not implemented in an abstract class. The implementation of the method is left to the subclasses that inherit from the abstract class.

Here is an example of an abstract class with an abstract method in Java:

```
public abstract class Shape {
  public abstract double getArea();
}
```

This indicates that the class ""Shape"" is incomplete and that any subclass that inherits from it must implement the ""getArea()"" method.

Here is an example of a subclass of the abstract class ""Shape"" that implements the abstract method:

```
public class Circle extends Shape {
  private double radius;

  public Circle(double radius) {
    this.radius = radius;
  }

  public double getArea() {
```

```
    return Math.PI * radius * radius;
  }
}
```

By implementing the ""getArea()"" method, the Circle class becomes a concrete class that can be instantiated.

############################ MANAS JHA ############################

LIST DECLARATION?

```
import java.util.List;
import java.util.ArrayList;

List<Type> listName = new ArrayList<>();
```

Here, `Type` specifies the type of elements that the List will contain, and `listName` is the name of the List variable. The implementation class used to create the List is `ArrayList,` which is a commonly used implementation of the `List` interface.

You can use other implementations of the `List` interface, such as `LinkedList` or `Vector`, depending on your specific requirements. The basic syntax for declaring a `List` remains the same, you just need to replace the implementation class with the desired one

############################ MANAS JHA ############################

IF NOT WANT TO USE STRING CLASS THEN WHAT CAN BE USED?

- **`char`** array: A **`char`** array can be used to store a sequence of characters. You can manipulate the **`char`** array just like a **`String`**, but without the convenience methods provided by the **`String`** class.
- **`StringBuilder`**: The **`StringBuilder`** class is a mutable sequence of characters. You can use it to build up a string incrementally, and it's more efficient than concatenating strings using the **`+`** operator.
- **`StringBuffer`**: The **`StringBuffer`** class is similar to **`StringBuilder`**, but it's synchronized, making it thread-safe. This means that it is slower than **`StringBuilder`**, but it's a good choice if you need to build strings in a multithreaded environment.
- **`Byte[]`**: You can use a **`byte`** array to store binary data, including text encoded in a particular character encoding. The **`String`** class can be used to convert the **`byte`** array to a **`String`** and vice versa.

############################ MANAS JHA ############################

WHERE USED SET?

In automation, Set is commonly used in several scenarios:

Data Validation: You can use a Set to store expected values and compare it with actual values to validate that the data is correct.

Removing duplicates: Set doesn't allow duplicate values, so you can use it to remove duplicates from a list of elements.

Storing unique elements: If you need to store a collection of unique elements, you can use a Set.

Testing: You can use a Set to store the elements you need to test, and use its methods to check if an element is present in the Set or not.

Performance Optimization: Set has fast lookup performance, so it's a good choice if you need to check if an element is present in a large collection of elements frequently.

These are some of the common uses of Set in automation. However, the specific use case may vary based on the requirements of the automation project.

########## MANAS JHA ##########

I HAVE A TABLE AND WANT TO STORE ALL TABLE DATA THEN WHICH COLLECTION SHOULD BE USE AND WHY?

The best collection to store table data in Java depends on the specific requirements of your application.

If you have a table with fixed number of columns and rows, it is best to use a 2D array.
If you have a dynamic number of columns and rows, then ArrayList of ArrayList or HashMap of HashMap can be used.
If you need to access the data using keys, then a HashMap is a good choice. You can store the data in key-value pairs where the key is a unique identifier and the value is the data you want to store.
If you need to maintain the order of the data, you can use a LinkedHashMap.
If you need a thread-safe data structure, you can use a Hashtable.
Ultimately, the best collection to use will depend on your specific requirements, including the size of the data, the type of data, and the operations you need to perform on the data.

########## MANAS JHA ##########

WHAT HASHMAP WILL RETURN?

A HashMap in Java is a class that implements the Map interface and stores data in key-value pairs. When a get() method is called on a HashMap, it returns the value associated with the specified key. If the key is not present in the HashMap, the get() method returns null.

For example:

```
HashMap<String, Integer> map = new HashMap<>();
map.put("A", 1);
map.put("B", 2);
map.put("C", 3);

System.out.println(map.get("A")); // Outputs 1
System.out.println(map.get("D")); // Outputs null
```

It's important to note that the keys in a HashMap must be unique, otherwise, the last value associated with a duplicate key will overwrite the previous value

WRITE A PROGRAM TO MERGE TWO SORTED ARRAYS?

```
public static int[] mergeArrays(int[] arr1, int[] arr2) {
   int[] mergedArray = new int[arr1.length + arr2.length];
   int i = 0, j = 0, k = 0;
   while (i < arr1.length && j < arr2.length) {
      if (arr1[i] < arr2[j]) {
         mergedArray[k++] = arr1[i++];
      } else {
         mergedArray[k++] = arr2[j++];
      }
   }
   while (i < arr1.length) {
      mergedArray[k++] = arr1[i++];
   }
   while (j < arr2.length) {
      mergedArray[k++] = arr2[j++];
   }
   return mergedArray;
}

public static void main(String[] args) {
   int[] array1 = {1, 3, 5, 7, 9};
   int[] array2 = {2, 4, 6, 8, 10};
   int[] mergedArray = mergeArrays(array1, array2);
   System.out.println(Arrays.toString(mergedArray));
}
```


HOW TO RESOLVED CONFLICTS WHILE PUSHING CODE IN GIT?

There are a few ways to resolve conflicts in Git:

1. Resolve the conflicts manually by editing the files with the conflicting changes and marking the changes as resolved.

2. Use the git rebase command to re-apply commits that caused the conflict.

3. Use the git merge tool command to automatically resolve the conflicts.

DO YOU KNOW OOPS CONCEPTS AND IN FRAMEWORK WHERE AND HOW YOU HAVE IMPLEMENTED IT?

Java Object-Oriented Concepts in Selenium Automation Framework

INTERFACE

An interface in the Java programming language is an abstract type that is used to specify a behaviour that classes must implement. An interface also contains methods and variables just like the class but the methods declared in the interface are by default abstract.

To understand this the very basic statement we write in Selenium

```
WebDriver driver = new Chromedriver();
```

In this case, WebDriver itself is an Interface. So based on this statement, WebDriver driver = new Chromedriver(); we are initializing chrome browser using Selenium WebDriver. It means we are creating a reference variable (driver) of the interface (WebDriver) and creating an Object. Here WebDriver is an Interface as mentioned earlier and Chromedriver is a class.

ABSTRACTION

As we are aware, Abstraction is a process of hiding the implementation details from the user and showing only relevant details to them. It also helps to reduce programming complexity and effort.

```
package Page_Object;
	import org.openqa.selenium.WebDriver;
	import org.openqa.selenium.WebElement;
	import org.openqa.selenium.support.FindBy;
	import org.openqa.selenium.support.How;
	public class login_page_object extends BaseClass{

		public login_page_object(WebDriver driver) {
			super(driver);
		}
		@FindBy(how=How.XPATH, using="//a[contains(text(),'Sign in')]")
		public static WebElement signInBtn;

		@FindBy(how=How.XPATH, using="//input[@id='email' and @name='email']")
		public static WebElement emailTxtBox;

		@FindBy(how=How.XPATH, using="//input[@id='passwd']")
		public static WebElement pswdTxtBox;
		@FindBy(how=How.XPATH, using="//button[@id='SubmitLogin']//span")
		public static WebElement submitBtn;

		@FindBy(how=How.XPATH, using="//a[@title='View my customer account']/span")
		public static WebElement usernameLnk;


	}
```

In our Automation Framework whenever we Use the Page object Model, we write all the locators in the page class and use this locator in our test it means we are hiding our implementation from the user this is a simple example of using abstraction in the framework.

INHERITANCE

The process by which one class acquires the properties (instance variables) and functionalities of another class are called inheritance.
When We create a Base Class in our automation Framework to initialize WebDriver interface, waits, loggers, reports, etc., and when we extend this Base Class in other classes such as Tests and Utility Class. In this case, extending one class into another class is an example of implementing Inheritance.

```
package Page_Object;
	import org.openqa.selenium.WebDriver;


	public class BaseClass {
		public static WebDriver driver;
		 public static boolean bResult;
		 public BaseClass(WebDriver driver){
			BaseClass.driver = driver;
			BaseClass.bResult = true;
			}
	}
```

ENCAPSULATION

Encapsulation is a mechanism of wrapping data (variables) and code together as a single unit. All the classes which we write in our automation framework are an example of Encapsulation. For e.g In Page object model classes, in which we declare the WebElement locator using @FindBy and initialization of this data members will be done using Constructor to utilize those in test methods.

```
package Page_Object;
	import org.openqa.selenium.WebDriver;
	import org.openqa.selenium.WebElement;
	import org.openqa.selenium.support.FindBy;
	import org.openqa.selenium.support.How;

	public class login_page_object extends BaseClass{

		public login_page_object(WebDriver driver) {

		super(driver);

		}

		@FindBy(how=How.XPATH, using="//a[contains(text(),'Sign in')]")
		public static WebElement signInBtn;

		@FindBy(how=How.XPATH, using="//input[@id='email' and @name='email']")
		public static WebElement emailTxtBox;
```

```
        @FindBy(how=How.XPATH, using="//input[@id='passwd']")
        public static WebElement pswdTxtBox;

        @FindBy(how=How.XPATH, using="//button[@id='SubmitLogin']//span")
        public static WebElement submitBtn;

        @FindBy(how=How.XPATH, using="//a[@title='View my customer account']/span")
        public static WebElement usernameLnk;


    }
```

POLYMORPHISM

Polymorphism means having many forms. Polymorphism allows us to perform a single action in different ways. In Java polymorphism can be achieved in two ways:

METHOD OVERLOADING

If a class has multiple methods having the same name but different in parameters, it is known as **Method Overloading**.

In Implicit wait when we use different time stamps such as SECONDS, MINUTES, HOURS, etc is one of the possible examples of method overloading.

```
@BeforeMethod
    public void setUp() throws Exception {
    driver = new FirefoxDriver();
    baseUrl = "http://www.google.com";


    //Method Overloading any one of the below Mentioned implicitlyWait can be used
    driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
    driver.manage().timeouts().implicitlyWait(30, TimeUnit.MINUTES);
    driver.manage().timeouts().implicitlyWait(30, TimeUnit.HOURS);
    }
```

METHOD OVERRIDING

If the subclass or child class has the same method as declared in the parent class, it is known as method overriding in Java.

Whenever we use a method that was already implemented/written in another class by changing its parameters this is the example of method overriding.

########################## MANAS JHA ##########################

CAN WE DECLARE A PRIVATE CLASS?

Yes, we can declare a class as private but these classes can be only inner or nested classes. We can't a top-level class as private because it would be completely useless as nothing would have access to it.

Example 1 with non inner class:

```
private class Main
{
        public static void main(String[] args) {
                System.out.println("Inside private class");
        }
}
```

Output:

Main.java:1: error: modifier private not allowed here

private class Main

    ^

1 error

Example 2 with non inner class:

```
private class Show{
    void display(){
      System.out.println("Inside display method.");
    }
}
public class Main
{
        public static void main(String[] args) {
                Show show = new Show();
                show.display();
        }
}
```

Output:

Main.java:1: error: modifier private not allowed here

private class Show{

    ^

1 error

Example with inner class:

```
class Display {

  //Private nested or inner class
  private class InnerDisplay {
    public void display() {
      System.out.println("Private inner class method called");
    }
  }

  void display() {
```

```
    System.out.println("Outer class (Display) method called");
    // Access the private inner class
    InnerDisplay innerDisplay = new InnerDisplay();
    innerDisplay.display();
  }
}

public class Main {

  public static void main(String args[]) {
    // Create object of the outer class (Display)
    Display object = new Display();

    // method invocation
    object.display();
  }
}
```

Output:

Outer class (Display) method called

Private inner class method called

########## MANAS JHA ##########

Difference Between == and. equals() Method in Java

In java both == and equals() method is used to check the equality of two variables or objects.

Following are the important differences between == and equals() method.

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

Why String is Immutable?

A String is an unavoidable type of variable while writing any application program. String references are used to store various attributes like username, password, etc. In Java, **String objects are immutable**. Immutable simply means unmodifiable or unchangeable.

Once String object is created its data or state can't be changed but a new String object is created.

Let's try to understand the concept of immutability by the example given below:

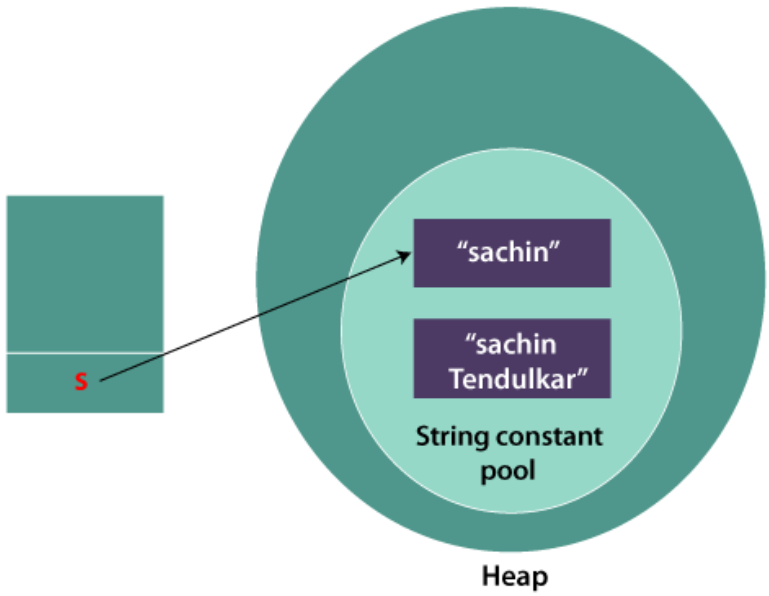
Testimmutablestring.java

```
class Testimmutablestring{
        public static void main(String args[]){
          String s="Sachin";
          s.concat(" Tendulkar");//concat() method appends the string at the end
          System.out.println(s);//will print Sachin because strings are immutable objects
 }
        }
```

Output:

Sachin

Now it can be understood by the diagram given below. Here Sachin is not changed but a new object is created with Sachin Tendulkar. That is why String is known as immutable.



As you can see in the above figure that two objects are created but *s* reference variable still refers to "Sachin" not to "Sachin Tendulkar".

But if we explicitly assign it to the reference variable, it will refer to "Sachin Tendulkar" object.

For example:

Testimmutablestring1.java

```
class Testimmutablestring1{
 public static void main(String args[]){
   String s="Sachin";
   s=s.concat(" Tendulkar");
   System.out.println(s);
 } }
```

Output:

Sachin Tendulkar

In such a case, s points to the "Sachin Tendulkar". Please notice that still Sachin object is not modified.

Why String objects are immutable in Java?

As Java uses the concept of String literal. Suppose there are 5 reference variables, all refer to one object "Sachin". If one reference variable changes the value of the object, it will be affected by all the reference variables. That is why String objects are immutable in Java.

Following are some features of String which makes String objects immutable.

1. ClassLoader:

A ClassLoader in Java uses a String object as an argument. Consider, if the String object is modifiable, the value might be changed and the class that is supposed to be loaded might be different.

To avoid this kind of misinterpretation, String is immutable.

2. Thread Safe:

As the String object is immutable we don't have to take care of the synchronization that is required while sharing an object across multiple threads.

3. Security:

As we have seen in class loading, immutable String objects avoid further errors by loading the correct class. This leads to making the application program more secure. Consider an example of banking software. The username and password cannot be modified by any intruder because String objects are immutable. This can make the application program more secure.

4. Heap Space:

The immutability of String helps to minimize the usage in the heap memory. When we try to declare a new String object, the JVM checks whether the value already exists in the String pool or not. If it exists, the same value is assigned to the new object. This feature allows Java to use the heap space efficiently.

Why String class is Final in Java?

The reason behind the String class being final is because no one can override the methods of the String class. So that it can provide the same features to the new String objects as well as to the old ones.

HOW TO FIND MISSING IMPLEMENTATION IN CUCUMBER?

There are a few ways to find missing implementation in Cucumber:

UndefinedStepException: Cucumber will throw an **UndefinedStepException** when it encounters a step definition that is not implemented. This will give you a clear indication of which steps need to be implemented.

PendingException: You can use the **PendingException** in your step definitions to temporarily mark steps as pending. When Cucumber encounters a step marked with a **PendingException**, it will consider the scenario as pending and print a message indicating that the step is pending.

Dry Run: You can run Cucumber with the **--dry-run** option, which will check all of your feature files and step definitions for missing or ambiguous steps without actually executing the tests. This can be a quick way to identify any missing or incorrect step definitions.

Reporting: Cucumber provides a number of reporting options, including the ability to generate reports in HTML, JSON, or JUnit format. These reports can give you an overview of which scenarios passed, failed, or are pending, which can help you identify any missing implementations.

HOW TO STORE MULTIPLE VALUES IN ONE REFERENCE?

HashMap – Single Key and Multiple Values Using List

```
package com.skilledmonster.examples.util.map;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;
```

```
/**
 * HashMap - Single Key and Multiple Values using List
 *
**/
public class SingleKeyMultipleValueUsingList {
  public static void main(String[] args) {
      // create map to store
      Map<String, List<String>> map = new HashMap<String, List<String>>();
      // create list one and store values
      List<String> valSetOne = new ArrayList<String>();
      valSetOne.add("Apple");
      valSetOne.add("Aeroplane");
      // create list two and store values
      List<String> valSetTwo = new ArrayList<String>();
     valSetTwo.add("Bat");
      valSetTwo.add("Banana");
      // create list three and store values
      List<String> valSetThree = new ArrayList<String>();
     valSetThree.add("Cat");
      valSetThree.add("Car");
      // put values into map
      map.put("A", valSetOne);
      map.put("B", valSetTwo);
      map.put("C", valSetThree);
      // iterate and display values
      System.out.println("Fetching Keys and corresponding [Multiple] Values n");
      for (Map.Entry<String, List<String>> entry : map.entrySet()) {
         String key = entry.getKey();
         List<String> values = entry.getValue();
```

```
        System.out.println("Key = " + key);
        System.out.println("Values = " + values + "n");
      }
   }
}
```

########## MANAS JHA ##########

DIFFERENCE BETWEEN STRING AND STRINGBUFFER.

StringBuffer:

StringBuffer is a peer class of String that provides much of the functionality of strings. The string represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences.

Syntax:

```
StringBuffer s = new StringBuffer("Manas Jha automation classes");
```

StringBuilder:

The StringBuilder in Java represents a mutable sequence of characters. Since the String Class in Java creates an immutable sequence of characters, the StringBuilder class provides an alternate to String Class, as it creates a mutable sequence of characters.

Syntax:

```
StringBuilder str = new StringBuilder();
str.append("Manas jha");
```

########## MANAS JHA ##########

Static keyword

In Java, the static keyword is used to create a class-level variable or method that can be accessed without creating an instance of the class. When a member of a class is declared as static, it belongs to the class and not to any particular instance of the class.

Here are some common uses of the static keyword in Java:

Static variables:

Static variables are shared among all instances of a class. They are initialized only once when the class is loaded into memory and exist throughout the lifetime of the program. They can be accessed using the class name, without the need for an instance of the class. For example:

```
class MyClass {
   static int count = 0;
   int id;

   MyClass() {
      id = count;
      count++;
   }
}

public class Main {
   public static void main(String[] args) {
      MyClass obj1 = new MyClass();
      MyClass obj2 = new MyClass();

      System.out.println(obj1.id); // Output: 0
      System.out.println(obj2.id); // Output: 1

      System.out.println(MyClass.count); // Output: 2
   }
}
```

In the above example, we have a MyClass with a static variable count that is incremented each time a new instance of the class is created. The id variable is assigned the current value of count when an instance is created. We can access the count variable using the class name MyClass.count without creating any instances of the class.

Static methods:

Static methods are associated with the class rather than any instance of the class. They can be called using the class name and can access only static variables and other static methods. For example:

```
class MyClass {
   static void printMessage() {
      System.out.println("Hello, World!");
   }
}

public class Main {
   public static void main(String[] args) {
      MyClass.printMessage(); // Output: Hello, World!
   }
}
```

In the above example, we have a MyClass with a static method printMessage(). We can call this method using the class name MyClass.printMessage() without creating any instances of the class.

Static blocks:

Static blocks are used to initialize static variables or to perform any other static initialization tasks. They are executed only once when the class is loaded into memory, before any static methods or variables are accessed. For example:

```
class MyClass {
   static int count;

   static {
      count = 10;
   }
}

public class Main {
   public static void main(String[] args) {
      System.out.println(MyClass.count); // Output: 10
   }
}
```


In the above example, we have a MyClass with a static block that initializes the static variable count to 10 when the class is loaded into memory. We can access the count variable using the class name MyClass.count without creating any instances of the class.

Java 8 introduced several new features and improvements to interfaces. Here are some of the key updates:

Default methods: Interfaces can now include default method implementations, which are methods that are already implemented within the interface itself. This allows developers to add new functionality to interfaces without breaking existing implementations.
Example:


```
public interface MyInterface {
  default void myMethod() {
    System.out.println("This is a default method.");
  }
}
```

Static methods: Interfaces can now have static methods, which are methods that can be called without creating an instance of the interface.
Example:

```
public interface MyInterface {
  static void myStaticMethod() {
    System.out.println("This is a static method.");
  }
}
```

Functional interfaces: Java 8 introduced functional interfaces, which are interfaces that have only one abstract method. These interfaces can be used with lambda expressions to create instances of the interface.
Example:

```
@FunctionalInterface
public interface MyFunctionalInterface {
```

```
  void myMethod();
}
```

Optional methods: Interfaces can also define methods that are optional to implement. These methods are marked with the @Deprecated annotation, which indicates that they may be removed in future versions of the interface.

Example:

```
public interface MyInterface {
  default void myMethod() {
    System.out.println("This is a default method.");
  }

  @Deprecated
  void myOptionalMethod();
}
```

Overall, these updates to interfaces in Java 8 have made them much more versatile and powerful, allowing developers to write more expressive and concise code.

SELENIUM

Without implicit wait selenium script will work or not?

The implicit wait in Selenium is used to set the maximum time for the browser to wait for an element to appear before throwing an error. If implicit wait is not set, the script may fail because the browser may not have enough time to locate the element before moving on to the next step.

Here's an example of setting an implicit wait in Java:

WebDriver driver = new ChromeDriver();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

This code sets an implicit wait of 10 seconds for the Chrome driver. If an element is not located within 10 seconds, the script will throw an error.

Difference between Factory design and Singleton framework?

Factory design pattern and Singleton pattern are both creational design patterns in Java. They differ in their purpose and implementation.

The **Factory design pattern** is used when you want to create objects without exposing the creation logic to the client. In this pattern, we create an interface or abstract class for creating objects, and subclasses are responsible for creating objects of their type. The client uses the interface to create objects, and then we can change the implementation of subclasses without affecting the client code.

Here is an example of implementing the Factory design pattern in Java:


```
interface Animal {
void speak();
}

class Dog implements Animal {
@Override
public void speak() {
System.out.println(""Woof!"");
}
}

class Cat implements Animal {
@Override
public void speak() {
System.out.println(""Meow!"");
}
}

class AnimalFactory {
public static Animal createAnimal(String type) {
if (type.equalsIgnoreCase(""dog"")) {
return new Dog();
```

```
} else if (type.equalsIgnoreCase(""cat"")) {
return new Cat();
} else {
throw new IllegalArgumentException(""Animal type not supported."");
}
}
}
```

The **Singleton pattern,** on the other hand, is used when we need to restrict the creation of a class to only one instance and provide global access to that instance. In this pattern, we make the constructor of the class private and provide a static method to get the instance of the class.

Here is an example of implementing the Singleton pattern in Java:

```
class Singleton {
private static Singleton instance;

private Singleton() {
// Private constructor to prevent instantiation from outside the class
}
public static Singleton getInstance() {
if (instance == null) {
instance = new Singleton();
}
return instance;
}
}
```


In short, the Factory pattern is used for creating objects, while the Singleton pattern is used for ensuring that only one instance of a class exists.

How do you achieve inheritance in your framework?

Inheritance can be achieved in a framework by creating a base class that contains common methods and properties, and then creating child classes that inherit these methods and properties.

For example, in a Selenium WebDriver framework, we can create a base class named ""BasePage"" that contains common methods like "penPage"" and ""closePage"". Then we can create child classes like ""LoginPage"" and ""HomePage"" that inherit the methods from the ""BasePage"" class.

Here is an example code snippet in Java:

```
public class BasePage {
public WebDriver driver;

public BasePage(WebDriver driver) {
```

```
this.driver = driver;
}

public void openPage(String url) {
driver.get(url);
}

public void closePage() {
driver.quit();
}
}

public class LoginPage extends BasePage {
public LoginPage(WebDriver driver) {
super(driver);
}
public void login(String username, String password) {
// code to enter username and password
}
}

public class HomePage extends BasePage {
public HomePage(WebDriver driver) {
super(driver);
}

public void clickButton(String buttonName) {
// code to click a button
}
}
```


In this example, the BasePage class contains the WebDriver instance and common methods like "penPage"" and ""closePage"". The LoginPage and HomePage classes inherit these methods and properties from the BasePage class, and also contain methods specific to their respective pages.

What's the fastest locator in Selenium?

The fastest locator in Selenium is usually considered to be the ID locator, as it directly matches the ID attribute of the element in the HTML code. However, this is dependent on the page structure and the IDs assigned to the elements. In some cases, using other locators like CSS or XPath may be faster or more efficient.

Here is an example of using the ID locator in Selenium WebDriver using Java:

```
java
// Assuming we have a WebDriver instance named driver
WebElement element = driver.findElement(By.id(""elementID""));
element.click(); // Click on the element
```

In this example, we are finding an element with the ID ""elementID"" using the By.id locator, and then performing a click operation on it using the WebElement.click() method.

What does :: (doubles colon) in sibling xpaths represent?

The:: (double colon) in sibling xpaths **is an axis selector**. It allows you to navigate to specific nodes relative to the current node. For example, the following xpath selects all the siblings that come after the current node:

//*[@id='currentNode']/following-sibling::*

In Java, you can use the Selenium webdriver to locate elements using xpath. Here's an example of using xpath to select a sibling element:

WebElement siblingElement = driver.findElement(By.xpath(""//*[@id='currentNode']/following-sibling::*""));

What is difference between get() and navigate().to() in Selenium?

The difference between get() and navigate().to() methods in Selenium is:

- **get():** This method is used to load a new web page in the current browser window. It waits for the page to load completely before returning the control to the test script.

Example:
WebDriver driver = new ChromeDriver();
driver.get(""https://www.google.com/"");

- **navigate().to():** This method navigates the browser to a new web page but it does not wait for the page to load completely before returning the control to the test script.

Example:

WebDriver driver = new ChromeDriver();
driver.navigate().to(""https://www.google.com/"");

Return type of findelement and findelements?

The return type of findelement method in Selenium WebDriver is **WebElement,** which represents an HTML element on the web page. The return type of findelements method is a List of WebElement, which represents multiple HTML elements on the web page.

For example, to find a single element by its id, we can use the following code:

WebElement element = driver.findElement(By.id(""element-id""));

To find multiple elements by their class name, we can use the following code:

List<WebElement> elements = driver.findElements(By.className(""element-class""));

Types of Exceptions and how to handle stale element exception?

There are two types of exceptions in Java: Checked and Unchecked exceptions. Checked exceptions are checked at compile time and Unchecked exceptions are checked at runtime.

StaleElementReferenceException is a type of Unchecked exception which is thrown when the web element being interacted with is no longer attached to the DOM. This can happen when the page refreshes or changes after the element is located but before it can be interacted with.

To handle **StaleElementReferenceException**, we can use the try-catch block. Here's an example code snippet:

```
try {
WebElement element = driver.findElement(By.xpath(""//input[@name='email']""));
element.sendKeys(""example@email.com"");
} catch (StaleElementReferenceException e) {
WebElement element = driver.findElement(By.xpath(""//input[@name='email']""));
element.sendKeys(""example@email.com"");
}
```


In the above code, we try to locate and interact with an element. If a StaleElementReferenceException is thrown, the catch block will locate the element again and perform the desired action. This ensures that our test doesn't fail due to an exception caused by a stale element.

In Selenium, one common exception that you may encounter is the **StaleElementReferenceException**. This exception occurs when an element that was previously found by the WebDriver becomes stale, or no longer exists in the DOM.

To handle the StaleElementReferenceException in Selenium, you can use one of the following methods:

Retry: You can try to find the element again by re-executing the same code that originally found the element. This method can be useful if the element is likely to become available again quickly.

Wait: You can use an explicit wait to wait for the element to become available again. This can be done using the "ExpectedConditions" class in Selenium, which provides a variety of conditions that can be used to wait for an element to become visible, clickable, etc.

Refresh the page: You can try to refresh the page to reload the element. This can be done using the WebDriver's "navigate().refresh()" method.

To avoid the StaleElementReferenceException in Selenium, you can use the following best practices:

Avoid caching elements: Do not store elements in variables and reuse them later. Instead, find the element each time you need to interact with it.

Use explicit waits: Use explicit waits to wait for the element to become available before interacting with it. This can help avoid the StaleElementReferenceException.

Avoid interacting with elements that may become stale: If you know that an element is likely to become stale, try to interact with a different element or wait until the element becomes available again.

Use stable locators: Use locators that are unlikely to change, such as IDs or unique attributes, to find elements. This can help avoid the StaleElementReferenceException caused by changes in the DOM.

In a web page, there are several Pop-up, but we don't when the pop-up will appear, in this case how we will handle the Pop-up using Selenium WebDriver (Java)

To handle Pop-ups using Selenium WebDriver (Java), you can use the Alert interface provided by WebDriver API. You can use the following code to handle Pop-ups:

```
// Switch to Pop-up
Alert alert = driver.switchTo().alert();

// Get the text from Pop-up
String message = alert.getText();

// Dismiss the Pop-up
alert.dismiss();

// Accept the Pop-up
alert.accept();

// Enter text in the Pop-up input
alert.sendKeys(""Text"");

// Switch back to the main window
driver.switchTo().defaultContent();

//Wait for Pop-up to appear
WebDriverWait wait = new WebDriverWait(driver,10);
wait.until(ExpectedConditions.alertIsPresent());
```


In the above code, we first switch to the alert by using switchTo() method and Alert interface. Once we switch to the alert, we can get alert text, dismiss or accept the alert, enter text in the alert input, or switch back to the main window using methods provided by the Alert interface.

WebDriverWait is used to wait for the alert to appear on the screen if it is not already present. This will help prevent exceptions and make our code more robust.

Other Way

How to handle file upload when type attribute does not file for upload web element.

In order to handle file upload when the type attribute does not specify a file upload web element, we can use the sendKeys() method of the WebElement class. This method allows us to send the path of the file we want to upload to the corresponding input field.

Here's an example of how to handle file upload using sendKeys() method in Java:

java
WebElement fileInput = driver.findElement(By.xpath("//input[contains(@class,'file-upload')]"));
fileInput.sendKeys("C:\Users\Documents\example.csv");

In this example, we first locate the input element that we want to use for file upload using the findElement() method. Then, we use the sendKeys() method to send the path of the file we want to upload to this element.

It is important to note that the path we provide to sendKeys() should be a valid path to the file on our local machine. Also, the input element should have the necessary permissions to access the file, and the file size should not exceed any limits set by the server.

How to cover character keyboard operation from the context menu utilizing user-defined keyword?

To cover character keyboard operation from the context menu utilizing user-defined keyword, we can use Selenium with Java. Here is an example code:

First, we need to define a user-defined keyword to perform the required action:


```
public void keyboardCharacterOperationByContextMenu(String character) {
Actions actions = new Actions(driver);
actions.sendKeys(character).perform();
actions.contextClick().perform();
actions.sendKeys(Keys.ARROW_DOWN).sendKeys(Keys.ARROW_DOWN).sendKeys(Keys.ENTER).perform();
}
```


In this method, we are using the `Actions` class to simulate keyboard operations. We first simulate typing the required character using the `sendKeys()` method. Then, we simulate right-clicking on the element using the `contextClick()` method. This should open the context menu. We then simulate navigating through the context menu by pressing the arrow keys using the `sendKeys()` method again. Finally, we select the desired option from the context menu by pressing the Enter key.

Here's an example usage of this user-defined keyword:

```
WebElement element = driver.findElement(By.id(""myInput""));
keyboardCharacterOperationByContextMenu(""a"");
```

In this example, we are finding an input element and passing the character `a` to our user-defined keyword to simulate right-clicking on the element and selecting the option to enter `a` in the input field.

Note: This is just one way to achieve this functionality. Depending on the webpage and its elements, the code may need to be modified accordingly.

########## MANAS JHA ##########

Where can “Dynamic Polymorphism” in Selenium WebDriver be useful?

Dynamic Polymorphism in Selenium WebDriver can be useful in situations where there are multiple web elements on a page that have similar properties and behaviors. By using dynamic polymorphism, we can create a single set of code that can work with different web elements at runtime, depending on their specific properties.

For example, let's say we have multiple button elements on a web page with similar attributes such as class name, value, etc. Instead of writing separate code for each button element, we can create a generic method that takes an argument for the specific button element and performs the required action. This approach allows us to write more reusable and maintainable code.

Here's an example of how we could use dynamic polymorphism in Selenium WebDriver with Java:

MANAS JHA

```
public class DynamicPolymorphismExample {
WebDriver driver = new ChromeDriver();

public void clickButton(WebElement element) {
element.click();
}
```

MANAS JHA

```
public static void main(String[] args) {
DynamicPolymorphismExample example = new DynamicPolymorphismExample();
example.driver.get(""https://www.example.com"");
```

MANAS JHA

```
WebElement button1 = example.driver.findElement(By.className(""button-class""));
WebElement button2 = example.driver.findElement(By.xpath(""//input[@value='Submit']""));
```

MANAS JHA

```
example.clickButton(button1);
example.clickButton(button2);
}
}
```

In this example, we have a `clickButton()` method that takes a `WebElement` as an argument and performs the `click()` action. We then use dynamic polymorphism to pass different button elements to the method and perform the click action on each of them.

########## MANAS JHA ##########

Will driver.findelements() throw an exception ?

It will throw empty list.

Element has an ID that changes every time the page is loaded" in it, then which of the following will you use?

If an element's ID changes every time the page is loaded, then using the element's ID to locate it in automated tests won't work. In this case, we will use other locating strategies such as XPath, CSS Selector, or using other attributes of the element to locate it.

Here is an example of using XPath to locate an element with a changing ID:

```
WebElement element = driver.findElement(By.xpath(""//div[contains(@id,'somePartialId')]""));
```

This code will find a `<div>` element that has an ID attribute containing the text 'somePartialId'. The `contains()` function is used to match the partial ID value. This way, we can locate the element even if its complete ID value changes every time.

How do you handle dynamic data in automation testing?

A: One approach is to use data-driven testing, where test data is stored separately from the test case and can be easily updated or modified. Another approach is to use regular expressions to extract dynamic data from the application.

Example code for data-driven testing:


```
// Assume we have an Excel sheet with test data in columns A and B
public class TestData {
public static Object[][] testData() {
Object[][] data = new Object[2][2];
data[0][0] = ""John"";
data[0][1] = ""Password1"";
data[1][0] = ""Jane"";
data[1][1] = ""Password2"";
return data;
}
}

// In our test case:
@Test(dataProvider = ""testData"", dataProviderClass = TestData.class)
public void testLogin(String username, String password) {
SignupPage signupPage = new SignupPage(driver);
// Enter dynamic data
signupPage.enterUsername(username);
signupPage.enterPassword(password);
// Assert login success
// ...
}
```

What is Page Object Model and how can it be implemented in Selenium?

A: Page Object Model is a design pattern used for automation testing where each page in the application has a corresponding Page Object class, which contains methods for interacting with the page's elements. This allows for more reusable and maintainable test code.

Example Page Object class:

```
public class SignupPage {
private WebDriver driver; // assume driver is initialized elsewhere

private By usernameInput = By.id(""username"");
private By passwordInput = By.id(""password"");
private By submitButton = By.id(""submit"");

public SignupPage(WebDriver driver) {
this.driver = driver;
}

public void enterUsername(String username) {
driver.findElement(usernameInput).sendKeys(username);
}

public void enterPassword(String password) {
driver.findElement(passwordInput).sendKeys(password);
}

public void clickSubmit() {
driver.findElement(submitButton).click();
}
}
```

In our test case:

```
@Test
public void testSignup() {
SignupPage signupPage = new SignupPage(driver);
// Enter data using PageObject methods
signupPage.enterUsername(""John"");
signupPage.enterPassword(""Password1"");
signupPage.clickSubmit();
// Assert signup success
// ...
}
```


Does takes screenshot is interface or class

Taking a screenshot is implemented as a method in a class, and it requires an interface or an object of a specific class to interact with the operating system or device's system APIs to capture the screen. In Java, the most

commonly used class for taking screenshots is ""java.awt.Robot."" Here's an example of how to use this class to take a screenshot in Java:

```
java
import java.awt.Rectangle;
import java.awt.Robot;
import java.awt.Toolkit;
import java.awt.image.BufferedImage;
import java.io.File;
import javax.imageio.ImageIO;

public class ScreenshotExample {

public static void main(String[] args) throws Exception {
String fileName = ""screenshot.png""; // specify the file name for the screenshot
Rectangle screenRect = new Rectangle(Toolkit.getDefaultToolkit().getScreenSize()); // get the screen size
BufferedImage screenCapture = new Robot().createScreenCapture(screenRect); // capture the screen
ImageIO.write(screenCapture, ""png"", new File(fileName)); // save the screenshot as a PNG file
System.out.println(""Screenshot saved as "" + fileName);
}
}
```


Note that the above example code creates a screenshot of the entire screen and saves it as a PNG file. You can modify it to capture a specific area of the screen or to save the screenshot in a different image format if required.

Selenium uses lots of third parties jars for scripting. Then why do we still go for selenium?

Selenium is a powerful and widely-used automation tool in the software industry, despite its reliance on third-party jars for scripting. The main reasons why Selenium is still preferred include:

1. Selenium has a huge support community: Due to its wide popularity, there is a vast community of automation experts who use Selenium and contribute to its development. This means that there is an abundance of resources available for learning and troubleshooting.

2. It supports multiple languages: Selenium is designed to support multiple languages, such as Java, Python, and C#. This provides a lot of flexibility and allows automation engineers to work with the programming language they are most comfortable with.

3. It integrates well with other tools: Selenium has a vast network of plugins and integrations that allow it to work seamlessly with other tools in the developer's ecosystem, such as Continuous Integration tools (CI/CD).

Example of Selenium code in Java for opening a browser:

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
public class OpenBrowser {
public static void main(String[] args) {
```

```
//Set path of chromedriver executable
System.setProperty(""webdriver.chrome.driver"", ""path/to/chromedriver.exe"");
//Create instance of ChromeDriver
WebDriver driver = new ChromeDriver();
//Navigate to Google website
driver.get(""https://www.google.com"");
//Close the browser
driver.quit();
}
}
```

In summary, Selenium's widespread adoption, language flexibility, and integration capabilities justify its use despite its reliance on third-party jars.

Window Handling in Selenium -Switching from another window to Parent window

Window handling in Selenium is a crucial task, especially when the application under test opens multiple windows. Switching between windows allows testers to perform necessary tasks on each window. One such task is switching from another window to the parent window using Selenium.

To switch from another window to the parent window in Selenium, we need to perform the following steps:

Step 1: Get the window handle of the current window using the WebDriver object's getWindowHandle() method.

Step 2: Get all window handles of the opened windows using the WebDriver object's getWindowHandles() method.

Step 3: Switch to the parent window handle by iterating over each window handle and comparing it with the current window handle. Once the parent window handle is found, switch to it using the WebDriver object's switchTo() method.

Here is an example code snippet in Java that switches from another window to the parent window using Selenium:

```
// Get the current window handle
String currentWindowHandle = driver.getWindowHandle();

// Get all window handles
Set<String> windowHandles = driver.getWindowHandles();

// Switch to the parent window handle
for (String windowHandle : windowHandles) {
if (!windowHandle.equals(currentWindowHandle)) {
driver.switchTo().window(windowHandle);
}
}

// Perform necessary tasks on the parent window
```

```
// Switch back to the original window
driver.switchTo().window(currentWindowHandle);
```

########## MANAS JHA ##########

If the button is disabled? how to check -using getattribute()

method in Selenium WebDriver, we can check if the button is disabled or not. The getAttribute() method returns the value of the given attribute of an element. In this case, we can use the method to get the value of the ""disabled"" attribute of the button element. If the value is ""true"", then the button is disabled. Here is an example code snippet in Java:

```
WebElement button = driver.findElement(By.id(""buttonID""));
String isDisabled = button.getAttribute(""disabled"");
if (isDisabled.equals(""true"")) {
System.out.println(""Button is disabled"");
} else {
System.out.println(""Button is enabled"");
}
```


########## MANAS JHA ##########

What are the advantages of using TestNG over JUnit?

TestNG provides several additional features over JUnit such as:

1. Flexible test configurations: TestNG allows us to configure tests using annotations or XML files. We can include or exclude specific test methods or groups, prioritize tests, and set up dependencies between tests.

2. Data-driven testing: TestNG allows us to perform data-driven testing using DataProvider annotations. We can pass test data from external sources such as CSV files or databases.

3. Parallel test execution: TestNG allows us to run tests in parallel to improve test performance.

4. Enhanced reporting: TestNG provides more detailed and customizable HTML reports compared to JUnit.

Here is an example of using DataProvider in TestNG:

```
@Test(dataProvider = ""testData"")
public void testMethod(String param1, int param2) {
//Test code
}

@DataProvider(name = ""testData"")
public Object[][] testData() {
return new Object[][] {{""value1"", 1}, {""value2"", 2}};
}
```

########## MANAS JHA ##########

What is the difference between implicit and explicit waits in Selenium?

- Implicit wait is a global wait that is applied to all elements in the WebDriver instance. It sets a timeout value for each element to be located or interacted with before throwing a NoSuchElementException. Implicit wait is set using the WebDriver instance:

driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

On the other hand, explicit wait is a condition-based wait that applies only to specific elements. It waits for a particular condition to be met before proceeding to the next step. Some of the conditions include element presence, clickable, visible, or invisible. Explicit wait is set using the WebDriverWait instance:

WebDriverWait wait = new WebDriverWait(driver, 10);
WebElement element = wait.until(ExpectedConditions.elementToBeClickable(By.id(""buttonID"")));

Here, the explicit wait will wait for the element with ID ""buttonID"" to be clickable for up to 10 seconds before proceeding to the next step.

How do you read excel in the script?

To read Excel files in a script, we can make use of Apache POI library in Java. Here is an example code:


```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class ExcelReader {

public void readExcel() throws IOException {
String filePath = ""C:/TestData.xlsx"";
File file = new File(filePath);
FileInputStream inputStream = new FileInputStream(file);
Workbook workbook = new XSSFWorkbook(inputStream);
Sheet sheet = workbook.getSheetAt(0);
Row row = sheet.getRow(0);
Cell cell = row.getCell(0);
String cellValue = cell.getStringCellValue();
System.out.println(""Value of first cell: "" + cellValue);
workbook.close();
}

}
```

Here, we first create a File object with the path to our Excel file. Then, we create a FileInputStream object to read data from the file. Next, we create a Workbook object using the XSSFWorkbook class for Excel files in XLSX format. We can also use the HSSFWorkbook class for XLS format.

We then get the first sheet of the workbook using the getSheetAt() method, and the first row of the sheet using the getRow() method. Finally, we get the value of the first cell in the row using the getCell() method, and then get the string value using the getStringCellValue() method.

We can loop through the rows and cells to read all the data from the Excel file. And for counter-question, we may need to handle exceptions such as IOException, NullPointerException, etc., which might occur during the execution of the code.

############################ MANAS JHA ############################

What is Page Factory in POM Design pattern?

Page Factory is a design pattern in Selenium WebDriver that helps to initialize the WebElements in a page object model (POM). Used in conjunction with POM, Page Factory provides an easier, faster, and more organized way of writing test scripts.

By using Page Factory, the WebElements of a page are initialized only once, when the page object is created. This reduces the time and resources needed to locate WebElements when executing test cases.

Example:

Assuming you have the following Page Object:

```
public class LoginPage {
private WebDriver driver;

public LoginPage(WebDriver driver) {
this.driver = driver;
PageFactory.initElements(driver, this);
}

@FindBy(id = ""email"")
private WebElement emailField;

@FindBy(id = ""password"")
private WebElement passwordField;

@FindBy(id = ""login-button"")
private WebElement loginButton;

public void login(String email, String password) {
emailField.sendKeys(email);
passwordField.sendKeys(password);
loginButton.click();
}
}
```

To use this Page Object in your test script, you simply initialize the LoginPage object and invoke the login() method. The WebElements are initialized automatically using PageFactory.initElements().

LoginPage loginPage = new LoginPage(driver);
loginPage.login(""user@email.com"", ""password123"");

In the above example, the web element fields (emailField, passwordField, and loginButton) are not initialized using the traditional method of using ""FindElement"". Instead, PageFactory.initElements() is used to initialize the fields automatically.

How to download a file using Selenium?

To download a file using Selenium, we can follow the below steps:

1. Set the preferences to download the file in a specific location:


```
FirefoxProfile profile = new FirefoxProfile();
profile.setPreference(""browser.download.folderList"", 2);
profile.setPreference(""browser.download.manager.showWhenStarting"", false);
profile.setPreference(""browser.download.dir"", ""C:\\Downloads"");
profile.setPreference(""browser.helperApps.neverAsk.saveToDisk"", ""application/pdf"");
```


2. Launch the browser with the above profile:

```
WebDriver driver = new FirefoxDriver(profile);
```


3. Navigate to the URL from where the file needs to be downloaded:

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


4. Find the download button/link and click it:

```
WebElement downloadButton = driver.findElement(By.id(""downloadButton""));
downloadButton.click();
```

5. Wait for the download to complete before closing the browser:

```
Thread.sleep(5000);
driver.quit();
```

In the above example, we are using Firefox browser and setting the download folder location to ""C:\Downloads"". We are also setting the file type to be downloaded as ""application/pdf"". After locating and clicking the download button/link, we are waiting for 5 seconds for the file to download before closing the browser.

How do you automate localization testing -diff language in UI?

Localization testing can be automated by using tools like Selenium and TestNG. Here's an example code snippet for testing multiple languages in the UI:

```
@Test
public void testLocalization() {
WebDriver driver = new ChromeDriver();
String[] languages = {""en"", ""fr"", ""es""};
for (String language : languages) {
// Change language
driver.get(""http://example.com/"" + language);

// Verify proper localization of UI elements
WebElement header = driver.findElement(By.id(""header""));
assertTrue(header.getText().contains(""Welcome""));
// Verify proper localization of form validation messages
WebElement submitButton = driver.findElement(By.id(""submit-button""));
submitButton.click();
WebElement errorMsg = driver.findElement(By.id(""error-message""));
assertTrue(errorMsg.getText().contains(""Please enter a valid email address.""));
}
driver.quit();
}
```

In this example, we use the Selenium WebDriver to launch the Chrome browser and navigate to the website in different languages. We then verify that the UI elements, such as the header, are properly localized by checking if the expected text, such as ""Welcome"", is present. Finally, we also verify that form validation messages, such as the error message for an invalid email address, are properly localized.

Even though CSS is faster than Xpath ,why do 95% of the companies use XPath ?

Although CSS selectors are generally faster and more efficient than XPath, many companies still use XPath because it allows for more complex and precise targeting of elements within the HTML document. XPath also provides a more flexible and expressive syntax for selecting elements, which can be especially useful for test automation scenarios where the target elements may be dynamically generated or have variable attributes.

For example, if we wanted to select all the links on a webpage that have the text ""Login"", using CSS selectors, we could use the following code:

```
driver.findElements(By.cssSelector(""a:contains('Login')""));
```

However, this would only work if the ""Login"" text is always contained within an <a> tag. If the text is sometimes contained within a or , for instance, then we would need to use a more complex CSS selector or resort to XPath.

By contrast, using an XPath expression, we could target all elements on the page that contain the text ""Login"" using the following code:

driver.findElements(By.xpath(""//*[contains(text(),'Login')]""));

This XPath expression will match any element on the page that contains the text ""Login"", regardless of whether it is contained within a link or some other element.

Ultimately, the choice of using CSS or XPath selectors will depend on the specific requirements of the test scenario and the preferences of the test automation team.

In Selenium, how to get text value from text-box if gettext() is not working?

We can use the ""getAttribute()"" method to get the value of a text-box if the ""getText()"" method is not working. This method retrieves the value of a specified attribute of an element.

For example:


```
WebElement textBox = driver.findElement(By.id(""textbox_id""));
String value = textBox.getAttribute(""value"");
System.out.println(""TextBox value is: ""+value);
```


Here, we retrieve the element using the ""findElement()"" method and then use the ""getAttribute()"" method to get its value. The attribute used here is ""value"" which gives us the text value of the text-box.

In Page object model once you create loginpage.java class what is the first thing you start with writing initially. How are you initiating writing something into a page class?

The first thing to do after creating a LoginPage.java class in the Page Object Model is to define the page elements or web elements of the Login page. This is done using Java annotations such as @FindBy or @FindBys for each element on the page.

For Example:


```
// LoginPage.java Class
public class LoginPage {
// Web Elements
@FindBy(id = ""username"")
private WebElement username;

@FindBy(id = ""password"")
private WebElement password;

@FindBy(id = ""loginButton"")
private WebElement loginButton;

// Constructor
public LoginPage(WebDriver driver) {
PageFactory.initElements(driver, this);
}
```

```
// Methods
public void enterUsername(String username) {
this.username.sendKeys(username);
}

public void enterPassword(String password) {
this.password.sendKeys(password);
}

public void clickLoginButton() {
this.loginButton.click();
}
}
```

In the above example, we have used @FindBy annotation to define web elements such as username, password, and loginButton on the login page. We have also included a constructor method and other methods to interact with these web elements.

After defining the page elements, we can start writing the test methods in our Test class to test the functionality of the Login page.

To initiate writing something into a page class, we can use page factory's initElements() method from Selenium to initialize the page elements of the page class.


```
// Test class
public class LoginPageTest {
// BeforeTest method to initialize the driver
@BeforeTest
public void setUp() {
WebDriver driver = new ChromeDriver();
// Maximize the browser window
driver.manage().window().maximize();
// Navigate to the login page
driver.get(""https://www.example.com/login"");
}
```



```
// Test method
@Test
public void testLogin() {
// Create an object of LoginPage
LoginPage loginPage = new LoginPage(driver);
// Enter the username
loginPage.enterUsername(""testuser"");
// Enter the password
loginPage.enterPassword(""testpassword"");
// Click the login button
loginPage.clickLoginButton();
// Assert that the user is logged in successfully
// assertion code goes here
}
```

```
// AfterTest method to close the browser
@AfterTest
public void tearDown() {
driver.quit();
}
}
```

In the above example, we have first initialized the driver and navigated it to the login page. Then, we have created an object of the LoginPage class passing the driver instance as a parameter to the constructor. After that, we have called the methods of LoginPage to interact with the web elements of the login page. Finally, we have written the assertion code to assert whether the user is logged in successfully or not.

Is it possible to change the behavior of a test at runtime?

Yes, it is possible to change the behavior of a test at runtime. For example, in Selenium WebDriver, we can use the WebDriverEventListener interface to listen to different events and change the behavior of the test accordingly. We can create our own custom listener that implements this interface, which will then be called whenever a certain event occurs during test execution.

Here is an example of how we can use the WebDriverEventListener interface in Java:


```
public class CustomListener implements WebDriverEventListener {

@Override
public void onException(Throwable throwable, WebDriver driver) {
// Handle exception and change test behavior accordingly
}

@Override
public void afterClickOn(WebElement element, WebDriver driver) {
// Change test behavior after a click event
}

// Other methods implementing the WebDriverEventListener interface

}
```


We can then register this custom listener with the WebDriver instance, and it will be used during test execution:

```
WebDriver driver = new ChromeDriver();
CustomListener customListener = new CustomListener();
EventFiringWebDriver eventDriver = new EventFiringWebDriver(driver);
eventDriver.register(customListener);

// Use the eventDriver in test code instead of the original driver instance
```

By using this approach, we can dynamically change the behavior of our tests based on certain events that occur during test execution.

How to scroll down a page?

To scroll down a page using Selenium WebDriver in Java, you can use the following code snippet:

```
// Locate the element you want to scroll to
WebElement element = driver.findElement(By.id(""element-id""));

// Scroll to the element using JavaScriptExecutor
((JavascriptExecutor) driver).executeScript(""arguments[0].scrollIntoView(true);"", element);
```


This code finds the element with the specified ""element-id"" and scrolls the page until the element is in view at the top of the page.

If you want to scroll the page by a specific amount, you can use the following code snippet:


```
// Scroll the page down by 500 pixels
((JavascriptExecutor) driver).executeScript(""window.scrollBy(0, 500);"");
```


This code scrolls the page down by 500 pixels. You can adjust the amount of scrolling by changing the value of the second parameter in the ""scrollBy"" method.

What will driver.getWindowHandels() return?

driver.getWindowHandles() will return a set of all the window handles available within the current session of the WebDriver. This method is very useful when working with multiple windows in a web application.

For example, to switch between different windows, we can use the getWindowHandles() method to get all the window handles. Then we can iterate over this set to get the handle of each window and switch to that window using the switchTo() method.

Code example:

```
Set<String> windowHandles = driver.getWindowHandles();

for (String handle : windowHandles) {
driver.switchTo().window(handle);
}
```


How to find the length of the string without using length?

One way to find the length of a string without using the length method is by looping through the string and counting each character. Here is an example code snippet in Java:

```
public static int findStringLength(String str) {
int count = 0;
for(char character : str.toCharArray()) {
count++;
}
return count;
}
```

Another way is by using recursion. Here is an example code snippet in Java:

```
public static int findStringLength(String str) {
if(str.equals("""")) {
return 0;
}
else {
return 1 + findStringLength(str.substring(1));
}
}
```


Both of these methods may not be as efficient as using the length method, but they provide an alternative solution to finding the length of a string.

Write a code to reverse the code, without using the built-in method.

Here is an example code in Java language to reverse a string without using the built-in method:

```
public static String reverseString(String str) {
char[] charArray = str.toCharArray();
int left = 0;
int right = charArray.length - 1;
while (left < right) {
char temp = charArray[left];
charArray[left] = charArray[right];
charArray[right] = temp;
left++;
right--;
}

return new String(charArray);
}
```

You can call this function and pass the string that you want to reverse as a parameter, for example:

```
String original = ""Hello World"";
String reversed = reverseString(original);
System.out.println("riginal: "" + original);
```

```
System.out.println(""Reversed: "" + reversed);

Output:
Original: Hello World
Reversed: dlroW olleH
```


Write a code to get Highest number using array.

Sure, here's a code snippet to get the highest number using an array in Java:

```
java
public class HighestNumberInArray {
public static void main(String args[]) {
int[] numbers = {10, 20, 30, 40, 50, 60, 70, 80, 90};
int highestNumber = numbers[0];
for(int i = 1; i < numbers.length; i++) {
if(numbers[i] > highestNumber) {
highestNumber = numbers[i];
}
}
System.out.println(""The highest number in the array is: "" + highestNumber);
}
}
```

In this code, we first define an array of numbers. We then set `highestNumber` to the first element in the array. We then loop through the rest of the array, checking if each element is greater than our current `highestNumber`. If it is, we update `highestNumber` to that value. Finally, we print out the highest number that we found.

"To remove duplicates from a string in Java, we can use the LinkedHashSet as it maintains the order of elements and also removes duplicates.

Example Code:

```
String str = ""Hello World!"";
LinkedHashSet<Character> set = new LinkedHashSet<>();
for(char c : str.toCharArray()){
   set.add(c);
}
String result = """";
for(char c : set){
   result += c;
}
System.out.println(result);

Output: Helo Wrd!"
```

"One way to find the length of a string without using the length method is to iterate over the characters of the string using a loop and keep track of the number of characters until a null character is encountered.

Here's an example code snippet in Java:

```
public static int getStringLength(String str) {
  int count = 0;
  for(char c : str.toCharArray()) {
    count++;
    if(c == '\0') {
      break;
    }
  }
  return count;
}
```

In this code, we first initialize a counter variable `count` to 0. Then, we iterate over the characters of the string `str` using a for-each loop. In each iteration, we increment the counter by 1 and check if the current character `c` is a null character (`'\0'`). If we encounter a null character, we break out of the loop. Finally, we return the count as the length of the string."

To find the largest number in an array, you can iterate through the entire array and compare each element with the current maximum value. Here is an example Java code:

```
int[] arr = {2, 5, 8, 3, 1};

int max = arr[0]; // assume first element is the max

for(int i=1; i<arr.length; i++) {
  if(arr[i] > max) {
    max = arr[i];
  }
}

System.out.println(""Largest number in array is: "" + max);
```

This code starts by assuming the first element as the maximum value and then compares each element with the current maximum using a for loop. If the current element is greater than the current maximum, it updates the maximum value. Finally, it prints out the largest number in the array."

To reverse a string without using the reverse() function, we can loop through the string and append each character to a new string in reverse order.

Here is an example implementation in Java:

```java
public String reverseString(String str) {
   String reversedStr = """";
   for (int i = str.length() - 1; i >= 0; i--) {
      reversedStr += str.charAt(i);
   }
   return reversedStr;
}
```

This function takes a string as input and returns the reversed string. It uses a for loop to iterate through the characters in the string starting from the last character (i.e. str.length() - 1) and appending each character to a new string in reverse order.

For example, if we call the function with the input string ""hello"", it will return the string ""olleh""."
"Sure, here is the Java code to print the Fibonacci series:

```
public static void printFibonacciSeries(int n) {
   int first = 0, second = 1;
   System.out.print(first + "" "" + second + "" "");
   for (int i = 2; i < n; i++) {
      int sum = first + second;
      System.out.print(sum + "" "");
      first = second;
      second = sum;
   }
}

// Example usage
int n = 10;
printFibonacciSeries(n);
```

This code will print the first `n` numbers in the Fibonacci series, where `n` is an integer passed as input to the `printFibonacciSeries()` method. For example, calling `printFibonacciSeries(10)` will print the first 10 numbers in the series: `0 1 1 2 3 5 8 13 21 34`."
"Sure, here's the Java code to print only the even numbers from an array:

```
int[] arr = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
for (int i = 0; i < arr.length; i++) {
   if (arr[i] % 2 == 0) {
      System.out.println(arr[i]);
   }
}
```

This code will loop through the array and check each element to see if it's divisible by 2 (i.e., is an even number). If it is, it will print that number to the console. If you run this code, it will output:

```
2
4
6
8
10
```

So, it only prints the even numbers from the array."
"Here is the Java code to find special character, number, capital and small letter in a given string:

```
public class FindSpecialCharNumCapsSmallLetters {
```

```
  public static void main(String[] args) {
    String inputString = ""This String Contains 1 Special @ Character"";
    findSpecialCharNumCapsSmallLetters(inputString);
  }

  public static void findSpecialCharNumCapsSmallLetters(String inputString) {
    int charCount = 0;
    int numCount = 0;
    int capCount = 0;
    int smallCount = 0;
    int specialCharCount = 0;

    for(int i=0; i<inputString.length(); i++) {
      char ch = inputString.charAt(i);

      if(Character.isLetter(ch)) {
        charCount++;
        if(Character.isUpperCase(ch)) {
          capCount++;
        } else {
          smallCount++;
        }
      } else if(Character.isDigit(ch)) {
        numCount++;
      } else {
        specialCharCount++;
      }
    }

    System.out.println(""Number of Characters: "" + charCount);
    System.out.println(""Number of Numbers: "" + numCount);
    System.out.println(""Number of Capital Letters: "" + capCount);
    System.out.println(""Number of Small Letters: "" + smallCount);
    System.out.println(""Number of Special Characters: "" + specialCharCount);
  }

}
```

Output:

```
Number of Characters: 36
Number of Numbers: 1
Number of Capital Letters: 3
Number of Small Letters: 32
Number of Special Characters: 3
```"

Below is the code to check if a number is palindrome in Java -

```
public class PalindromeNumberExample {
 public static void main(String args[]){
  int num = 121, reversed=0, remainder, original = num;

  while(num != 0){
    remainder = num % 10;
    reversed = reversed * 10 + remainder;
    num /= 10;
  }

  if(original == reversed){
    System.out.println(original + "" is a palindrome."");
  }else{
    System.out.println(original + "" is not a palindrome."");
  }
 }
}
```


In the above code, we first declare a variable num and set its value to 121, which needs to be checked if it's a palindrome. We declare 3 more variables - reversed, remainder and original.

We calculate the reverse of the number using the while loop and check if the original number and the reversed number are equal. If they are equal, then the number is a palindrome. Otherwise, it's not a palindrome."
"Sure, here's an example of how to **write a code to reverse a string in Java without using any built-in methods:**

```
public static String reverseString(String str) {
    char[] strArr = str.toCharArray();
    int len = strArr.length;

    for (int i = 0; i < len / 2; i++) {
        char temp = strArr[i];
        strArr[i] = strArr[len - i - 1];
        strArr[len - i - 1] = temp;
    }

    return new String(strArr);
}
```

In this code, we convert the input string into a character array and then use a for loop to swap the characters at opposite ends of the array until we have traversed half of it. Finally, we convert the reversed character array back into a string and return it."

Why we use TestNG in your framework

TestNG is a popular testing framework in Java used for automation testing. It is highly used to create and organize test cases effectively. TestNG can be integrated easily with various tools like Selenium, Maven, Ant,

Jenkins, etc., making it easier to execute test cases as part of the Continuous Integration process using CI/CD tools. The following are some of the reasons why we use TestNG in our testing framework:

1. Annotation Support - TestNG provides a variety of annotations that help to better control the test case execution. Annotations allow test cases to be marked with Pre-conditions, Post-conditions, Test data setup, Test data cleanup, etc.

2. Test Case Grouping - TestNG provides the ability to group test cases, allowing us to execute tests with similar or related functionalities.

3. Parallel Test Execution - TestNG offers parallel execution of test cases, enabling faster and more efficient testing.

4. Reporting - TestNG provides detailed test reports for each test case, which allows us to identify failed test scenarios easily.

Example Code:


```
import org.testng.annotations.*;

public class TestClass {
@Test
public void testMethod() {
// Enter test code here
}
@BeforeMethod
public void setupMethod() {
// Enter setup code here
}
@AfterMethod
public void cleanupMethod() {
// Enter cleanup code here
}
}
```


In the above code example, we define a TestNG test class using annotations `@Test`, `@BeforeMethod` and `@AfterMethod`. The `@Test` annotation indicates that the code following the annotation will be executed as a test method. The `@BeforeMethod` annotation indicates that the code following the annotation will be executed before each test method, and the `@AfterMethod` annotation indicates that the code following the annotation will be executed after each test method. We can add multiple test methods inside this class with different annotations to control test case execution.

Out of 50 testcases, how you will run only the failed testcases?

To run only the failed test cases, we can use testNG framework in Java. We can use the ""rerun"" feature of testNG to re-run only the failed test cases.

Steps to follow:

1. First, we need to execute all the test cases.

2. After executing all the test cases, we can get the list of failed test cases using testNG 'ITestResult' interface.
3. Then, we can set the failed test cases in the testNG XML file using the ""FailedTestRetryAnalyzer"" class to re-run only the failed test cases.

Sample code:

```
public class FailedTestRetryAnalyzer implements IRetryAnalyzer {
private int retryCount = 0;
private static final int MAX_RETRY_COUNT = 2; //this will re-run failed cases 2 times.

@Override
public boolean retry(ITestResult result) {
if (retryCount < MAX_RETRY_COUNT) {
retryCount++;
return true;
}
return false;
}
}
```


In testNG XML file:


```
<!DOCTYPE suite SYSTEM ""http://testng.org/testng-1.0.dtd"">
<suite name=""Test"" verbose=""1"" >
<test name=""Test1"" >
<classes>
<class name=""com.automation.tests.TestCase1"" >
<methods>
<include name=""test1"" />
<include name=""test2"" />
</methods>
</class>
<class name=""com.automation.tests.TestCase2"" >
<methods>
<include name=""test1"" />
<include name=""test2"" />
</methods>
</class>
</classes>
<listeners>
<listener
class-name=""com.automation.tests.FailedTestRetryAnalyzer"" />
</listeners>
</test>
</suite>
```


Types of Listeners?

In TestNG, there are several types of listeners that can be used to perform specific actions during different stages of the test execution process.

1. ITestListener - This listener is used to perform actions before and after a test method is executed.

For example, you can use the following code to implement ITestListener in Java:

```
public class MyTestListener implements ITestListener {
@Override
public void onTestStart(ITestResult result) {
System.out.println(""Test started: "" + result.getName());
}

@Override
public void onTestSuccess(ITestResult result) {
System.out.println(""Test passed: "" + result.getName());
}

@Override
public void onTestFailure(ITestResult result) {
System.out.println(""Test failed: "" + result.getName());
}

//Other methods can also be implemented to perform actions on test skip or test finish
}
```


2. ISuiteListener - This listener is used to perform actions before and after a test suite is executed.

For example, you can use the following code to implement ISuiteListener in Java:

```
public class MySuiteListener implements ISuiteListener {
@Override
public void onStart(ISuite suite) {
System.out.println(""Suite started: "" + suite.getName());
}

@Override
public void onFinish(ISuite suite) {
System.out.println(""Suite finished: "" + suite.getName());
}
}
```

3. IAnnotationTransformer - This listener is used to modify the annotations of a test or suite at runtime.

For example, you can use the following code to implement IAnnotationTransformer in Java:

```
public class MyAnnotationTransformer implements IAnnotationTransformer {
@Override
public void transform(ITestAnnotation annotation, Class testClass, Constructor testConstructor, Method testMethod) {
//Modify the annotation
}
}
```

4. IMethodInterceptor - This listener is used to modify the test methods that are executed within a test class.

For example, you can use the following code to implement IMethodInterceptor in Java:

```
public class MyMethodInterceptor implements IMethodInterceptor {
@Override
public List<IMethodInstance> intercept(List<IMethodInstance> methods, ITestContext context) {
List<IMethodInstance> result = new ArrayList<>();
//Filter the methods based on certain criteria and add them to the result list
return result;
}
}
```


How many suits can be there in testNG , what if I run all the suits?

In TestNG, we can create multiple test suites based on the requirements. The number of test suites that can be created is not limited.

If we run all the test suites, then all the tests that are included in those suites will be executed accordingly. The test cases will run one after another in the order specified in the suite file.

For example, if we have two test suites named ""LoginSuite"" and ""RegistrationSuite"", and if we run all the suites, TestNG will execute all the tests that are included in these two suites. This process will continue until all the tests have been executed.

Sample code for including test suites in TestNG:

```
java
import org.testng.TestNG;
import org.testng.xml.XmlSuite;
import java.util.ArrayList;
import java.util.List;
public class TestRunner {
public static void main(String[] args) {
//Create an instance of TestNG
TestNG testng = new TestNG();

//Create an instance of XmlSuite and set the suite name
XmlSuite suite1 = new XmlSuite();
suite1.setName(""Login Suite"");

//Create a list of classes to be included in this suite
List<String> classNames = new ArrayList<String>();
classNames.add(""com.example.tests.LoginTest"");
classNames.add(""com.example.tests.ForgotPasswordTest"");

//Set the test classes for this suite
suite1.setIncludedTests(classNames);
```

```
//Create another instance of XmlSuite and set the suite name
XmlSuite suite2 = new XmlSuite();
suite2.setName(""Registration Suite"");

//Create a list of classes to be included in this suite
classNames = new ArrayList<String>();
classNames.add(""com.example.tests.RegisterTest"");

//Set the test classes for this suite
suite2.setIncludedTests(classNames);

//Create a list of suites
List<XmlSuite> suites = new ArrayList<XmlSuite>();
suites.add(suite1);
suites.add(suite2);
```



```
//Set the list of suites for TestNG and run it
testng.setXmlSuites(suites);
testng.run();
}
}
```


This code creates two test suites (Login Suite and Registration Suite) and includes the respective test classes in those suites. Finally, it runs both the suites using TestNG.

Write skeleton of test runner?

```
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;
import org.junit.runner.RunWith;
```



```
@RunWith(Cucumber.class)
@CucumberOptions(
    features = "src/test/resources/features", // Directory path of feature files
    glue = {"stepDefinitions"}, // Directory path of step definitions
    plugin = {"pretty", "html:target/cucumber-reports"}) // Optional: Reports plugin for generating test reports

public class TestRunner {
}
```

In this example, we are using Cucumber as the BDD framework and JUnit as the test runner. The @RunWith(Cucumber.class) annotation specifies that JUnit should run the tests using Cucumber. The @CucumberOptions annotation is used to configure the options for the test runner.

The features parameter specifies the directory path of the feature files. The glue parameter specifies the directory path of the step definitions. The plugin parameter is optional and can be used to specify the plugins for generating test reports.

You can add more parameters based on your requirements. Once you have defined the Test Runner class, you can execute it using JUnit.

Cucumber tags? And how to run different combinations of tags when multiple tags are present

Cucumber tags are used to attach metadata to each scenario in the feature file, allowing for easy organization as well as specified execution of specific scenarios.

To run different combinations of tags, you can use the logical operators ""and"" and "r"". For example, if you have tags like ""@login"" and ""@admin"", you can run only the scenarios with both of these tags using the following command:

`mvn test -Dcucumber.filter.tags=""@login and @admin""`

Alternatively, you can use the "r"" operator to run scenarios that have either tag using the following command:

`mvn test -Dcucumber.filter.tags=""@login or @admin""`

And here's an example of how to write tags in a feature file:


```
@login
@regression
Scenario: User logs in successfully
Given the user is on the login page
When they enter valid credentials
Then they should be taken to the dashboard

@signup
@smoke
Scenario: User can sign up
Given the user is on the sign up page
When they enter valid details
Then they should receive a confirmation message
```

In this example, there are four tags: ""@login"", ""@regression"", ""@signup"", ""@smoke"". The logical operators can be used to run specific scenarios based on the tag combination you need.

What is Jenkins?

Jenkins is a popular open-source continuous integration and continuous delivery (CI/CD) tool, which is used to automate the building, testing, and deployment of software. It provides a number of useful features, such as scheduling builds, sending notifications, managing plug-ins, managing source code, and much more. Jenkins is widely used in agile development environments, where fast and frequent delivery of software is critical.

Example:

Suppose, you are working on a web application that requires frequent code changes and a fast turnaround time. To automate the testing and deployment process, you can use Jenkins. You can set up Jenkins to

automatically build, test, and deploy your code every time you make a change, using a configuration file that specifies the build and test steps. Once the build is complete, Jenkins can automatically deploy the new code to a staging environment for further testing, and then to the production environment after it has been thoroughly tested. This ensures that every code change is tested and deployed quickly and reliably, which enables you to deliver new features faster and more efficiently.

Java code:

Below is the sample Jenkins pipeline script in Java language that performs Maven-based builds:

```
pipeline {
agent any

stages {
stage('Build') {
steps {
sh 'mvn clean package'
}
}
stage('Test') {
steps {
sh 'mvn test'
}
}
stage('Deploy') {
steps {
sh 'mvn deploy'
}
}
}
}
```


Note: This is just an example, and the actual script may vary depending on the application requirements.

Today we have executed some tests using maven, but tomorrow when you see that someone deleted all dependencies from .pom file then in that case will you be able to execute tests or not.

No, I won't be able to execute tests if someone has deleted all dependencies from .pom file as dependencies are necessary for executing the tests. I will need to add the dependencies back to the .pom file for successful test execution.

Example:

Suppose, we have a Selenium automated test script that requires dependencies like Selenium WebDriver, TestNG, and Apache POI for data-driven testing. If someone accidentally removes these dependencies from the .pom file, then the test script won't be executed as it will not be able to recognize the imported classes. In this case, I will need to add the dependencies back to the .pom file to ensure successful test execution.

Code example to add dependency of Selenium WebDriver in a Maven project:

```
<dependency>
<groupId>org.seleniumhq.selenium</groupId>
<artifactId>selenium-java</artifactId>
<version>3.141.59</version>
</dependency>
```


How will you configure Jenkins job?

What is difference between group id and artefact id?

In Maven, group id and artifact id are used to uniquely identify a project or module in the repository.

The group id is used to identify the project's group or organization. It provides a unique identifier for the project's namespace. For example, com.mycompany or org.apache.

The artifact id, on the other hand, is used to uniquely identify the project or module within the group. It gives the name of the module such as core or web.

For example, in the following Maven coordinates:

```
<groupId>com.mycompany</groupId>
<artifactId>myproject-core</artifactId>
<version>1.0.0</version>
```


The group id is ""com.mycompany"" and the artifact id is ""myproject-core"". In this way, we can uniquely identify this project in the repository.

Here is an example of how these two identifiers are used in a Java project using Maven.


```
<project xmlns=""http://maven.apache.org/POM/4.0.0""
xmlns:xsi=""http://www.w3.org/2001/XMLSchema-instance""
xsi:schemaLocation=""http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"">
<modelVersion>4.0.0</modelVersion>

<groupId>com.mycompany</groupId>
<artifactId>myproject-core</artifactId>
<version>1.0.0</version>

<dependencies>
<!-- Dependencies for the core module -->
</dependencies>
</project>
```

In this example, the group id is ""com.mycompany"", the artifact id is ""myproject-core"", and the version is ""1.0.0"". These Maven coordinates uniquely identify this project in the repository. The dependencies for this module are also listed in the dependencies section.

########## MANAS JHA ##########

Difference between REST and SOAPUI.

REST (Representational State Transfer) is a web services architecture that provides a lightweight and flexible approach for communication between client and server applications over the network. REST uses HTTP protocol for communication and supports XML, JSON, and other data formats.

On the other hand, SOAPUI (Simple Object Access Protocol User Interface) is an open-source tool used for testing web services. It is a protocol-based testing tool that supports SOAP (Simple Object Access Protocol), REST, and other web services protocols.

The main difference between REST and SOAPUI is their architectural approach. REST follows a resource-based architectural style, whereas SOAPUI follows a message-based architectural style. REST uses HTTP status codes to handle errors and exceptions, while SOAPUI has its own set of error codes.

Here's a sample code in Java for making a REST API call:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

public class RestAPICall {
public static void main(String[] args) throws Exception {

URL url = new URL(""https://api.example.com/users"");
HttpURLConnection connection = (HttpURLConnection) url.openConnection();
connection.setRequestMethod(""GET"");

BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
String line;
StringBuffer response = new StringBuffer();

while ((line = in .readLine()) != null) {
response.append(line);
} in .close();

System.out.println(response.toString());
}
}
```

This code shows how to make a GET call to a REST API endpoint and fetch data.

########## MANAS JHA ##########

Method in REST.

A Method in REST refers to the HTTP verb used to interact with a resource. REST stands for Representational State Transfer, and it is an architectural style that allows communication over HTTP to exchange resources between clients and servers. There are various HTTP methods, such as GET, POST, PUT, DELETE, OPTIONS, HEAD, and PATCH.

Some examples of how HTTP methods can be used in REST API:

- GET Method: It is used to retrieve information or data from a resource. For example, if we want to retrieve a list of all users from a web application, we can use the HTTP GET method. Here's an example of how to implement GET method in Java:

```
import java.net.HttpURLConnection;
import java.net.URL;
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class GetExample {
public static void main(String[] args) {
try {
URL url = new URL(""https://example.com/users"");
HttpURLConnection con = (HttpURLConnection) url.openConnection();
con.setRequestMethod(""GET"");
BufferedReader in = new BufferedReader(
new InputStreamReader(con.getInputStream()));
String inputLine;
StringBuffer content = new StringBuffer();
while ((inputLine = in.readLine()) != null) {
content.append(inputLine);
}
in.close();
System.out.println(content.toString());
} catch (Exception e) {
System.out.println(""Exception: "" + e);
}
}
}
```

- POST Method: It is used to create a new resource on the server. For example, if we want to create a new user in a web application, we can use the HTTP POST method. Here's an example of how to implement POST method in Java:

```
import java.net.HttpURLConnection;
import java.net.URL;
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.InputStreamReader;

public class PostExample {
```

```
public static void main(String[] args) {
try {
URL url = new URL(""https://example.com/users"");
HttpURLConnection con = (HttpURLConnection) url.openConnection();
con.setRequestMethod(""POST"");
con.setRequestProperty(""Content-Type"", ""application/json"");
con.setDoOutput(true);
String jsonInputString = ""{\""name\"": \""John Doe\"", \""email\"": \""johndoe@example.com\""}"";
try (DataOutputStream wr = new DataOutputStream(con.getOutputStream())) {
byte[] input = jsonInputString.getBytes(""utf-8"");
wr.write(input, 0, input.length);
}
BufferedReader in = new BufferedReader(
new InputStreamReader(con.getInputStream()));
String inputLine;
StringBuffer response = new StringBuffer();
while ((inputLine = in.readLine()) != null) {
response.append(inputLine);
}
in.close();
System.out.println(response.toString());
} catch (Exception e) {
System.out.println(""Exception: "" + e);
}
}
}
```


These are just a few examples of how HTTP methods can be used in REST API. Depending on the requirements of the application, different methods can be used to interact with resources.

Can you tell us about your experience with Selenium and API testing? What types of projects have you worked on and what was your role in those projects?

In my 11 years of experience in the software testing field, I have worked extensively with both Selenium and API testing. I have experience in automating test cases using Selenium WebDriver, working with different browsers and their drivers, and integrating Selenium with testing frameworks such as JUnit or TestNG.

In terms of API testing, I have used tools like Postman and SoapUI to test the functionality of APIs. I have also worked with RESTful APIs and have experience in testing for security vulnerabilities and performance issues.

In my previous projects, I have played the role of a test automation engineer and have been responsible for designing, developing, and executing test cases. I have also been involved in the creation of test plans, test strategies, and test data management.

Can you walk us through your testing methodology? How do you ensure

that the software you test is of high quality and meets customer requirements?

My testing methodology involves several steps, including requirement analysis, test planning, test case design, test execution, and test closure.
During the requirement analysis phase, I make sure to understand the customer requirements thoroughly and identify any potential risks or areas that need to be tested.
In the test planning phase, I create a detailed test plan that outlines the objectives, scope, and resources required for the testing effort.
During test case design, I design test cases that cover the requirements and identify any potential scenarios that could occur during testing.
In the test execution phase, I execute the test cases and document any issues that are found. I also work with the development team to resolve any issues that are identified.
Finally, in the test closure phase, I document the results of the testing effort and ensure that all necessary documentation is completed.
To ensure that the software I test is of high quality, I follow industry standard testing processes and techniques and make use of various testing tools and methodologies. I also make sure to perform thorough regression testing after any changes are made to the software to ensure that it continues to meet customer requirements.

Can you give an example of a complex bug that you have encountered and how you went about resolving it?

One example of a complex bug I encountered was while testing an e-commerce platform. The platform was supposed to display the correct price of an item after discounts were applied, but the price was incorrect in certain cases.

I went about resolving this issue by first reproducing the bug and then examining the code to identify the root cause of the problem. I found that the issue was caused by a mistake in the logic that was used to calculate the discounted price.

I worked with the development team to resolve the issue and, after the fix was made, I performed thorough regression testing to ensure that the fix had not introduced any new issues.

In the end, the issue was resolved and the platform was able to correctly display the discounted price of items.

How do you stay up-to-date with the latest testing techniques and technologies?

I stay up-to-date with the latest testing techniques and technologies by continuously learning and exploring new tools and methodologies. I regularly attend industry conferences and events, read industry publications and blogs, and participate in online communities and forums.

I also like to stay abreast of the latest advancements in software testing and make sure to continuously improve my skills and knowledge in the field.

Can you tell us about a time when you had to work with a cross-functional team to resolve a challenging issue? How did you communicate and coordinate with the different team members to reach a resolution?

Can you describe a testing project you have led and how you managed to deliver it on time and within budget?

Can you discuss your experience with creating and maintaining automated test suites? How do you decide which tests to automate and which to perform manually?

In my 11 years of experience, I have created and maintained many automated test suites. To decide which tests to automate and which to perform manually, I follow a risk-based approach. I evaluate the test cases based on factors such as the frequency of execution, the complexity of the test case, the criticality of the feature being tested, and the likelihood of human error. If a test case is executed frequently, is highly critical, or is prone to human error, it is a good candidate for automation. On the other hand, if a test case is simple, rarely executed, or is not critical to the system, it may not be worth the effort to automate.

Can you tell us about a time when you had to make a trade-off between time and quality? How did you handle that situation and what was the outcome?

There have been several instances where I had to make a trade-off between time and quality. One example that comes to mind is when we were testing a critical system that had to be released by a specific deadline. We had a large number of test cases to execute, but we had limited time to complete the testing. We prioritized the test cases based on their criticality and executed the high-priority tests first. We also leveraged automation to speed up the testing process. However, as a result of the time constraint, we had to compromise on the depth of testing, and we could not execute all the test cases we had planned. We communicated the risks and the limitations of testing to the stakeholders and released the system with the identified risks. We continued to monitor the system in production and addressed the identified issues in subsequent releases.

Can you discuss your experience with performance testing and how you have used tools such as JMeter or Gatling to identify performance bottlenecks?

I have extensive experience in performance testing and have used tools such as JMeter and Gatling to identify performance bottlenecks. I use JMeter or Gatling to simulate realistic user traffic on the system under test and capture performance metrics such as response time, throughput, and error rates. I analyze the metrics to identify performance bottlenecks and potential scalability issues. I also use profiling tools such as Java Flight Recorder or VisualVM to analyze the performance of the application at the code level. Based on the identified bottlenecks, I collaborate with the development team to optimize the system's performance. This may involve code changes, database optimizations, or infrastructure changes. Once the identified issues are addressed, I rerun the performance tests to confirm the improvements.

Can you discuss your experience with API testing and how you have used tools such as Postman or SoapUI to test API functionality?

Sure, I have extensive experience with API testing, and I have used tools such as Postman and SoapUI to test API functionality. Here are some details about my experience:

API testing approach: I follow a structured approach to API testing that involves creating test cases based on the API's specifications and requirements. I use a combination of manual and automated testing techniques to ensure that the API functions as expected.
API testing types: I have experience testing different types of APIs, including RESTful APIs, SOAP APIs, and microservices. For RESTful APIs, I use HTTP methods such as GET, POST, PUT, DELETE, etc., to test various endpoints and payloads. For SOAP APIs, I use XML-based request and response messages to test the API functionality.
API testing tools: I have used various API testing tools, including Postman and SoapUI. These tools provide a user-friendly interface to create and execute API test cases, and they can also generate detailed test reports.
API testing scenarios: I have tested various scenarios in APIs, including positive and negative scenarios. I test for input validation, boundary conditions, error handling, authentication and authorization, and other functional aspects of the API.
API automation: I have automated API test cases using tools such as Postman and SoapUI. I create automated scripts to test the API's functionality, and I also use assertions to validate the API's response. I have also integrated API testing into the CI/CD pipeline using tools such as Jenkins and GitLab.
In summary, my experience with API testing includes testing various types of APIs using a structured approach, using tools such as Postman and SoapUI, testing various scenarios, and automating API test cases using different tools.

How do you handle conflicts with stakeholders or team members and what is your approach to resolving them?

Handling conflicts with stakeholders or team members is an important aspect of being an effective testing professional. Here's my approach to resolving conflicts:
Identify the root cause: I always try to identify the underlying cause of the conflict. This involves listening to all parties involved and trying to understand their perspective. I ask questions to gather information and clarify any misunderstandings. Once I have a clear understanding of the issue, I move on to the next step.
Collaborate to find a solution: I believe in collaboration to find a solution that is acceptable to all parties. I involve all stakeholders in the conflict resolution process and work together to find a mutually beneficial solution. I encourage open communication and active listening to ensure that everyone's ideas are heard and considered.
Document and communicate the solution: Once a solution is agreed upon, I document it and communicate it to all stakeholders. This ensures that everyone is on the same page and understands the resolution. It also helps prevent similar conflicts from arising in the future.
Follow up: Finally, I follow up on the resolution to ensure that it is effective and that all parties are satisfied. If there are any further issues, I encourage open communication to address them as quickly as possible.
Overall, my approach to conflict resolution is based on active listening, collaboration, documentation, and follow-up. I believe that this approach helps ensure that conflicts are resolved in a professional and effective manner.

What challenges you faced in API testing?

As an automation expert, I have faced several challenges in API testing, some of which are:

1. Understanding API documentation: Sometimes, the API documentation is not clear or outdated, which makes it challenging to understand the appropriate request and response formats.

2. Dealing with authentication and authorization: APIs often require authentication and authorization to access, so testing such APIs can be challenging. It may require setting up tokens, keys, or certificates to authenticate and authorize requests.

3. Handling data validation and parsing: APIs return data in various formats, such as XML, JSON, or HTML, making it crucial for automated tests to validate and parse them correctly.

4. Managing different API versions: When an API has released several versions, maintaining and testing different versions becomes a challenge due to the number of variables involved.

One example of API testing challenges is handling SSL/TLS (Secure Sockets Layer/Transport Layer Security) errors. When an API endpoint requires SSL/TLS, but the certificate is invalid, self-signed, or expired, an SSL/TLS error can occur. It can create difficulties in testing, and the solution is to bypass or ignore these errors during testing.

Here's an example showing how to ignore SSL/TLS errors using Java and RestAssured library:


```
// Ignore SSL/TLS errors
RestAssured.config().sslConfig(new SSLConfig().relaxedHTTPSValidation());

// Make API request
Response response = given()
.when()
.get(""https://example.com/api"");

// Validate response
response.then().statusCode(200);
```


What is difference between Authorization and Authentication?

Authentication is the process of identifying a user by providing credentials such as username and password. It checks if the user is who they claim to be. It is like a gatekeeper who allows only authorized users to enter the system. Authorization, on the other hand, is the process of granting or denying access to specific resources or actions based on the user's identity or role. It defines what a user is allowed to do once they are authenticated. It is like a bouncer who permits only authorized users to perform specific actions.

Here's an example in Java for authentication:

```
// Authentication
String username = ""john"";
String password = ""123456"";
if (username.equals(""john"") && password.equals(""123456"")) {
System.out.println(""Authentication successful"");
} else {
System.out.println(""Authentication failed"");
}
```

And an example for authorization:

```
// Authorization
public class User {
private String username;
private String role;

// getters and setters

public boolean canAccessResource(String resource) {
if (role.equals(""admin"")) {
// Checks if the user is an admin
return true;
} else if (role.equals(""user"")) {
// Checks if the user is a regular user
if (resource.equals(""read"")) {
// Only allows reading access
return true;
}
}
return false;
}
}

// Usage
User user = new User();
user.setUsername(""john"");
user.setRole(""user"");

if (user.canAccessResource(""read"")) {
// Grants access to read the resource
System.out.println(""Access granted"");
} else {
System.out.println(""Access denied"");
}
```


Difference between severity and priority?

Severity refers to the degree of seriousness of a defect or issue found during testing. It is the impact of an issue on the system under test. Priority, on the other hand, refers to the order of importance assigned to resolve the issue. It is the importance of fixing an issue based on business needs.

For example, if a login page is not functioning properly, it could be a severe issue as it prevents users from accessing the system. However, if a minor spelling mistake is found on a less important page, it may have low severity. But, if the typo is on the company's logo or tagline, it could have high priority as it affects the company image.

Here is an example code snippet that shows how to assign priority and severity to a defect using TestNG framework in Java:

```
@Test(priority = 1, severity = Severity.CRITICAL)
```

```
public void loginTest() {
//code for login test
}
```

In this code, priority has been assigned as 1 (highest) and severity as CRITICAL.

Difference between boundary value analysis and equivalence partitioning?

Boundary value analysis and equivalence partitioning are two test case design techniques used in software testing.

Boundary value analysis involves testing the boundary values of inputs, such as the maximum and minimum values, to ensure that the software works correctly at the limits of its input range. For example, if a software application accepts values between 1 and 100, then the boundary values could be 1 and 100. In this case, the test cases would be designed to check the behavior of the software at these values.

Equivalence partitioning involves dividing the input domain into partitions or groups of input values that behave in a similar way. The aim is to test a representative sample from each partition, as testing all values in each partition is not practical. For example, if a software application accepts values between 1 and 100, then the input values could be divided into three partitions: values less than 1, values between 1 and 100, and values greater than 100. In this case, test cases would be designed to test representative values from each partition.

Both boundary value analysis and equivalence partitioning can be used to design effective test cases that can identify defects in software. However, boundary value analysis is useful when testing the edge conditions where software often behaves in unpredictable ways. In contrast, equivalence partitioning is useful when testing large ranges of input values, as it reduces the number of test cases required while still ensuring that a representative sample is tested.

Example of Boundary value analysis:

Suppose you are testing a login page that allows a user to enter a username and password. The login page specifies that the username should be between 6 and 20 characters long. In this case, you can design test cases as follows:

- Test case 1: Enter a valid username of exactly 6 characters
- Test case 2: Enter a valid username of 20 characters
- Test case 3: Enter an invalid username of less than 6 characters
- Test case 4: Enter an invalid username of more than 20 characters

Example of Equivalence partitioning:

Suppose you are testing a search function that allows the user to search for products by price. The search function specifies that the price should be between $10 and $100. For this scenario, you can divide the input values into three partitions:

- Partition 1: Values less than $10 (invalid input)
- Partition 2: Values between $10 and $100 (valid input)
- Partition 3: Values greater than $100 (invalid input)

In this case, you can design test cases as follows:

- Test case 1: Enter price as $5 (invalid input)
- Test case 2: Enter price as $50 (valid input)
- Test case 3: Enter price as $150 (invalid input)

Difference between white box and black box testing?

White box testing (also known as clear box testing or structural testing) is a software testing technique that examines the internal structure and implementation of the software being tested. The aim of white box testing is to ensure that all lines of code are executed and all possible paths are tested. It is typically performed by developers.

Black box testing (also known as functional testing) is a software testing technique that evaluates software functionality without knowing the internal workings of the software. The aim of black box testing is to validate the software from an end-user perspective. It is typically performed by QA testers.

Example of white box testing: Unit testing is a type of white box testing that tests individual units of code, such as classes or methods. For example, a developer may write a unit test to ensure that a particular method of a class is functioning correctly.

Example of black box testing: User acceptance testing (UAT) is a type of black box testing where end-users test the software to verify if it meets the requirements and is suitable for use. For example, a QA tester could conduct UAT by testing a website's functionality, such as filling out forms and logging in, without knowing how the software was implemented.

Here's an example of code for a white box testing scenario:

```
public class Calculator {
int add(int num1, int num2) {
return num1 + num2;
}
int subtract(int num1, int num2) {
return num1 - num2;
}
}

public class CalculatorTest {
@Test
public void testAdd() {
Calculator calculator = new Calculator();
assertEquals(10, calculator.add(6, 4)); //test that adding 6 and 4 equals 10
}
@Test
public void testSubtract() {
Calculator calculator = new Calculator();
assertEquals(2, calculator.subtract(4, 2)); //test that subtracting 4 from 2 equals 2
}
}
```

This code tests the methods of the Calculator class using the JUnit testing framework. The testAdd() method verifies that adding 6 and 4 using the add() method of the Calculator class equals 10. The testSubtract() method verifies that subtracting 4 from 2 using the subtract() method of the Calculator class equals 2. This is an example of white box testing because it tests the internal implementation of the code.

Bug Triage?

Bug triage is the process of prioritizing and categorizing bugs based on their severity, impact, and other factors. This process helps to ensure that the most critical bugs are addressed first and that resources are allocated effectively.

For example, in an Agile software development environment, the bug triage process may involve a cross-functional team that includes developers, testers, and product owners. The team may use a tool or spreadsheet to track bugs, and they may assign each bug a priority level based on its severity, impact on the user experience, and other factors. They may also categorize bugs by type (e.g., functional, performance, security) and assign them to specific team members for resolution.

Java code example for bug triaging:

If a bug is identified in an e-commerce website that is impacting the checkout process, it would be considered a critical bug with a high priority. The following Java code can be used to set the priority level:


```
if (bugType.equals(""checkout"") && bugSeverity.equals(""critical"")){
bugPriority = ""high"";
}
```

In this example, the bugType variable is used to identify that the bug is related to the checkout process, and the bugSeverity variable indicates that the bug is critical. The code then sets the bugPriority variable to ""high"" based on these conditions.

Explain severity and priority and High severity with low priority, low severity and high priority?

Severity and priority are important terms in software testing.

Severity- It refers to the degree of impact that a bug or issue will have on the system. In simple terms, it is the seriousness of an issue. For example, if a bank software has a bug that leads to money being transferred into the wrong account, it would be considered a severe issue.

Priority- It refers to the level of urgency in resolving an issue. It indicates how soon the bug or issue should be resolved, based on its impact on the system. For instance, a high priority issue would require immediate attention, while a low priority issue may be deferred or resolved at a later time.

An example of high severity and low priority issue could be the incorrect spelling of a few words in the user interface of a system. While it is a significant problem, it may not be an issue that requires an immediate fix.

On the other hand, a low severity and high priority issue could be a UI element that is not aligned correctly. It may not harm the system's functionality, but it may take a lot of effort to fix it properly.

Code snippet in Java for log severity levels:

```
Logger logger = LoggerFactory.getLogger(MyClass.class);
logger.debug(""This is a debug level log."");
logger.info(""This is an info level log."");
logger.warn(""This is a warning level log."");
logger.error(""This is an error level log."");
logger.fatal(""This is a fatal level log."");
```

In the above code, we have declared a logger object and used different severity levels such as debug, info, warning, error, and fatal. This will allow developers to prioritize issues based on their severity levels.

Describe Scrum ceremony?

Scrum is an Agile methodology used by software development teams. Scrum ceremonies are the events or meetings that take place during a sprint to ensure that the team is aligned and working together to achieve the sprint goal.

The major Scrum ceremonies are:

1. Sprint Planning - a meeting where team decides on what can be achieved during the sprint and how it will be achieved.
Example: The team discusses and plans the tasks/challenges that need to be accomplished during the sprint.

2. Daily Stand-up - a brief meeting that happens once a day during the sprint to ensure that the team is on track and aligned to utilize their time effectively.
Example: The team gathers together for a quick 15-minute meeting to discuss their progress updates, any recent challenges or blockers and plans for the next 24 hours.

3. Sprint Review - a showcase or demo of the completed work during the sprint to the stakeholders.
Example: The team demonstrates the functionality of the product or software to the customers and important stakeholders.

4. Sprint Retrospective - an opportunity for the team to reflect on their performance in the last sprint and identify areas of improvement for upcoming sprints.
Example: The team review their performance in previous sprint, highlighting what went poorly and prioritizing improvements for the upcoming sprint.

When do you automate in current sprint or next sprint?

In agile methodology, automation is one of the core principles, and it is essential to automate as much as possible to achieve faster releases and better quality software.

The decision of whether to automate in the current sprint or the next sprint depends on the specific project's requirements and context. However, a general guideline is to automate in the same sprint as the development work. This approach ensures that the team will complete the automated tests simultaneously with the development work, which helps to identify and fix defects at an early stage.

Furthermore, if the team automates in the same sprint, they can get feedback on whether the automated tests are working well for the features they are testing. If the automated tests show defects or do not provide coverage for the expected user behavior, the team will have the entire sprint to refine the tests.

On the other hand, if the project has a tight schedule and the team cannot automate in the same sprint, they should plan to automate in the next sprint. However, in such cases, the team should ensure that they have the necessary resources, skills, and tools required for automation before starting the actual testing process.

Example: Suppose the team is working on developing a new feature that has complex business rules and a large number of test cases. In such cases, it may not be possible to automate all the test cases in the same sprint. In this situation, the team can split the test cases into high, medium, and low priority categories and start by automating the high priority ones. Once the development work is complete, the team can start automating the remaining test cases in the next sprint.

Explain velocity in sprint.

Velocity is a metric used in Agile methodology to measure the amount of work completed by a team during a sprint. It is the amount of work that team can typically finish in a given sprint, based on historical performance.

Velocity is calculated by adding up the effort estimates for all completed user stories in the previous sprints. For example, if a team completes 5 user stories in a sprint, and each story is estimated at 8 points, then the team's velocity for that sprint is 40 points.

The velocity metric helps the team to understand how much work they can realistically complete in each sprint, which can be used to plan future sprints and make adjustments to the team's capacity and workload accordingly. It can also be used to track the progress of the team's work against their overall goals.

Suppose if we give manual testing for six months or one year what will you do?

As an automation expert, I would suggest implementing automation testing to reduce the manual testing efforts and improve efficiency. This would involve analyzing the manual test cases, creating test scripts, and executing them using automation tools.

For example, if we have a regression test suite with over 500 manual test cases, we can use an automation tool like Selenium WebDriver to automate the test cases. This would reduce the time and effort required to execute the regression tests, and the tests can be run more frequently.

Additionally, I would also recommend introducing agile methodologies to enhance the testing process further. This would involve breaking down the testing tasks into smaller, manageable chunks and working in iterations. Agile testing would help to identify defects early on in the development cycle and allow for faster feedback and bug fixes.

Overall, leveraging automation testing and agile methodologies would significantly improve the efficiency and effectiveness of the testing process.

How to handle frame in Selenium?

We can handle frames in Selenium using the switchTo() method. We can switch to the frame using the name, id or index. Here is an example code in Java:

```
//Identifying the frame element
WebElement frameElement = driver.findElement(By.id(""frameId""));

//Switching to the frame
driver.switchTo().frame(frameElement);

//Performing actions inside the frame
driver.findElement(By.id(""elementInsideFrame"")).click();

//Switching back to the main window
driver.switchTo().defaultContent();
```


What will driver.getWindowHandels() return?

driver.getWindowHandles() will return a set of all the window handles available within the current session of the WebDriver. This method is very useful when working with multiple windows in a web application.

For example, to switch between different windows, we can use the getWindowHandles() method to get all the window handles. Then we can iterate over this set to get the handle of each window and switch to that window using the switchTo() method.

Code example:

```
Set<String> windowHandles = driver.getWindowHandles();

for (String handle : windowHandles) {
driver.switchTo().window(handle);
}
```


Write a code to reverse the code, without using the built-in method.

Here is an example code in Java language to reverse a string without using the built-in method:

```
public static String reverseString(String str) {
char[] charArray = str.toCharArray();
int left = 0;
int right = charArray.length - 1;

while (left < right) {
char temp = charArray[left];
charArray[left] = charArray[right];
charArray[right] = temp;
left++;
right--;
}
```

return new String(charArray);
}

You can call this function and pass the string that you want to reverse as a parameter, for example:

String original = ""Hello World"";
String reversed = reverseString(original);
System.out.println("riginal: "" + original);
System.out.println(""Reversed: "" + reversed);

Output:
Original: Hello World
Reversed: dlroW olleH

Explain the purpose of listeners? is it the selenium concept of TestNG?

Listeners are an important part of TestNG framework in Selenium automation testing. They are used to monitor and customize the behavior of test cases, by hooking into various test execution events.

The primary purpose of listeners is to help testers and developers to:

- Monitor and track the execution of test cases in real-time
- Customize the way tests are executed and reported
- Handle test failures, errors or exceptions gracefully
- Generate additional logs or metrics to aid in debugging and analysis

For example, a common use case of listeners is to capture screenshots or videos of test cases when they fail, and attach them to the test report. This can help in identifying the root cause of a test failure, and provide more context to the stakeholders.

Here's an example code snippet of how to implement a TestNG listener to capture screenshots on test failure:

java

```
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;
import org.testng.ITestResult;
import org.testng.TestListenerAdapter;

public class TestFailureListener extends TestListenerAdapter {

@Override
public void onTestFailure(ITestResult tr) {
super.onTestFailure(tr);
WebDriver driver = TestBase.getDriver(); // Get the WebDriver instance
TakesScreenshot sc = (TakesScreenshot) driver;
byte[] screenshot = sc.getScreenshotAs(OutputType.BYTES); // Capture screenshot as bytes
tr.setAttribute(""screenshot"", screenshot); // Store the screenshot in ITestResult
}
```

}

In this example, we extend the TestListenerAdapter class from TestNG, and override the `onTestFailure()` method to capture a screenshot using Selenium's `TakesScreenshot` interface. We then store the screenshot as a byte array in the `ITestResult` object, which can be accessed later by the reporting framework.

Types of Listeners?

In TestNG, there are several types of listeners that can be used to perform specific actions during different stages of the test execution process.

1. ITestListener - This listener is used to perform actions before and after a test method is executed.

For example, you can use the following code to implement ITestListener in Java:

```
public class MyTestListener implements ITestListener {
@Override
public void onTestStart(ITestResult result) {
System.out.println(""Test started: "" + result.getName());
}

@Override
public void onTestSuccess(ITestResult result) {
System.out.println(""Test passed: "" + result.getName());
}

@Override
public void onTestFailure(ITestResult result) {
System.out.println(""Test failed: "" + result.getName());
}
//Other methods can also be implemented to perform actions on test skip or test finish
}
```


2. ISuiteListener - This listener is used to perform actions before and after a test suite is executed.

For example, you can use the following code to implement ISuiteListener in Java:

```
public class MySuiteListener implements ISuiteListener {
@Override
public void onStart(ISuite suite) {
System.out.println(""Suite started: "" + suite.getName());
}

@Override
public void onFinish(ISuite suite) {
System.out.println(""Suite finished: "" + suite.getName());
```

```
}
}
```

3. IAnnotationTransformer - This listener is used to modify the annotations of a test or suite at runtime.

For example, you can use the following code to implement IAnnotationTransformer in Java:

```
public class MyAnnotationTransformer implements IAnnotationTransformer {
@Override
public void transform(ITestAnnotation annotation, Class testClass, Constructor testConstructor, Method testMethod) {
//Modify the annotation
}
}
```

4. IMethodInterceptor - This listener is used to modify the test methods that are executed within a test class.

For example, you can use the following code to implement IMethodInterceptor in Java:

```
public class MyMethodInterceptor implements IMethodInterceptor {
@Override
public List<IMethodInstance> intercept(List<IMethodInstance> methods, ITestContext context) {
List<IMethodInstance> result = new ArrayList<>();
//Filter the methods based on certain criteria and add them to the result list
return result;
}
}
```


How will you handle dependencies in Maven at run time?

In Maven, dependencies are managed using a Project Object Model (POM) file, which defines the project's dependencies and their version numbers. During runtime, Maven downloads the necessary dependencies from a repository to the local machine.

There are two types of dependencies in Maven: compile-time and runtime.

- Compile-time dependencies: These are required for the project's source code to compile. They are automatically included in the classpath during build time.
- Runtime dependencies: These are required for the application to run. Maven does not include these dependencies in the classpath during build time. Instead, they are downloaded at runtime from the repository.

To handle runtime dependencies, Maven provides a plugin called the Maven Dependency Plugin. This plugin allows you to list all the runtime dependencies of the project and download them to a specified location.

Example code for downloading runtime dependencies using Maven Dependency Plugin in Java:

1. Add the following code to the project's POM file:

```
<build>
```

```
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-dependency-plugin</artifactId>
<version>3.1.2</version>
<executions>
<execution>
<id>copy-dependencies</id>
<phase>package</phase>
<goals>
<goal>copy-dependencies</goal>
</goals>
<configuration>
<outputDirectory>${project.build.directory}/dependency-jars/</outputDirectory>
</configuration>
</execution>
</executions>
</plugin>
</plugins>
</build>
```


2. Run the following command to download the runtime dependencies:

mvn dependency:copy-dependencies

This will download all the runtime dependencies to the specified directory.

3. You can now add the downloaded JAR files to the classpath during runtime using the following command:

java -cp ${project.build.directory}/dependency-jars/*:${project.build.directory}/classes com.example.MyClass

This will include all the runtime dependencies along with the project's compiled classes in the classpath.

How will you configure Jenkins job?

To configure Jenkins job, follow the steps below:

1. Login to Jenkins portal with your credentials.
2. Click on the 'New Job' option to configure a new job.
3. Enter a job name and select the type of job you want to create.
4. Select 'Build Triggers' to configure how the build process will be initiated. You can choose from options like Git/SVN polling, periodic builds, or other triggers like email notifications or manual build initiation.
5. In 'Build Environment' section, configure the environment variables to be used while building. You can also choose to configure the build on a specific agent or node.
6. In 'Build' section, specify the build steps or commands that should be executed.
7. Once the configuration is done, save the job.

Example:

Let's assume you want to create a Jenkins job for a Java application. You can follow the steps mentioned above and configure the job as follows:

1. Enter a job name, say 'Java Application Build Job'.
2. Select 'Freestyle project' as the job type.
3. For build triggers, select 'Git Polling' to trigger the build process every time there is a new commit to the Git repository.
4. In 'Build Environment' section, add an environment variable 'JAVA_HOME' with the path to your Java installation directory.
5. Under 'Build' section, add a build step to compile and package the Java application using Maven. The command can be like 'mvn clean install'.
6. Save the job.

Now, every time there is a new commit to the Git repository, Jenkins will automatically trigger a build process for the Java application, compile and package it using Maven.

Difference between REST and SOAPUI.

REST (Representational State Transfer) is a web services architecture that provides a lightweight and flexible approach for communication between client and server applications over the network. REST uses HTTP protocol for communication and supports XML, JSON, and other data formats.

On the other hand, SOAPUI (Simple Object Access Protocol User Interface) is an open-source tool used for testing web services. It is a protocol-based testing tool that supports SOAP (Simple Object Access Protocol), REST, and other web services protocols.

The main difference between REST and SOAPUI is their architectural approach. REST follows a resource-based architectural style, whereas SOAPUI follows a message-based architectural style. REST uses HTTP status codes to handle errors and exceptions, while SOAPUI has its own set of error codes.

Here's a sample code in Java for making a REST API call:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

public class RestAPICall {
public static void main(String[] args) throws Exception {

URL url = new URL(""https://api.example.com/users"");
HttpURLConnection connection = (HttpURLConnection) url.openConnection();
connection.setRequestMethod(""GET"");

BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
String line;
StringBuffer response = new StringBuffer();

while ((line = in .readLine()) != null) {
```

```
response.append(line);
} in .close();

System.out.println(response.toString());
}
}
```

This code shows how to make a GET call to a REST API endpoint and fetch data.

What is difference between Authorization and Authentication?

Authentication is the process of identifying a user by providing credentials such as username and password. It checks if the user is who they claim to be. It is like a gatekeeper who allows only authorized users to enter the system. Authorization, on the other hand, is the process of granting or denying access to specific resources or actions based on the user's identity or role. It defines what a user is allowed to do once they are authenticated. It is like a bouncer who permits only authorized users to perform specific actions.

Here's an example in Java for authentication:


```
// Authentication
String username = ""john"";
String password = ""123456"";
if (username.equals(""john"") && password.equals(""123456"")) {
System.out.println(""Authentication successful"");
} else {
System.out.println(""Authentication failed"");
}
```


And an example for authorization:


```
// Authorization
public class User {
private String username;
private String role;

// getters and setters

public boolean canAccessResource(String resource) {
if (role.equals(""admin"")) {
// Checks if the user is an admin
return true;
} else if (role.equals(""user"")) {
// Checks if the user is a regular user
if (resource.equals(""read"")) {
// Only allows reading access
return true;
```

```
}
}
return false;
}
}

// Usage
User user = new User();
user.setUsername(""john"");
user.setRole(""user"");

if (user.canAccessResource(""read"")) {
// Grants access to read the resource
System.out.println(""Access granted"");
} else {
System.out.println(""Access denied"");
}
```


Difference between severity and priority?

Severity refers to the degree of seriousness of a defect or issue found during testing. It is the impact of an issue on the system under test. Priority, on the other hand, refers to the order of importance assigned to resolve the issue. It is the importance of fixing an issue based on business needs.

For example, if a login page is not functioning properly, it could be a severe issue as it prevents users from accessing the system. However, if a minor spelling mistake is found on a less important page, it may have low severity. But, if the typo is on the company's logo or tagline, it could have high priority as it affects the company image.

Here is an example code snippet that shows how to assign priority and severity to a defect using TestNG framework in Java:


```
@Test(priority = 1, severity = Severity.CRITICAL)
public void loginTest() {
//code for login test
}
```

In this code, priority has been assigned as 1 (highest) and severity as CRITICAL.

"• Difference between boundary value analysis and equivalence partitioning?
Boundary value analysis and equivalence partitioning are two test case design techniques used in software testing.

Boundary value analysis involves testing the boundary values of inputs, such as the maximum and minimum values, to ensure that the software works correctly at the limits of its input range. For example, if a software application accepts values between 1 and 100, then the boundary values could be 1 and 100. In this case, the test cases would be designed to check the behavior of the software at these values.

Equivalence partitioning involves dividing the input domain into partitions or groups of input values that behave in a similar way. The aim is to test a representative sample from each partition, as testing all values in each partition is not practical. For example, if a software application accepts values between 1 and 100, then the input values could be divided into three partitions: values less than 1, values between 1 and 100, and values greater than 100. In this case, test cases would be designed to test representative values from each partition.

Both boundary value analysis and equivalence partitioning can be used to design effective test cases that can identify defects in software. However, boundary value analysis is useful when testing the edge conditions where software often behaves in unpredictable ways. In contrast, equivalence partitioning is useful when testing large ranges of input values, as it reduces the number of test cases required while still ensuring that a representative sample is tested.

Example of Boundary value analysis:

Suppose you are testing a login page that allows a user to enter a username and password. The login page specifies that the username should be between 6 and 20 characters long. In this case, you can design test cases as follows:

- Test case 1: Enter a valid username of exactly 6 characters
- Test case 2: Enter a valid username of 20 characters
- Test case 3: Enter an invalid username of less than 6 characters
- Test case 4: Enter an invalid username of more than 20 characters

Example of Equivalence partitioning:

Suppose you are testing a search function that allows the user to search for products by price. The search function specifies that the price should be between $10 and $100. For this scenario, you can divide the input values into three partitions:

- Partition 1: Values less than $10 (invalid input)
- Partition 2: Values between $10 and $100 (valid input)
- Partition 3: Values greater than $100 (invalid input)

In this case, you can design test cases as follows:

- Test case 1: Enter price as $5 (invalid input)
- Test case 2: Enter price as $50 (valid input)
- Test case 3: Enter price as $150 (invalid input)

########## MANAS JHA ##########

Difference between white box and black box testing?

White box testing (also known as clear box testing or structural testing) is a software testing technique that examines the internal structure and implementation of the software being tested. The aim of white box testing is to ensure that all lines of code are executed and all possible paths are tested. It is typically performed by developers.

Black box testing (also known as functional testing) is a software testing technique that evaluates software functionality without knowing the internal workings of the software. The aim of black box testing is to validate the software from an end-user perspective. It is typically performed by QA testers.

Example of white box testing: Unit testing is a type of white box testing that tests individual units of code, such as classes or methods. For example, a developer may write a unit test to ensure that a particular method of a class is functioning correctly.

Example of black box testing: User acceptance testing (UAT) is a type of black box testing where end-users test the software to verify if it meets the requirements and is suitable for use. For example, a QA tester could conduct UAT by testing a website's functionality, such as filling out forms and logging in, without knowing how the software was implemented.

Here's an example of code for a white box testing scenario:

```
public class Calculator {
int add(int num1, int num2) {
return num1 + num2;
}
int subtract(int num1, int num2) {
return num1 - num2;
}
}

public class CalculatorTest {
@Test
public void testAdd() {
Calculator calculator = new Calculator();
assertEquals(10, calculator.add(6, 4)); //test that adding 6 and 4 equals 10
}
@Test
public void testSubtract() {
Calculator calculator = new Calculator();
assertEquals(2, calculator.subtract(4, 2)); //test that subtracting 4 from 2 equals 2
}
}
```

This code tests the methods of the Calculator class using the JUnit testing framework. The testAdd() method verifies that adding 6 and 4 using the add() method of the Calculator class equals 10. The testSubtract() method verifies that subtracting 4 from 2 using the subtract() method of the Calculator class equals 2. This is an example of white box testing because it tests the internal implementation of the code.

########## MANAS JHA ##########

What is difference between Validation and Verification?

Validation and verification are two distinct activities performed during software testing.

Verification is the process of evaluating the software system or component to determine whether it meets specified requirements and standards. It focuses on determining whether the software is built according to the design specifications and requirements.

On the other hand, validation is the process of evaluating the software system or component to determine whether it meets the expectations and requirements of the end user. It focuses on determining whether the software meets the customer's needs and requirements.

An example of verification is testing the login functionality of a website to ensure that it meets the specified requirements, such as verifying that the correct username and password are accepted, and incorrect ones are rejected.

An example of validation is testing the usability of the website, such as ensuring that users can easily navigate through the website and find the information they need.

Following is a code snippet in Java to perform validation of input data:

```
public boolean validateInput(String input) {
boolean isValid = false;
// Perform validation logic here
if (input != null && !input.isEmpty()) {
isValid = true;
}
return isValid;
}
```


Explain severity and priority and High severity with low priority, low severity and high priority?

Severity and priority are important terms in software testing.

Severity- It refers to the degree of impact that a bug or issue will have on the system. In simple terms, it is the seriousness of an issue. For example, if a bank software has a bug that leads to money being transferred into the wrong account, it would be considered a severe issue.

Priority- It refers to the level of urgency in resolving an issue. It indicates how soon the bug or issue should be resolved, based on its impact on the system. For instance, a high priority issue would require immediate attention, while a low priority issue may be deferred or resolved at a later time.

An example of high severity and low priority issue could be the incorrect spelling of a few words in the user interface of a system. While it is a significant problem, it may not be an issue that requires an immediate fix.

On the other hand, a low severity and high priority issue could be a UI element that is not aligned correctly. It may not harm the system's functionality, but it may take a lot of effort to fix it properly.

Code snippet in Java for log severity levels:

```
Logger logger = LoggerFactory.getLogger(MyClass.class);
logger.debug(""This is a debug level log."");
logger.info(""This is an info level log."");
logger.warn(""This is a warning level log."");
logger.error(""This is an error level log."");
logger.fatal(""This is a fatal level log."");
```

In the above code, we have declared a logger object and used different severity levels such as debug, info, warning, error, and fatal. This will allow developers to prioritize issues based on their severity levels.

When you decide to stop the testing?

In Agile methodology, testing is an ongoing process and it does not end until the application is fully developed and deployed. However, if there is a specific criteria or exit condition defined in the test plan, then testing can be stopped based on those conditions. For example, a test plan may include a success criterion of achieving a certain level of code coverage or a certain number of test cases passing. Once those criteria are met, testing can be stopped.

Here's an example of code to check if a certain number of test cases have passed:

```
if (numberOfPassingTests >= 90) {
stopTesting();
}
```


Where 'numberOfPassingTests' is a variable tracking the number of test cases that have passed and 'stopTesting()' is a method to cease testing.

When do you automate in current sprint or next sprint?

In agile methodology, automation is one of the core principles, and it is essential to automate as much as possible to achieve faster releases and better quality software.

The decision of whether to automate in the current sprint or the next sprint depends on the specific project's requirements and context. However, a general guideline is to automate in the same sprint as the development work. This approach ensures that the team will complete the automated tests simultaneously with the development work, which helps to identify and fix defects at an early stage.

Furthermore, if the team automates in the same sprint, they can get feedback on whether the automated tests are working well for the features they are testing. If the automated tests show defects or do not provide coverage for the expected user behavior, the team will have the entire sprint to refine the tests.

On the other hand, if the project has a tight schedule and the team cannot automate in the same sprint, they should plan to automate in the next sprint. However, in such cases, the team should ensure that they have the necessary resources, skills, and tools required for automation before starting the actual testing process.

Example: Suppose the team is working on developing a new feature that has complex business rules and a large number of test cases. In such cases, it may not be possible to automate all the test cases in the same sprint. In this situation, the team can split the test cases into high, medium, and low priority categories and start by automating the high priority ones. Once the development work is complete, the team can start automating the remaining test cases in the next sprint.

What is backlog in Scrum methodology?

Backlog in Scrum methodology refers to a prioritized list of tasks or features that need to be completed in order to achieve the project goals. It is a living document that is constantly updated by the product owner, based on feedback from stakeholders and the development team.

The Scrum backlog typically consists of user stories, bugs, technical tasks, and other requirements that are broken down into small, manageable chunks of work. These items are prioritized based on business value, degree of complexity, and other factors that are important to the project.

For example, a backlog for a web application could consist of items such as ""Allow users to register and log in,"" ""Implement a search functionality,"" or "optimize website speed."" Each item in the backlog is given a priority score, which is used to determine the order in which the development team will work on them.

Suppose if we give manual testing for six months or one year what will you do?

As an automation expert, I would suggest implementing automation testing to reduce the manual testing efforts and improve efficiency. This would involve analyzing the manual test cases, creating test scripts, and executing them using automation tools.

For example, if we have a regression test suite with over 500 manual test cases, we can use an automation tool like Selenium WebDriver to automate the test cases. This would reduce the time and effort required to execute the regression tests, and the tests can be run more frequently.

Additionally, I would also recommend introducing agile methodologies to enhance the testing process further. This would involve breaking down the testing tasks into smaller, manageable chunks and working in iterations. Agile testing would help to identify defects early on in the development cycle and allow for faster feedback and bug fixes.

Overall, leveraging automation testing and agile methodologies would significantly improve the efficiency and effectiveness of the testing process.

Suppose you are the leam QA and 1 new member join your team and at the same time you have a deadline to meet in next 2 or 3 days so how will you involve that new member in team so that you can utilise him/her to meet deadlines?

As the lead QA, I will first introduce the new member to the team and provide an overview of the project and its deadlines. I will assign the new member with smaller tasks that are crucial for the project and assist him/her in understanding the project structure, requirements, and testing methodology. This will help the new member to get acquainted with the project and be productive from the beginning.

Furthermore, I will create a checklist of tasks that need to be completed to meet the deadline and assign different tasks to different team members including the new member. This will ensure that each team member is aware of their responsibility and the project is progressing towards its completion.

For instance, if the project involves test automation, I will assign the new member with the task of preparing test data or creating automation scripts for smaller modules. This will free up more time for experienced team members to work on more complex scenarios or issues.

After you have run a full regression test, and find new regression bugs, which bugs would you prioritize. Bugs that suggest that functionality has regressed, or bugs that appear in new features?

As an automation expert, I would prioritize the bugs that suggest that functionality has regressed after a full regression test. These bugs may indicate that previously working functionality has been affected by changes made during development or maintenance. If these issues are not fixed, they can not only impact the existing functionality but can also affect the system's overall stability.

On the other hand, bugs that appear in new features can also be critical, but they may not necessarily pose an immediate threat to a system's stability. They can be addressed in subsequent releases or sprints, depending on their severity and impact on the overall system.

For example, suppose a regression test identifies a critical bug in a payment processing feature that is affecting the system's core functionality. In that case, it would be prioritized over a bug in a new feature such as a reporting tool with a lower severity level.

How to handle Alerts in Selenium?

In Selenium, we can handle alerts using the Alert interface. Whenever an alert is encountered, we can switch to it and perform actions on it using this interface.

Here is an example code snippet in Java:

```
// Switch to alert
Alert alert = driver.switchTo().alert();

// Get alert text
String alertText = alert.getText();

// Click OK on alert
alert.accept();

// Enter text in alert prompt
alert.sendKeys(""text"");

// Click Cancel on alert
alert.dismiss();
```


In this example, we first switch to the alert using the `driver.switchTo().alert()` method. We can then get the text of the alert using the `getText()` method. We can click on the OK button of the alert using the `accept()` method, or cancel it using the `dismiss()` method. We can also enter text in a prompt alert using the `sendKeys()` method.

Different types of Navigation Commands?

There are several types of navigation commands used in automation testing:

1. Navigation to URL: This command is used for opening a web page or website by providing its URL. For example:

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

2. Navigation commands for forward and backward: These commands are used for navigating backward and forward in the application. For example:

driver.navigate().back();

driver.navigate().forward();

3. Refresh command: This command is used to refresh the current web page. For example:

driver.navigate().refresh();

4. Navigation to specific element: This command is used to navigate to a certain web element on the page. For example, let's say we want to navigate to a button with the ID of ""exampleButton"":

WebElement element = driver.findElement(By.id(""exampleButton""));
element.click();

5. Navigation to specific frame: This command is used to navigate to a certain frame on the page. For example, let's say we want to navigate to a frame with the ID of ""exampleFrame"":

driver.switchTo().frame(""exampleFrame"");

Difference between assert and verify?

Assert and Verify are two verification mechanism in automation testing.

Assert: In assert, when a condition fails, the test is immediately aborted and marked as failed. It is used to validate a condition, and the test case is considered failed if the assertion fails.

Example:

String actualTitle = driver.getTitle();
String expectedTitle = ""Automation Testing Expert"";
Assert.assertEquals(actualTitle, expectedTitle);

Here, if the actual title and expected title are not equal, the test will fail and the assertion will not allow the execution of further steps.

Verify: In verify, when a condition fails, the test does not immediately abort. It is used to validate a condition, and the test case continues to execute even if the verification fails.

Example:

```
String actualTitle = driver.getTitle();
String expectedTitle = ""Automation Testing Expert"";
Verify.verifyEquals(actualTitle, expectedTitle);
```

Here, even if the actual title and expected title are not equal, the test will continue executing and allow the execution of further steps.

Code in Java for assert:

```
String actualTitle = driver.getTitle();
String expectedTitle = ""Automation Testing Expert"";
assert.assertEquals(actualTitle, expectedTitle);
```

Code in Java for verify:

```
String actualTitle = driver.getTitle();
String expectedTitle = ""Automation Testing Expert"";
Verify.verifyEquals(actualTitle, expectedTitle);
```


How do you manage a set of Data Tables in Selenium?

In Selenium, data tables can be managed using various approaches including using Excel files, CSV files, and data providers.

One approach is to use testng data providers in Java. Testng is a testing framework that provides data providers to feed data to test cases. Here's an example code that uses data providers to manage data tables in Selenium:


```
// Define Data Provider
@DataProvider(name=""userData"")
public Object[][] userData() {
return new Object[][] {
{""John"", ""Doe"", ""johndoe@test.com"", ""password123""},
{""Jane"", ""Doe"", ""janedoe@test.com"", ""abcdefg""}
};
}

// Test Case using Data Provider
@Test(dataProvider=""userData"")
public void registerUser(String firstName, String lastName, String email, String password) {
// Open Registration Page
driver.get(""https://www.example.com/register"");
// Fill User Registration Form
driver.findElement(By.id(""firstName"")).sendKeys(firstName);
```

```
driver.findElement(By.id(""lastName"")).sendKeys(lastName);
driver.findElement(By.id(""email"")).sendKeys(email);
driver.findElement(By.id(""password"")).sendKeys(password);
// Submit Form
driver.findElement(By.id(""submit"")).click();
// Assert Registration Success
Assert.assertEquals(driver.getCurrentUrl(), ""https://www.example.com/success"");
}
```

In this example, the `userData()` data provider returns an array of arrays that represent the data table. The `registerUser()` tests uses the data provider to feed data to the test case. The test case opens the registration page, fills the user registration form with the data provided by the data provider, submits the form, and asserts if the registration was successful.

By using data providers, managing data tables in Selenium becomes much easier and more efficient.

How to handle web tables whose values change dynamically?

One way to handle web tables whose values change dynamically is to use Selenium's ""WebDriverWait"" class. This class allows the automation script to wait for a certain condition to be met before proceeding with the next step. In the case of a dynamic web table, the condition could be the presence of a specific value or a certain number of rows or columns.

Here is an example code snippet in Java using the WebDriverWait class to wait for a specific element in a dynamic web table:


```
// find the web table element
WebElement table = driver.findElement(By.id(""tableId""));

// wait for a specific value to appear in the table
WebDriverWait wait = new WebDriverWait(driver, 10);
wait.until(ExpectedConditions.textToBePresentInElement(table, ""dynamicValue""));

// perform actions on the dynamic web table
WebElement row = table.findElement(By.xpath(""//tr[contains(., 'dynamicValue')]""));
row.click();
// ... other actions
```


This code first finds the web table element and then creates a WebDriverWait instance with a timeout of 10 seconds. It then waits for the specified value ""dynamicValue"" to be present in the table using the ""textToBePresentInElement"" condition. Once the value is found, the code performs actions on the dynamic web table such as finding a specific row and clicking on it.

Why is CSS locator faster than Xpath?

CSS locator is faster than Xpath due to the way they are processed by the browser. CSS locators are processed directly by the browser's engine and are optimized for speed, while Xpath locators require the browser to search through the entire HTML DOM tree, which can be slow and resource-intensive.

For example, consider the following HTML code:

html
<div class=""example"">
<h1>Title</h1>
Paragraph

One way to select the ""Title"" element using CSS locator would be:

css
.example > h1

Using Xpath, it would be:
xpath
//div[@class='example']/h1

In this case, the CSS locator would be faster, since it only needs to traverse one level of the DOM tree to find the ""Title"" element, while the Xpath locator needs to traverse two levels.

Error is throwing as Element not found but when I go and check that element is available in the web page? The element is not hidden so no need to use Java script executor? How do you solve this?

This could be due to a timing issue where the element is not yet fully loaded at the time of the script execution. You can try implementing a wait mechanism to allow the element to fully load before accessing it.

One example of using the explicit wait in Selenium WebDriver with Java:

WebDriverWait wait = new WebDriverWait(driver, 10);
WebElement element = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id(""elementID"")));

This code will wait for a maximum of 10 seconds for the element with ID ""elementID"" to become visible on the page. Once it is visible, the element is returned and can be interacted with.

How do you execute using headless mode?

Headless mode is a way to run tests in a browser through command line interface without displaying the browser window. This mode is particularly useful for running tests on servers and for running tests on machines that don't have graphical interfaces.

To execute using headless mode in Selenium WebDriver with Java, you first need to create a new ChromeOptions object and set the ""--headless"" option to true. Then, you need to pass this object to the ChromeDriver constructor.

Here's an example:

```
ChromeOptions options = new ChromeOptions();
options.addArguments(""--headless"");

WebDriver driver = new ChromeDriver(options);
```

This code will launch Chrome in headless mode, meaning the browser window won't be displayed. You can then use the WebDriver methods to perform automated tasks in the headless browser.

If we are using correct locator but still getting element not found error then how you will resolve this error?

There could be several reasons for the element not found error, even if we are using the correct locator. Some possible solutions are:

1. Wait for the element: There might be a delay in page load or rendering the element. We can apply an explicit wait using WebDriverWait class in Selenium to wait for the element to appear on the page.

Example:

```
WebDriverWait wait = new WebDriverWait(driver,30);

WebElement element = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id(""elementId"")));
```

2. Check if the element is present in the HTML DOM: Sometimes, the element might not be in the DOM yet or might be present in an iframe. We can check if the element is present in the DOM by using driver.getPageSource() method.

Example:

```
if(driver.getPageSource().contains(""elementId"")){

//do something if the element is present

}
```

3. Verify the correctness of the locator: There might be a mistake in the locator or the element might have changed its attribute values. We can verify the locator by using browser developer tools or some locator validation tools like SelectorGadget.

Example:

```
WebElement element = driver.findElement(By.xpath(""//input[@name='username']""));

if(element.isDisplayed()){

element.sendKeys(""username"");

}
```

If none of the above solutions work, we can also try to debug the issue by adding some logging or breakpoints to locate where the issue is.

Different ways to handle hidden elements?

There are several ways to handle hidden elements in automation testing:

1. Using WebDriver's executeScript() method: This method allows you to execute JavaScript on the page and manipulate the hidden element. For example:


```
JavascriptExecutor js = (JavascriptExecutor) driver;
WebElement hiddenElement = driver.findElement(By.id(""hidden-element""));
js.executeScript(""arguments[0].setAttribute('style', 'display:block;');"", hiddenElement);
```


2. Using Actions class: Actions class can be used to perform actions on the hidden element. For example:

```
Actions actions = new Actions(driver);
WebElement hiddenElement = driver.findElement(By.id(""hidden-element""));
actions.moveToElement(hiddenElement).click().build().perform();
```


3. Using FluentWait: FluentWait can be used to wait until the hidden element becomes visible. For example:


```
Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
.withTimeout(Duration.ofSeconds(30))
.pollingEvery(Duration.ofSeconds(5))
.ignoring(NoSuchElementException.class);
WebElement hiddenElement = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id(""hidden-element"")));
```

Note: It is important to note that manipulating hidden elements may not be ideal as they are hidden for a reason and changes to them may adversely affect the functioning of the page.

How to click right click of mouse?

To click the right button on the mouse in Java, we can use the actions class from Selenium WebDriver. Here's an example code:

```
WebElement element = driver.findElement(By.id(""elementId""));
```

```
Actions actions = new Actions(driver);
actions.contextClick(element).build().perform();
```

In this code, ""elementId"" should be replaced with the ID of the element you want to click the right mouse button on. The actions class is used to perform the contextClick method, which simulates a right-click on the element.

Alternatively, we can use the Robot class from the Java AWT (Abstract Window Toolkit) package to perform a right mouse click. Here's an example code:

```
import java.awt.Robot;
import java.awt.event.InputEvent;

Robot robot = new Robot();
robot.mousePress(InputEvent.BUTTON3_DOWN_MASK);
robot.mouseRelease(InputEvent.BUTTON3_DOWN_MASK);
```

In this code, the mousePress and mouseRelease methods simulate a right-click on the current mouse position. However, this approach is not recommended for use with Selenium WebDriver as it does not interact with the browser directly.

How will you automate Windows based application?

To automate a Windows based application, I would use a tool such as Selenium or WinAppDriver.

With Selenium, I would use the WebDriver API to locate and interact with elements within the application. For example, to click a button I would use the following Java code:


```
WebDriver driver = new ChromeDriver(); // or any other browser driver
WebElement button = driver.findElement(By.id(""buttonId""));
button.click();
```

To automate a Windows desktop application, I would use WinAppDriver. This allows me to use Selenium's WebDriver API to interact with the desktop application's UI elements. For example, to click a button in a desktop application I would use the following Java code:

```
DesiredCapabilities capabilities = new DesiredCapabilities();
capabilities.setCapability(""app"", ""C:/path/to/application.exe"");

WebDriver driver = new WindowsDriver<WindowsElement>(new URL(""http://127.0.0.1:4723""), capabilities);
WindowsElement button = (WindowsElement) driver.findElement(By.name(""buttonName""));
button.click();
```

Swap string without 3rd variable?

To swap two strings in Java without using a third variable, we can utilize the concept of concatenation and substring(). Here's an example code snippet to achieve this:

```
String str1 = ""Hello"";
String str2 = ""World"";

// Before swapping
System.out.println(""String 1: "" + str1);
System.out.println(""String 2: "" + str2);

// Swapping without third variable
str1 = str1 + str2;
str2 = str1.substring(0, str1.length() - str2.length());
str1 = str1.substring(str2.length());

// After swapping
System.out.println(""String 1: "" + str1);
System.out.println(""String 2: "" + str2);
```

Output:

```
String 1: Hello
String 2: World
String 1: World
String 2: Hello
```

In the above code, we first concatenate both strings into str1. Then, we use substring() method to extract the original values and assign them to str2 and str1 respectively.

Reverse string without using reverse function

To reverse a string without using the reverse function, we can use loops to iterate through each character in the string and append it to a new string in reverse order. Here's an example code in Java:

```
public static String reverseString(String str) {
String reversedStr = """";
for (int i = str.length() - 1; i >= 0; i--) {
reversedStr += str.charAt(i);
}
return reversedStr;
}
```

In this code, we first initialize an empty string `reversedStr`. Then we iterate through each character in the original string `str`, starting from the last character (i.e. index `str.length()-1`) and going backwards to the first character (i.e. index `0`). For each character, we append it to the `reversedStr` using the `+=` operator, which is shorthand for `reversedStr = reversedStr + str.charAt(i);`. Finally, we return the reversed string.

Example usage of this method:

```
String originalStr = ""hello world"";
String reversedStr = reverseString(originalStr);
System.out.println(reversedStr); // prints ""dlrow olleh""
```


Write code to print only the even numbers from an array.

Here's a sample Java code to print only the even numbers from an array:


```
public static void printEvenNumbers(int[] arr) {
for (int num : arr) {
if (num % 2 == 0) {
System.out.print(num + "" "");
}
}
}
```


Here, we're passing an integer array as a parameter to the `printEvenNumbers` method. Inside the method, we're using a for-each loop to iterate over each element of the array. For each element, we're checking if it's divisible by 2 using the modulo operator. If the modulo result is 0, then it's an even number, so we're printing it using the `System.out.print` method.

Example usage:

```
int[] nums = {1, 2, 3, 4, 5, 6};
printEvenNumbers(nums); // Output: 2 4 6
```


Write a Java code to identify, if the pair of strings are an Anagram or not?

An anagram is a word or phrase that can be formed by rearranging the letters of another word or phrase. To identify if two strings are anagrams or not, we can check if they consist of the same set of characters, regardless of their position.

Here's the Java code to check if two strings are anagrams or not:

```
import java.util.Arrays;

public class AnagramChecker {
public static boolean isAnagram(String s1, String s2) {
// Convert the strings to character arrays and sort them
char[] c1 = s1.toLowerCase().toCharArray();
char[] c2 = s2.toLowerCase().toCharArray();
Arrays.sort(c1);
Arrays.sort(c2);

// Compare the sorted character arrays
return Arrays.equals(c1, c2);
}

public static void main(String[] args) {
String s1 = ""LISTEN"";
String s2 = ""SILENT"";

if (isAnagram(s1, s2)) {
System.out.println(s1 + "" and "" + s2 + "" are anagrams"");
} else {
System.out.println(s1 + "" and "" + s2 + "" are not anagrams"");
}
}
}
```


In this example, we first convert the input strings to lowercase characters and then sort them using the `Arrays.sort()` method. We then compare the two sorted character arrays using the `Arrays.equals()` method.

For the input strings ""LISTEN"" and ""SILENT"", the method should return true because both strings consist of the same set of characters.

Note that this implementation is case-insensitive, meaning that it will consider ""Listen"" and ""silent"" as anagrams as well. If case sensitivity is desired, remove the `toLowerCase()` method calls.

What is the importance of the testng framework?

TestNG is an open-source automation testing framework that is widely used in the software industry for writing and executing automated tests. The key importance of the TestNG framework are:

1. Flexible Test Suite Configuration: TestNG allows the user to build and configure test suites in various ways, allowing the user to run tests the way they want.

2. Parallel Test Execution: TestNG supports parallel test execution, enabling faster test execution times and reduced test execution time.

3. Grouping and Prioritizing Tests: TestNG enables grouping and prioritizing tests to run first or at last, providing better control over test execution.

4. Rich Annotation Support: TestNG provides rich annotation support, which allows developers to add metadata to the test case, making it easy to understand the test results.

5. Easy Configuration: TestNG provides simple and easy test configuration and integration with third-party tools like Maven.

Example:

@Test annotation in TestNG Framework:

@Test annotation is the core feature of the TestNG framework. It is used to identify the method as a test method and run it as part of the test case. For example, consider the code below:

```
public class TestClass {

@Test
public void testMethod() {
// test code goes here
}
}
```


In the above example, the @Test annotation marks the method as a test method, and TestNG will execute the code inside the method as part of the test case. This allows developers to easily identify which methods are test methods, and which ones are not.

What is the purpose of testing XML

The purpose of testing XML is to ensure that the XML documents or data files are structured and formatted correctly, as well as to verify if they are readable by the application or software that will consume the XML data. XML testing can also determine if the XML files meet the required specifications and standards, as well as assess their accuracy, completeness, and consistency.

For example, using Java, we can write a unit test that checks if an XML file contains valid and expected data. We can use the following code to read the XML file and validate its structure using an XML schema:

```
@Test
public void testXmlStructure() throws Exception {
// Load the XML file
InputStream xmlInput = getClass().getClassLoader().getResourceAsStream(""test.xml"");

// Prepare the XML schema
SchemaFactory factory = SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
InputStream schemaInput = getClass().getClassLoader().getResourceAsStream(""test.xsd"");

// Create the schema object
```

```
Source schemaSource = new StreamSource(schemaInput);
Schema schema = factory.newSchema(schemaSource);

// Create the validator object
Validator validator = schema.newValidator();

// Validate the XML against the schema
validator.validate(new StreamSource(xmlInput));

// If no exception was thrown, then the XML is valid
}
```

This code reads an XML file called ""test.xml"" and validates its structure against an XML schema called ""test.xsd"". If the XML file does not match the schema, the XML validator will throw an exception, which will cause the test to fail.

Case Scenario: How to run the same method 100 times in TestNG with the same data?

To run the same method 100 times in TestNG with the same data, we can use the ""dataProvider"" attribute of the TestNG framework. The dataProvider attribute allows us to pass the same data to the same test method multiple times. Here's how we can do it:

1. Create a data provider method that returns the same data each time it is called. For example:


```
@DataProvider(name = ""myDataProvider"")
public Object[][] getData() {
Object[][] data = new Object[1][1];
data[0][0] = ""myTestData"";
return data;
}
```


2. In the test method, use the dataProvider attribute to specify the data provider method and the number of times to repeat the test. For example:

```
@Test(dataProvider = ""myDataProvider"", invocationCount = 100)
public void myTestMethod(String testData) {
// Do something with the test data...
}
```

In this example, the test method ""myTestMethod"" will be called 100 times, each time with the same test data ""myTestData"". You can modify the data provider method to return different data if needed.

Here's the complete code in Java:

```
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;
```

```
public class MyTestClass {

@DataProvider(name = ""myDataProvider"")
public Object[][] getData() {
Object[][] data = new Object[1][1];
data[0][0] = ""myTestData"";
return data;
}

@Test(dataProvider = ""myDataProvider"", invocationCount = 100)
public void myTestMethod(String testData) {
System.out.println(""Test data: "" + testData);
}
}
```


What is TestNG XML?

TestNG XML is an XML file that contains the configuration details for a TestNG test suite. It includes information such as the classes, methods, and parameters to be included in the test, as well as the test groups, dependencies, and execution order.

How to create a TestNG XML file?

To create a TestNG XML file, you need to define the configuration details using XML tags. You can use a text editor or an IDE like Eclipse to create the file, and save it with the .xml extension. Here is an example of a basic TestNG XML file:


```
xml
<!DOCTYPE suite SYSTEM ""http://testng.org/testng-1.0.dtd"" >
<suite name=""MyTestSuite"" verbose=""1"">
<test name=""MyFirstTest"">
<classes>
<class name=""com.example.TestClass1"" />
</classes>
</test>
<test name=""MySecondTest"">
<classes>
<class name=""com.example.TestClass2"" />
</classes>
</test>
</suite>
```


What are some of the commonly used tags in TestNG XML?

Some commonly used tags in TestNG XML include:

- suite: Defines the test suite and its properties.
- test: Defines a test and its properties, such as the name and parallel execution.
- classes: Defines the classes and methods to be included in the test.
- groups: Defines the test groups to be executed.

- parameters: Defines the parameters to be passed to the test methods.
- listeners: Defines the listeners to be used in the test execution.

How to specify test dependencies in TestNG XML?

You can specify test dependencies in TestNG XML using the dependsOnMethods or dependsOnGroups attributes. For example:

java
@Test(dependsOnMethods = { ""loginTest"" })
public void homePageTest() {
// Test code
}

In this example, the homePageTest method depends on the loginTest method to be executed first.

5. How to execute TestNG XML using Maven?

To execute TestNG XML using Maven, you need to include the surefire plugin in your pom.xml file, and specify the path to the TestNG XML file. Here is an example:

xml

<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-surefire-plugin</artifactId>

<version>2.22.2</version>
<configuration>
<suiteXmlFiles>
<suiteXmlFile>src/test/resources/testng.xml</suiteXmlFile>
</suiteXmlFiles>

</configuration>
</plugin>

</plugins>
</build>

In this example, the surefire plugin is configured to execute the TestNG XML file located at src/test/resources/testng.xml.

What are different testng annotations?

TestNG provides several annotations that can be used to control test execution flow and provide additional information about test cases. Some of the commonly used annotations are:

1. @Test: This is the main annotation that is used to identify a test method. It can also be used to set priority, enable/disable the test method, and define expected exceptions.

Example:

```
@Test(priority=1)
public void loginTest() {
// Test code goes here
}
```

2. @BeforeTest: This annotation is used to execute a method before any of the test methods in a test class. This is useful for setting up test data or performing any other necessary setup tasks.

Example:

```
@BeforeTest
public void setUp() {
// Test setup goes here
}
```

3. @AfterTest: This annotation is used to execute a method after all the test methods in a test class have been run. This is useful for cleaning up any resources or data created during the test.

Example:

```
@AfterTest
public void tearDown() {
// Test cleanup goes here
}
```

4. @DataProvider: This annotation is used to provide test data to a test method. A method annotated with @DataProvider must return a two-dimensional array of objects.

Example:

```
@DataProvider(name = ""testData"")
public Object[][] provideTestData() {
// Test data generation code goes here
}
```

5. @Parameters: This annotation is used to pass parameter values to a test method. The parameter values can be obtained from testng.xml or from the command line.

Example:

```
@Test
@Parameters(""username"")
public void loginTest(String username) {
// Test code goes here
}
```

These are just a few of the many annotations available in TestNG. Other annotations include @BeforeMethod, @AfterMethod, @BeforeClass, @AfterClass, @BeforeSuite, and @AfterSuite.

########################## MANAS JHA ##########################

How can you configure tests in testng?

In order to configure tests in TestNG, you need to create the testng.xml file. This file contains all the necessary information for TestNG to run your tests.

Here is an example of a testng.xml file:

```
<!DOCTYPE suite SYSTEM ""http://testng.org/testng-1.0.dtd"" >
<suite name=""My Test Suite"">
<test name=""My Test"">
<classes>
<class name=""com.mycompany.tests.MyTestClass""/>
<class name=""com.mycompany.tests.MyOtherTestClass""/>
</classes>
</test>
</suite>
```


In this example, we have defined a suite named ""My Test Suite"" with a test named ""My Test"". The test includes two classes, ""MyTestClass"" and ""MyOtherTestClass"", which contain the actual test methods.

You can also use various TestNG annotations to configure your tests, such as @Test, @BeforeTest, @AfterTest, @BeforeClass, and @AfterClass. These annotations allow you to control the order in which tests are run, set up test data, and clean up after tests.

Here is an example of a test class with TestNG annotations:


```
public class MyTestClass {

@BeforeClass
public void setUp() {
// Set up test data and environment
}

@Test
public void testMethod() {
// Perform test actions and assertions
}

@AfterClass
public void tearDown() {
// Clean up test data and environment
}
}
```


What is @dataprovider?

@DataProvider is an annotation in TestNG framework that enables data-driven testing, i.e., executing the same test method with different test data. It allows the tester to pass a set of parameters to the test method as arguments. The values are provided through this annotation to the test method as arguments. Each set of values is considered as a test case.

Here is an example of using @DataProvider in TestNG:

```
@DataProvider(name = ""testData"")
public Object[][] getData() {
return new Object[][] {
{ ""user1"", ""password1"" },
{ ""user2"", ""password2"" },
{ ""user3"", ""password3"" }
};
}

@Test(dataProvider = ""testData"")
public void loginTest(String username, String password) {
// Perform login operation using the provided credentials
// Assertion and other operations
}
```


In the above example, `getData()` method is annotated with @DataProvider and it returns a 2-D object array, i.e., the set of test data. The `loginTest()` method is annotated with @Test and dataProvider attribute is set to ""testData"". This means `loginTest()` method will be executed for each set of data returned by the `getData()` method. The test method takes two arguments, which are mapped with the values from the data provider.

Difference between @Factory and @DataProvider?

@Factory and @DataProvider are both features in TestNG, a testing framework in Java. The main difference between the two is the way they provide data to test cases.

@DataProvider is a method that is used to provide test data to test methods in a test class. It can return either a two-dimensional array or an object array where each object array contains the test data for one test case. For example:

```
@DataProvider
public Object[][] testData() {
return new Object[][] {
{ ""John"", 25 },
{ ""Jane"", 30 },
{ ""Mike"", 35 }
};
}

@Test(dataProvider = ""testData"")
public void testName(String name, int age) {
// Test code using name and age
```

```
}
```

@Factory, on the other hand, is used to create multiple instances of a test class, each with different data. It is useful when you want to run the same test multiple times with different inputs. For example:

```
@Factory
public Object[] createTests() {
return new Object[] {
new TestClass(""John"", 25),
new TestClass(""Jane"", 30),
new TestClass(""Mike"", 35)
};
}

public class TestClass {
private String name;
private int age;

public TestClass(String name, int age) {
this.name = name;
this.age = age;
}
@Test
public void testName() {
// Test code using name and age
}
}
```


In summary, @DataProvider provides test data to test methods, whereas @Factory creates multiple instances of a test class with different data.

Explain the difference between beforemethod, beforetest, and beforeclass

In TestNG, beforemethod, beforetest, and beforeclass are three important annotations used for setting up the test environment. The main difference between these three annotations is the scope of execution.

@BeforeMethod: This annotation is used to set up the environment before each test method is executed. If you have multiple test methods in your test class, the code inside this method will be executed before each test method. For example, if you want to login to your application before executing each test case, you can use the @BeforeMethod annotation.

@BeforeTest: This annotation is used to set up the environment before any test method in the test class is executed. This means that the code inside this method will be executed only once, regardless of the number of test methods in your test class. For example, if you want to launch the application and set up some global configuration properties before executing any test case, you can use the @BeforeTest annotation.

@BeforeClass: This annotation is used to set up the environment before any test method in the test class is executed, but it is executed only once for the entire test class. This means that the code inside this method is executed only once, even if you have multiple test methods in your test class. For example, if you want to

initialize a database connection or load some data before executing any test case, you can use the @BeforeClass annotation.

Here's an example code snippet to demonstrate the use of these annotations:

```
public class MyTestClass{

@BeforeClass
public void setUpClass(){
//initialize database connection or load some data
}

@BeforeTest
public void setUpTest(){
//launch the application and set up some global configuration properties
}
```



```
@BeforeMethod
public void setUpMethod(){
//login to your application
}
```



```
@Test
public void testMethod1(){
//test code
}

@Test
public void testMethod2(){
//test code
}
}
```


How you achieve parallel execution using testng?

Parallel execution can be achieved using TestNG through the use of the parallel attribute in the <test> tag of the testng.xml file. There are two types of parallel execution that can be used:

1. Test-Level Parallelism: This type of parallelism is achieved when multiple test methods are executed at the same time. It can be implemented by setting the attribute parallel to “methods” in the <test> tag of the testng.xml file. For example:

```
<!DOCTYPE suite SYSTEM ""http://testng.org/testng-1.0.dtd"">
<suite name=""Suite1"" verbose=""1"">
<test name=""Test1"" parallel=""methods"">
<classes>
<class name=""com.example.TestClass""/>
</classes>
</test>
```

```
</suite>
```

2. Suite-Level Parallelism: This type of parallelism is achieved when multiple test classes are executed at the same time. It can be implemented by setting the attribute parallel to “tests” in the <suite> tag of the testng.xml file. For example:

```
<!DOCTYPE suite SYSTEM ""http://testng.org/testng-1.0.dtd"">
<suite name=""Suite1"" verbose=""1"" parallel=""tests"">
<test name=""Test1"">
<classes>
<class name=""com.example.TestClass1""/>
</classes>
</test>
<test name=""Test2"">
<classes>
<class name=""com.example.TestClass2""/>
</classes>
</test>
</suite>
```


In addition, TestNG also supports thread pooling to execute tests in parallel. This can be achieved by setting the parameters thread-count and threadpool-size in the <suite> tag of the testng.xml file. For example:

```
<!DOCTYPE suite SYSTEM ""http://testng.org/testng-1.0.dtd"">
<suite name=""Suite1"" verbose=""1"" parallel=""tests"" thread-count=""2"" threadpool-size=""2"">
<test name=""Test1"">
<classes>
<class name=""com.example.TestClass1""/>
</classes>
</test>
<test name=""Test2"">
<classes>
<class name=""com.example.TestClass2""/>
</classes>
</test>
</suite>
```


This will create a thread pool of 2 threads which will be used to execute the tests in parallel.

Difference between after suite and before suite?

The main difference between after suite and before suite in TestNG is that the @BeforeSuite annotated method will run before the execution of all test cases in the suite while the @AfterSuite annotated method will be executed after the completion of all the test cases in the suite.

For example, if we have a suite of test cases for a website, we can use the @BeforeSuite annotation to initialize the browser and login into the website before running any test case. On the other hand, we can use the @AfterSuite annotation to close the browser and logout from the website after completing all the test cases.

Here is an example code snippet in Java:

```
@BeforeSuite
public void beforeSuite() {
// Initialize browser and login into website
}

@AfterSuite
public void afterSuite() {
// Close browser and logout from website
}
```


Syntax to perform parallel testing in TestNG and what do you write in <suite tag> also what do you mention in double quotes like parallel = " "

To perform parallel testing in TestNG, we need to follow the following syntax:

```
<suite name=""Suite1"" parallel=""tests"" thread-count=""2"">
```

Here, the parallel attribute is set to ""tests,"" which means TestNG will run all the tests in the same suite in parallel. The thread-count attribute specifies the maximum number of threads to be used for running the tests.

For example, if we have two test classes in our suite, Test1 and Test2, and we want to run them in parallel with two threads, we can use the following code:


```
<suite name=""Suite1"" parallel=""tests"" thread-count=""2"">
<test name=""Test1"">
<classes>
<class name=""com.example.Test1""/>
</classes>
</test>
<test name=""Test2"">
<classes>
<class name=""com.example.Test2""/>
</classes>
</test>
</suite>
```

In this example, Test1 and Test2 classes will be executed in parallel using two threads.

Note: The thread-count attribute value should not exceed the number of tests in the suite.

To mention the parallel parameter value in double quotes, we can use:

parallel=""tests"" or parallel=""methods"" or parallel=""classes""

For example, if we want to run all the test methods of a class in parallel, we can use:

```
<suite name=""Suite1"" parallel=""methods"" thread-count=""2"">
<test name=""Test1"">
<classes>
<class name=""com.example.Test1""/>
</classes>
</test>
</suite>
```

In this example, all the test methods of Test1 class will be executed in parallel using two threads.

In testNG, do we have multiple suite in one XML file and what If I want to run all suits?

Yes, it is possible to have multiple test suites in one TestNG XML file. To run all the suites in the file, you can execute the XML file using the TestNG framework. Here is an example code snippet:


```
import org.testng.TestNG;
import java.util.ArrayList;
import java.util.List;

public class RunTestSuites {
public static void main(String[] args) {
TestNG testng = new TestNG();
List<String> suites = new ArrayList<String>();
suites.add(""testngsuite.xml""); //Add the XML file name with path
testng.setTestSuites(suites);
testng.run();
}
}
```


In this example, we are creating an object of the TestNG class and adding the name of the XML file that contains multiple suites using the `setTestSuites()` method. Then we run the tests using the `run()` method.

This will execute all the test suites in the specified XML file.

What is invocationcount in testng?

InvocationCount in TestNG is a parameter that allows a test method to run multiple times with the same set of input parameters. This parameter specifies the number of times that a method should be invoked.

For example, consider a test method that performs login to a website with different user credentials. We can use the InvocationCount parameter to run this method multiple times with different user credentials.

Below is an example of using the InvocationCount parameter in TestNG:

```
@Test(invocationCount = 5)
public void loginTest() {
// Code to perform login
}
```

This will run the loginTest() method five times.

What is background in Cucumber?

In Cucumber, a background is a common step or set of steps that can be defined for all scenarios in a feature file. It is executed before every scenario in the feature file, providing a baseline for the steps that follow. This can be useful for setting up preconditions for all scenarios or defining common steps that otherwise would be repeated in each scenario.

For example, let's say we have a feature file with multiple scenarios that require the user to be logged in before they can perform any actions. We can define a background for the feature file that logs the user in and sets up the necessary environment, like so:


```
Feature: User profile management
Background:
Given the user is on the login page
When they enter valid credentials
Then they should be logged in

Scenario: Edit user profile
Given the user is on their profile page
When they click the ""Edit"" button
Then they should see the profile edit form

Scenario: View another user's profile
Given the user is on the user search page
When they search for another user
Then they should see that user's profile
```


In this example, the background defines the steps needed to log in the user, and is executed before every scenario in the feature file. This saves us from having to repeat the login steps in each scenario, making our tests more efficient and easier to maintain.

Difference between Scenario and Scenario Outline?

Scenario is a single instance of a test case with a specific set of inputs and expected outcomes. It represents a specific test case scenario.

Scenario Outline is a template for defining multiple related scenarios, each of which can have different input values and outcomes. It allows the same scenario to be executed with multiple sets of data. The different data sets are defined using Examples section, which provide input value to the same scenario.

Here's an example of a Scenario in Gherkin format:

Scenario: User Login
Given the user has navigated to the login page
When the user enters valid credentials
Then the user should be directed to the dashboard page

Here's an example of a Scenario Outline:

Scenario Outline: User Login with Multiple Users
Given the user has navigated to the login page
When the user enters ""<username>"" and ""<password>""
Then the user should be directed to the dashboard page

Examples:
username	password
user1	pass1
user2	pass2

In this example, the same scenario of user login is executed twice with different input data, hence the input values for username and password are defined in the Examples table.

########################## MANAS JHA ##########################

Explain retry analyzer?

Retry analyzer is a feature in TestNG, which allows the test script to retry a failed test case a specified number of times before marking it as failed. This functionality is useful when the automated tests encounter intermittent failures due to system load, network issues, or any other transitory issues.

To use the retry analyzer in TestNG, we need to implement the IRetryAnalyzer interface and provide the number of retries we want to perform in the constructor. Here is an example code:

```
import org.testng.IRetryAnalyzer;
import org.testng.ITestResult;

public class RetryAnalyzer implements IRetryAnalyzer {

private int retryCount = 0;
private final int maxRetryCount = 3;

@Override
public boolean retry(ITestResult result) {
if (retryCount < maxRetryCount) {
retryCount++;
return true;
```

```
}
return false;
}
}
```

In this example, the RetryAnalyzer class implements the IRetryAnalyzer interface and overrides the retry() method. We have set the maximum number of retries as 3 by declaring a constant variable 'maxRetryCount'. The retry() method returns true if the number of retries is less than the maximum retry count, otherwise it returns false.

We can use this retry analyzer by adding the @Test(retryAnalyzer = RetryAnalyzer.class) annotation to our test method, as shown in the following example:

```
import org.testng.annotations.Test;

public class MyTest {

@Test(retryAnalyzer = RetryAnalyzer.class)
public void myTest() {
// Your Test Code Here
}
}
```


By adding this annotation, TestNG will automatically use the RetryAnalyzer class to retry the test method if it fails.

Difference between hooks and tags?

Hooks and tags are features provided by the Cucumber framework, which is an essential tool for test automation using behavior-driven development (BDD) methodology.

Hooks are a way to execute code before or after a scenario, feature or step definition in the Cucumber test run. They are essentially the code blocks that run automatically based on certain trigger events such as Before or After. Hooks are typically used for setting up the test environment, establishing database connections, initializing test data, and performing cleanup tasks. Here's an example of a hook in Java:

```
import cucumber.api.java.Before;
import cucumber.api.java.After;

public class TestHooks {
@Before
public void setUp() {
// Code to set up test environment
}

@After
```

```
public void tearDown() {
// Code to clean up database connections, etc.
}
}
```

Tags, on the other hand, are a way to organize scenarios, features or step definitions in the Cucumber test run. Tags are metadata labels that are used to mark a scenario or feature with a certain category or attribute. Tags help testers to filter and group tests based on specific criteria and to run selective tests quickly. For example, if a tester wants to run only a subset of tests that relate to a particular module or function, they can use the --tags option in the command line to filter tests. Here's an example of how tags are used in feature file:

```
@smoke-test @homepage
Feature: Website Navigation
As a user, I want to navigate through web pages easily

@smoke-test @homepage
Scenario: Verify homepage links
Given user is on the homepage
When user clicks on each link
Then all links should work properly
```


Use of Maven surfire plugin. If yes, where and why?

The Maven Surefire plugin is commonly used for executing automated tests within a Maven-based project. It helps to integrate testing into the build process and provides a reliable way to run tests and generate test reports.

The Surefire plugin is usually configured in the project's pom.xml file, specifying the test classes to run, the type of test framework used (JUnit, TestNG, etc.), and any additional configurations.

Here is an example configuration for the Surefire plugin in a Maven project:

```
<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-surefire-plugin</artifactId>
<version>2.22.2</version>
<configuration>
<testFailureIgnore>true</testFailureIgnore>
<parallel>methods</parallel>
<threadCount>5</threadCount>
</configuration>
</plugin>
```

```
</plugins>
</build>
```

In this example, we configure the Surefire plugin version to 2.22.2 and set some properties such as ""testFailureIgnore"", which ignores test failures, ""parallel"", which specifies to run tests in parallel, and ""threadCount"", which sets the number of threads to use.

Overall, the Maven Surefire plugin is useful for automating test runs and generating accurate test reports as part of a continuous integration and delivery process.

What is the use of pom.xml?

POM (Project Object Model) is a fundamental part of Maven, which is a build automation and project management tool for Java projects. The pom.xml file contains information about the project and configuration details for building the project such as dependencies, plugin configuration, and project details.

Here is an example of a pom.xml file that sets the project version, dependencies, and plugins:

```
xml
<project xmlns=""http://maven.apache.org/POM/4.0.0""
xmlns:xsi=""http://www.w3.org/2001/XMLSchema-instance""
xsi:schemaLocation=""http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>
<artifactId>project-name</artifactId>
<version>1.0-SNAPSHOT</version>

<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.12</version>
<scope>test</scope>
</dependency>
<dependency>
<groupId>com.google.guava</groupId>
<artifactId>guava</artifactId>
<version>27.0-jre</version>
</dependency>
</dependencies>

<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
```

```
<version>3.8.0</version>
<configuration>
<source>1.8</source>
<target>1.8</target>
</configuration>
</plugin>
</plugins>
</build>

</project>
```

The pom.xml file is commonly used in CI/CD (continuous integration/continuous deployment) pipelines to automate the building, testing, and deployment of Java projects. When the project is built using Maven, the pom.xml file is used to download the required dependencies and plugins, and configure the build process.

Consider you have to write test/suite for different environments(qa, preproduction, production) and pass different set of data for each environment. How will you do it using maven file(Pom.xml)?

To pass different sets of data for different environments, we can make use of Maven profiles in the POM.XML file. We can define different profiles for different environments and define the properties accordingly.

For example, in the below POM.XML file, we have defined two profiles - one for QA environment and the other for Production environment. We have defined the properties with different values for each environment.

```
<profiles>
<profile>
<id>qa</id>
<properties>
<url>http://qa.example.com</url>
<username>qauser</username>
<password>qapassword</password>
</properties>
</profile>
<profile>
<id>prod</id>
<properties>
<url>http://prod.example.com</url>
<username>produser</username>
<password>prodpassword</password>
</properties>
</profile>
</profiles>
```

We can then use these properties in our test code by accessing them using the Maven build system. For example:

```
String url = System.getProperty(""url"");
String username = System.getProperty(""username"");
String password = System.getProperty(""password"");

// Use the URL, username and password to log in to the application and perform tests
```

When we run the Maven build command, we can specify the profile we want to use. For example:

```
mvn clean test -Pqa
```

This will use the QA profile and pass the corresponding properties to the test code. Similarly, we can use the 'prod' profile for production environment.

Can you give some basic commands used in maven project?

Yes, certainly. Some basic commands used in Maven projects are:

1. mvn clean: This command is used to clean the project and remove any compiled output files.

2. mvn compile: This command is used to compile the Java code in the project.

3. mvn package: This command is used to package the compiled Java code into a Jar or War file.

4. mvn install: This command is used to install the packaged application into the local Maven repository.

5. mvn test: This command is used to run the unit tests in the project.

Example:

To compile and package a Java project using Maven, we can use the following command in the terminal:

```
mvn clean compile package
```

This will first clean the project, then compile the Java code and finally package the compiled code into a Jar/War file.

Another example,if we want to run the unit tests in the project, we can use the following command:

```
mvn test
```

This will execute all the unit tests in our project, and provide output on the results of each test.

What are two components Jenkins is integrated with?

Jenkins has a wide range of integrations that make it one of the most popular automation servers. Two of the most commonly integrated components include:

1. Version Control Systems (VCS): Jenkins can easily integrate with VCS tools such as GitHub, GitLab, and Bitbucket. This integration helps automate build triggers by detecting the changes in the code repositories.

2. Testing Tools: Jenkins can integrate with various testing tools such as Selenium, JUnit, TestNG, and Cucumber to automate the testing process. Integration with these tools helps in generating test reports and tracking the results of automated tests.

Example:

The following example demonstrates how Jenkins can be integrated with a VCS tool (GitHub) in Java.

1. Create a new Jenkins project by selecting ""New Item"" from the Jenkins dashboard.

2. Choose ""Freestyle project"" and enter the project name.

3. In the ""Source Code Management"" section, select ""Git"" and provide the GitHub repository URL.

4. Configure the build triggers to automatically build the project whenever there is a new commit in the repository.

5. Add a build step to build the Java code using Maven.

6. Finally, run the Jenkins job to build and test the Java code.

How you will schedule the deployments?

As an automation expert, the scheduling of deployments can be done in various ways. One way to schedule the deployment is through Continuous Integration/Continuous Deployment (CI/CD) pipeline. This pipeline automates the build, test, and deployment process, reducing manual effort and time for deployment. Jenkins is one popular tool used for CI/CD pipeline.

Another way to schedule deployment is through tools like Kubernetes, Docker, or AWS Elastic Beanstalk, which offer automated deployment services. These tools allow configuring deployment schedules according to the needs of the team, including release cadence and firewall restrictions.

As an example, we can use Jenkins for scheduling deployment. Below is the code to create a pipeline using Jenkins in the Jenkinsfile:

```
pipeline {
agent any
stages {
stage ('Clone Repository') {
steps {
git 'https://github.com/example/repo.git'
}
```

```
}
stage ('Build') {
steps {
sh 'mvn clean package'
}
}
stage ('Deploy') {
steps {
sh 'ansible-playbook deploy.yaml'
}
}
}
}
```

In this pipeline, the first stage clones the repository, the second stage builds the code, and the third stage deploys the application using Ansible. This pipeline can be scheduled to run automatically every day or on a specific date and time.

########## MANAS JHA ##########

What is the purpose of version control tool?

########## MANAS JHA ##########

What are the git commands you have used?

As an automation expert, I have used several git commands for version control, some of which are:

1. **git clone**: This command is used to create a copy of the repository on the local machine. For example, `git clone https://github.com/username/repo.git`

2. **git add**: This command is used to stage changes before committing them. For example, `git add file.txt`

3. **git commit**: This command is used to save changes to the local repository. For example, `git commit -m ""Updated README file""`

4. **git push**: This command is used to send the committed changes to the remote repository. For example, `git push origin main`

5. **git pull**: This command is used to fetch the latest changes from the remote repository and merge them with the local repository. For example, `git pull origin main`

6. **git branch**: This command is used to create or switch to a different branch. For example, `git branch new-feature`

7. **git merge**: This command is used to merge changes from one branch to another. For example, `git merge feature-branch`

Here is an example of how these commands can be used in a Java project:

Suppose you are working on a Java project and you need to add a new feature to the application. You can start by creating a new branch for the feature using the `git branch` command:

```
git branch new-feature
```

Then, you make the necessary changes to the code and stage them using the `git add` command:

```
git add MyClass.java
```

Next, you commit the changes to the local repository using the `git commit` command:

```
git commit -m ""Added new feature to MyClass""
```


Once you have committed the changes, you can push them to the remote repository using the `git push` command:


```
git push origin new-feature
```

Finally, when the feature is complete and tested, you can merge the changes back into the main branch using the `git merge` command:


```
git checkout main
git merge new-feature
```


How to integrate postman to project?

To integrate Postman to a project, you can follow the below steps:

1. Install Postman and create an account if you haven't already.
2. Create a new API request in Postman.
3. Test the API request in Postman to make sure it works correctly.
4. Export the API request to a file.
5. Import the API request file into your project.

Here's an example code snippet in Java to import a Postman API request file into your project:

```
import com.jayway.jsonpath.JsonPath;
import io.restassured.RestAssured;
```

```
import io.restassured.response.Response;

public class PostmanIntegration {

public static void main(String[] args) {
Response response = RestAssured.given()
.get(""https://apiurl.com/users"");

String responseBody = response.getBody().asString();
String userName = JsonPath.read(responseBody, ""$[0].name"");

System.out.println(""The first user's name is: "" + userName);
}
}
```

In this example, we're using RestAssured to make a GET request to an API endpoint. We're then using JsonPath to extract the name of the first user from the response body. This is just one example of how you can integrate Postman into your project.

How will you handle dynamic payloads in API?

To handle dynamic payloads in API, we can use libraries like Jackson or Gson in Java. These libraries provide methods to convert JSON payloads into Java objects and vice versa. We can create a generic method that parses the JSON payload and maps it to a corresponding Java class. We can then use this method to handle different payloads dynamically.

For example, consider the following JSON payload:

```
{
""name"": ""John"",
""age"": 30,
""address"": {
""street"": ""123 Main St"",
""city"": ""Anytown"",
""state"": ""CA"",
""zip"": ""12345""
}
}
```

To handle this payload dynamically, we can create a Java class:

```
public class Person {
private String name;
private int age;
private Address address;

// Getters and setters
}
```

We can then use the Jackson library to parse the JSON payload and map it to the Person class:

```
ObjectMapper objectMapper = new ObjectMapper();
Person person = objectMapper.readValue(jsonPayload, Person.class);
```

We can now use the person object to perform any necessary operations. This approach allows us to handle different payloads dynamically without needing to create separate classes for each payload.

What are the API status codes, you have come across?

As an automation expert, I have worked with various API status codes, some of the commonly used codes include:

1. 200 OK - This status code indicates that the requested operation has been completed successfully.
Example: When a user requests to retrieve their profile information from a social media platform, the 200 OK status code is returned when the information is successfully retrieved.

2. 201 Created - This status code indicates that a new resource has been created successfully.
Example: If a user creates a new post on a blogging website, the 201 Created status code is returned when the post is successfully created and saved.

3. 400 Bad Request - This status code indicates that the request sent by the client is invalid or malformed.
Example: If a user tries to send a request without including a required parameter, the server responds with a 400 Bad Request status code.

4. 401 Unauthorized - This status code indicates that the client's authentication credentials are invalid or missing.
Example: If a user tries to access a secured resource on a website without providing valid login credentials, the server returns a 401 Unauthorized status code.

5. 403 Forbidden - This status code indicates that the client does not have access to the requested resource.
Example: If a user tries to access an admin-only section of a website, the server returns a 403 Forbidden status code if the user is not authorized to access that section.

6. 404 Not Found - This status code indicates that the requested resource is not available on the server.
Example: If a user tries to access a page on a website that does not exist, the server returns a 404 Not Found status code.

7. 500 Internal Server Error - This status code indicates that there was an error on the server-side while processing the request.
Example: If the server encounters an unexpected error while processing a request, it returns a 500 Internal Server Error status code.

In Java, we can handle these API status codes by using HTTP client libraries like Apache HttpClient or OkHttp. We can also use frameworks like RestAssured to test and validate these status codes in automated test cases. Here's an example code snippet for making an HTTP GET request and handling the response status codes using OkHttp:

```
// create HTTP client
OkHttpClient client = new OkHttpClient();

// create HTTP request
Request request = new Request.Builder()
.url(""https://example.com/api/resource"")
.build();

// send HTTP request and get response
Response response = client.newCall(request).execute();

// get response status code
int statusCode = response.code();

// handle different status codes
if (statusCode == 200) {
// handle successful response
} else if (statusCode == 400) {
// handle bad request
} else if (statusCode == 401) {
// handle unauthorized access
} else if (statusCode == 404) {
// handle resource not found
} else if (statusCode == 500) {
// handle server error
} else {
// handle other errors
}

// close HTTP response
response.close();
```


########## MANAS JHA ##########

What is difference between OAuth1.0 and OAuth2.O, When and where do you use and how. Can you write a sample code?

OAuth 1.0 and OAuth 2.0 are both authentication protocols used to secure APIs. However, the key difference between the two lies in their security mechanisms.

OAuth 1.0 is a token-based protocol that uses signatures to validate requests. It involves three parties: the user (resource owner), the client (consumer), and the server (service provider). In OAuth 1.0, the client needs to provide the user's credentials to the server to access protected resources. This can be a security risk if the client's credentials are compromised.

On the other hand, OAuth 2.0 is a more modern protocol that relies on SSL/TLS encryption to secure communication between parties. It uses access tokens to authenticate requests and does not require the client to provide the user's credentials to the server. Instead, the user authorizes the client to access protected resources on their behalf.

OAuth 2.0 is widely used in modern applications and APIs due to its improved security compared to OAuth 1.0.

Here is sample code for implementing OAuth 2.0 authorization with the Google API using the Google OAuth2 library in Java:

```
import com.google.auth.oauth2.GoogleCredentials;
import com.google.auth.oauth2.UserCredentials;
import com.google.auth.oauth2.AccessToken;
import com.google.auth.oauth2.GoogleCredentials.Builder;
import java.io.FileInputStream;
import java.io.IOException;

public class OAuth2Example {
public static void main(String[] args) throws IOException {
// Load the client configuration and user credentials
FileInputStream creds = new FileInputStream(""client_secret.json"");
GoogleCredentials clientCreds = GoogleCredentials.fromStream(creds);
UserCredentials userCreds = new UserCredentials.Builder(clientCreds).build();

// Get an access token to authenticate API requests
AccessToken token = userCreds.refreshAccessToken();
String accessToken = token.getTokenValue();

// Use the access token to call the Google API
// ...
}
}
```


In this example, the `client_secret.json` file contains the client configuration for the Google API. The `GoogleCredentials.fromStream()` method is used to load the client configuration, while the `UserCredentials.Builder` is used to construct the user credentials object. Finally, `userCreds.refreshAccessToken()` is called to retrieve an access token to authenticate API requests.

How you get the response from one api and send to another api?

To get the response from one API and send it to another API, you can use the Java programming language and the HTTP client libraries. Here is an example code in Java:

```
// First, make a GET request to the first API endpoint
HttpClient httpClient = HttpClient.newHttpClient();
HttpRequest request = HttpRequest.newBuilder()
.uri(URI.create(""https://api.example.com/endpoint1""))
.build();
HttpResponse<String> response = httpClient.send(request, HttpResponse.BodyHandlers.ofString());

if (response.statusCode() == 200) {
// If the response is successful, extract the data and send it to the second API endpoint
String data = response.body();
```

```
HttpClient httpClient2 = HttpClient.newHttpClient();
HttpRequest request2 = HttpRequest.newBuilder()
.uri(URI.create(""https://api.example.com/endpoint2""))
.setHeader(""Content-Type"", ""application/json"")
.POST(HttpRequest.BodyPublishers.ofString(data))
.build();
HttpResponse<String> response2 = httpClient2.send(request2, HttpResponse.BodyHandlers.ofString());

// Check the response from the second API endpoint
if (response2.statusCode() == 200) {
System.out.println(""Data successfully sent to second API endpoint."");
} else {
System.out.println(""Error sending data to second API endpoint."");
}
} else {
System.out.println(""Error getting data from first API endpoint."");
}
```

This code first makes a GET request to the first API endpoint and checks if the response is successful. If so, it extracts the data from the response body and sends it as a POST request to the second API endpoint. Finally, it checks the response from the second API endpoint to ensure that the data was successfully sent.

What is Test Plan?

A Test Plan is a document that outlines the overall approach, objectives, scope, and requirements for conducting testing on a software application or system. It includes the testing strategy, test cases, test environment, timelines and resources required to execute the testing.

For example, a Test Plan for a website may include objectives to verify the functionality of the site, the usability of its interface, and its compatibility with various browsers. It would identify specific test cases to be executed, such as entering and submitting a form, and document the expected results. The Test Plan would also outline the hardware and software requirements, timelines, and resources needed to conduct testing on the website.

Explain the bug life cycle?

The bug life cycle is a process that describes the stages through which a software defect passes from discovery to resolution. Generally, the bug life cycle has the following stages:

1. New: Initial stage when a bug is detected and reported for the first time.

2. Open: Once the bug is identified, it is assigned to the relevant team, and it becomes an open bug.

3. Assigned: Once the bug is assigned to the team, the team starts working on a resolution plan.

4. In-progress: At this stage, the development team starts working on the bug fix.

5. Resolved: Once the bug fix is completed, it undergoes testing to ensure it completely solves the problem.

6. Verified: After the bug fix is tested, the testing team will verify it to confirm it resolves the issue.

7. Closed: Once verified, the bug is marked as closed.

For example, suppose you're working on an automation project using Selenium. During the testing phase, you identify that when you click on a button, it is redirecting to the wrong page. You log a bug report, and it goes through the bug life cycle stages until it is resolved and verified by the testing team.

Difference between smoke and sanity tests?

Smoke testing and Sanity testing are two important types of testing that are performed during software testing. Both of these types of testing are performed to check the quality of the application or the software.

Smoke Testing:

Smoke Testing is performed to check the basic functionalities of the application. The main goal of Smoke Testing is to ensure that the application or software is working as expected and there are no major issues. Smoke Testing is performed after the software is developed, and it checks whether the software can perform its primary functions.

Example:
Suppose a software application is developed with a login functionality where a user enters their username and password. Smoke Testing would involve testing whether the login functionality is working correctly, like testing whether the system recognizes the correct username and password, and whether the login process is successful.

Sanity Testing:
Sanity testing is performed to check whether the new or modified features of the software have been implemented correctly without affecting the existing functionalities of the application. The goal of Sanity testing is to ensure that the application is stable enough for further testing.

Example:
Suppose an e-commerce website has a shopping cart feature that has been modified to add a discount coupon feature. Sanity Testing would involve testing whether the new feature of the discount coupon has been implemented correctly without affecting the existing shopping cart feature of the e-commerce website.

Difference between regression and retesting?

The main difference between regression testing and retesting is the scope and purpose of each type of testing. Regression testing is performed to ensure that recent changes or additions to software did not unintentionally affect existing functionality or introduce new defects. It involves running a suite of tests that cover the affected areas of the codebase, as well as other relevant parts of the system.

On the other hand, retesting is performed to verify that defects that were previously identified and fixed have indeed been resolved. It typically involves running the same test cases that failed or exhibited issues before, to ensure that the problem has been remediated.

For example, if a developer adds a new feature to a web application, regression testing would be used to ensure that the new functionality does not break any existing features that were previously working correctly. If a bug is found in the login functionality of the same application, retesting would be performed after the fix to ensure that users can now log in successfully without any issues.

Difference between Test Plan and Test Strategy?

Test Plan and Test Strategy are two important documents used in the software testing process. The main differences between these documents are as follows:

1. Test Plan: It is a formal document that defines the scope, objectives, methods, and approach to be used to test the software. This document is usually prepared by the test lead or manager and outlines the entire testing process, including the types of testing to be performed, the test environment, test schedule, and resources required.

Example: A Test Plan for a web application may include the types of testing to be performed such as functional, performance, load, usability, and security testing. It may also include the test approach, test environment, test data, testing tools, test schedule, and roles and responsibilities of each team member.

2. Test Strategy: It is a high-level document that defines the testing approach to be used by the testing team. This document is usually prepared by the project manager and outlines the overall testing objectives, testing scope, and testing procedures that will be used to achieve those objectives.

Example: A Test Strategy for a project may incorporate various types of testing such as unit testing, integration testing, system testing, and acceptance testing. It may also include the testing tools and techniques to be used, the test environment, and the roles and responsibilities of each team member.

Bug Life Cycle?

Bug Life Cycle refers to the different stages that a defect or bug goes through during its lifetime. The stages include creation, reporting, triaging, assigning, fixing, retesting, and closing. The following are the stages in detail:

1. Creation: The bug is identified by either the tester or end-user.

2. Reporting: The tester or end-user reports the bug with all necessary details such as the steps to reproduce, screenshots, and environment details.

3. Triage: The bug is reviewed by the team and prioritized based on the impact, severity, and frequency.

4. Assigning: The bug is assigned to the developer who will be responsible for fixing it.

5. Fixing: The developer fixes the bug and prepares a patch or a new build.

6. Retesting: The tester verifies the fixed bug to make sure that it is indeed fixed and also to ensure that the fix has not introduced new issues.

7. Closing: The bug is marked as resolved and the corresponding issue is closed.

Here is an example of a bug life cycle:

1. A tester finds a bug that causes the application to crash on a particular screen if the user taps on the Cancel button.

2. The tester reports the bug with all necessary details such as the steps to reproduce, screenshots, and environment details.

3. The bug is triaged, and it is given a high severity as it causes the app to crash.

4. The bug is assigned to the developer.

5. The developer fixes the code and prepares a new build.

6. The tester retests the bug and confirms that it is fixed.

7. The bug is closed as resolved.

What is exploratory testing?

Exploratory testing is an approach to software testing where the tester actively and simultaneously designs, executes, and learns from tests. It is an unscripted testing technique that is best suited for testing applications that are not well documented, are changing frequently, or when there is a lack of requirements.

An example of exploratory testing would be testing a new feature on a web application. Instead of following a set of scripted test cases, the tester would explore the feature from different angles and try to find different ways to use it. The tester would then document any issues or defects found during the testing process.

In Java, exploratory testing can be achieved using tools like JUnit, TestNG, or Cucumber. These tools provide the ability to create test scenarios on the fly as the tester is exploring an application. The tester can execute these new scenarios immediately and verify the results.

What is adhoc testing?

Ad-hoc testing is a type of software testing that is performed without any specific plan or test case in mind. In this type of testing, the tester is free to explore the software and perform any actions that they feel may uncover defects. Ad-hoc testing is an informal and usually unstructured approach to testing, which is often done with the aim of finding defects quickly. This type of testing is particularly useful in scenarios where there is not enough time to create detailed test plans and where the tester needs to leverage their experience to find defects.

For example, suppose we have a software application that allows users to create and edit documents. In an ad-hoc testing approach, a tester may first explore the user interface and try to understand what each button does. They may then try to create a document and check if all the basic functionalities like save, copy, and paste are working as expected. After that, they may try to create different versions of the document, copy and paste text from different documents to check the formatting, and see how the software behaves in various scenarios. The tester would try to identify any issues they find during the process of exploring the software.

Given the test cases having priority of -1,0,1,2 tell me the sequence of execution.

The sequence of execution for test cases with priority -1, 0, 1, and 2 negative will run first.

Explain inner join and outer join in SQL?

Inner join and outer join are two types of joins used in SQL to combine data from two or more tables.

Inner join: Inner join returns only the matching rows from both tables involved in the join. It includes the rows that have common values in both tables. For example, consider two tables 'Student' and 'Marksheet', each having a column 'Roll no'. The following SQL query will give the inner join of these two tables:

```
SELECT *
FROM Student
INNER JOIN Marksheet
ON Student.Roll no = Marksheet.Roll no;
```

Outer join: Outer join returns all the rows of one table and the matching rows of another table involved in the join. It includes the rows that do not have matching values in both tables. There are three types of outer joins:

- Left outer join: It returns all the rows of the left table and matching rows from the right table. For example, consider the same tables as above. The following SQL query will give the left outer join of these two tables:

```
SELECT *
FROM Student
LEFT OUTER JOIN Marksheet
ON Student.Roll no = Marksheet.Roll no;
```

- Right outer join: It returns all the rows of the right table and matching rows from the left table. For example, consider the same tables as above. The following SQL query will give the right outer join of these two tables:

```
SELECT *
FROM Student
RIGHT OUTER JOIN Marksheet
ON Student.Roll no = Marksheet.Roll no;
```

- Full outer join: It returns all the rows of both tables, including the ones without matching values. For example, consider the same tables as above. The following SQL query will give the full outer join of these two tables:

```
SELECT *
FROM Student
FULL OUTER JOIN Marksheet
ON Student.Roll no = Marksheet.Roll no;
```

Here is an example of how these joins can be implemented in Java:

```
String sql = ""SELECT * FROM Student FULL OUTER JOIN Marksheet ""
+ "N Student.Roll no = Marksheet.Roll no"";
Connection conn = DriverManager.getConnection(url, username, password);
```

```
PreparedStatement stmt = conn.prepareStatement(sql);
ResultSet rs = stmt.executeQuery();

while (rs.next()) {
// process the data
}
```


What is deferred bug?

Deferred bug is a bug that is not fixed immediately but is postponed to a later release or iteration. It could be due to various reasons such as low severity/priority, lack of time, limited resources or not being critical to the current release. The bug is documented and tracked to make sure it gets resolved in the future.

For example, let's say during the testing of a mobile application, a minor issue of the misspelled word in the navigation button is detected. As the priority is low and it does not affect the functionality of the application, it can be marked as a deferred bug and addressed in the next release.

What is the scrum and who is your scrum master?

Scrum is a framework for Agile software development that emphasizes adaptive planning, iterative development, and delivering working software incrementally. The Scrum Master is a role in Scrum responsible for facilitating the Scrum process and uses techniques such as sprint planning, daily stand-up meetings, sprint review, and sprint retrospectives to ensure the team is aligned and focused on delivering quality products.

An example of this in practice would be a software development team using Scrum to develop an e-commerce platform. The Scrum Master would be responsible for ensuring that sprints are planned, daily stand-up meetings are conducted, sprint reviews are held, and the sprint retrospectives are conducted to identify areas for improvement. Using Scrum, the team can deliver working increments of the e-commerce platform in a timely and efficient manner.

In tight sprint schedule, if a new requirement is added, how will you handle this situation?

I understand the importance of being flexible and adaptable to changes in requirements. In a tight sprint schedule, if a new requirement is added, I would first assess the impact of the change on the current sprint and project timeline.

If the change is critical and cannot wait, I would work with the product owner and team to prioritize it and adjust the sprint backlog accordingly. This may involve re-prioritizing the current backlog items or adding the new requirement as a separate task.

To ensure efficient testing, I would also assess the impact of the new requirement on the existing test cases and update them accordingly. Automated tests can be utilized to help expedite the testing process and ensure thorough coverage.

You have 30 + sprints in your release how will you design your test scripts and run them?

I would recommend designing and running test scripts in each sprint. This will help in identifying defects at an earlier stage and prevent them from becoming bigger issues. Here are the steps I would suggest:

1. Identify test scenarios and prioritize them in each sprint based on their criticality and impact on the application.

2. Design test cases for the identified scenarios.

3. Map the test cases to the user stories in the sprint backlog.

4. Create and maintain a test automation framework that supports both functional and non-functional testing.

5. Develop and execute automated test scripts for each identified scenario.

6. Integrate the test automation scripts with the continuous integration and delivery pipeline.

7. Monitor the test results and report defects.

To illustrate, let's take an example of an e-commerce application. In one of the sprints, the user story is to verify the payment gateway integration. Here is how I would design and run the test scripts:

1. Identify the test scenarios - Verify the payment gateway integration for positive and negative cases.

2. Design test cases - Create test cases for successful payment, payment with insufficient funds, payment with invalid card details, etc.

3. Map test cases to user stories - Link the test cases to the user story in the sprint backlog for easier tracking.

4. Create a test automation framework - Develop a framework using Java, Selenium, and TestNG to automate the identified test cases.

5. Develop and execute test scripts - Automate the payment scenarios and execute the scripts using the TestNG test runner.

6. Integrate with the CI/CD pipeline - Integrate the test automation scripts with the pipeline to execute them automatically whenever a new code is deployed.

7. Monitor and report defects - Analyze the test results and report any defects found during the payment gateway integration testing.

Overall, this approach ensures that the test scripts are designed and executed efficiently in each sprint, ensuring faster feedback and better quality of the application.

How you will be an asset to the team?

I bring a unique set of skills that can greatly benefit the team. I can help in designing and implementing effective automated testing frameworks that can boost the team's productivity and reduce the time and effort required for manual testing.

My expertise in Agile methodologies can also be valuable in ensuring that the team is following the best practices for releasing high-quality software in a timely manner. I can assist in creating user stories, conducting sprint planning, and defining acceptance criteria, among other things.

As a Java expert, I can help in developing and maintaining automated test scripts using various testing frameworks such as JUnit, TestNG, Selenium, and Appium.

Suppose you are the leam QA and 1 new member join your team and at the same time you have a deadline to meet in next 2 or 3 days so how will you involve that new member in team so that you can utilise him/her to meet deadlines?

As a QA, where do you see yourself after 3 years?

As a QA, I see myself in a leadership position after 3 years, possibly as a QA manager or team lead. I believe my extensive experience in automation testing, manual testing, agile methodologies, and Java programming will serve me well in this role. Additionally, I plan to continue my education and stay current with emerging trends and technologies in the industry to provide the best value to my team and company. One example of my leadership skills is when I trained and mentored team members on automation testing using Selenium WebDriver in Java, resulting in a more efficient and effective testing process for our projects.

What are some best practices you learnt and how much difference it made in testing career? Explain before and after situations

As an automation expert with extensive experience in manual testing, Agile methodology, and Java programming, I have learned several best practices that have helped me improve my testing career. Here are a few examples:

1. Continuous Integration and Deployment (CI/CD): Implementing CI/CD has made a significant difference in my testing career. Before adopting this practice, we used to manually build, deploy and test the application, which was time-consuming and error-prone. With CI/CD, we have automated the entire process, which has reduced the overall testing time and helped us identify issues earlier in the development cycle. This has resulted in faster releases, improved quality, and reduced costs.

2. Test Automation Framework: Developing a test automation framework has been another best practice that has made a huge difference in my testing career. Before implementing the framework, we had several automation scripts that were difficult to maintain, and it was hard to track the test results. With the framework, we have a standard way of writing and executing automation tests, which has made the tests

more robust and maintainable. Additionally, the framework provides detailed reports that help us identify the issues quickly and take corrective actions.

############################## MANAS JHA ##############################

After you have run a full regression test, and find new regression bugs, which bugs would you prioritize. Bugs that suggest that functionality has regressed, or bugs that appear in new features?

############################## MANAS JHA ##############################

Reporting tools used in testing Framework

In addition to the TestNG HTML Report, there are several other popular reporting tools and plugins available for different testing frameworks. Here are a few commonly used reporting tools in the testing ecosystem:

ExtentReports: ExtentReports is a widely used reporting library for creating interactive and detailed reports. It supports frameworks like TestNG, JUnit, and Cucumber. ExtentReports provides rich visualizations, such as charts, graphs, and test logs, to represent test execution results effectively.
Allure: Allure is a flexible reporting framework that supports various testing frameworks, including TestNG, JUnit, and Cucumber. It generates visually appealing and interactive reports with detailed information about test execution, including steps, attachments, and test history. Allure reports are easy to navigate and provide insightful analytics.
Cucumber Reports: If you are using Cucumber for behavior-driven development (BDD) testing, Cucumber provides built-in reporting options. Cucumber reports provide comprehensive information about feature files, scenarios, and their execution status. These reports can be customized to include additional information and can be integrated with other reporting tools as well.
ReportNG: ReportNG is an HTML reporting plugin for TestNG. It generates detailed and customizable HTML reports with visual representations of test execution results. ReportNG provides various features like grouping tests, test configuration details, and customization options for adding additional information to the reports.
TestNG-XSLT: TestNG-XSLT is an XSLT-based reporting tool for TestNG. It generates reports in various formats like HTML, PDF, and Excel. It offers customizable report templates and provides detailed information about test execution, including test configurations, logs, and screenshots.
Serenity BDD: Serenity BDD is a comprehensive reporting and test management tool that integrates with popular testing frameworks like JUnit and TestNG. It generates HTML reports with test execution details, including features, stories, and test steps. Serenity BDD reports provide rich visuals, including test coverage and performance metrics.
These are just a few examples of reporting tools available in the testing ecosystem. The choice of reporting tool depends on your specific requirements, testing framework, and the level of detail and customization you need in your reports.

############################## MANAS JHA ##############################

Run failed Test Cases

Steps to Create a Custom Java Annotation

The first thing that we have to do is create an Java annotation that we will to annotate our test method. Lets say the name of the annotation is ***RetryCountIfFailed***. This annotation is a pure Java annotation. Here is code to do that

```
package CustomAnnotations;
```

```
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;

@Retention(RetentionPolicy.RUNTIME)
public @interface RetryCountIfFailed  {

	// Specify how many times you want to
	// retry the test if failed.
	// Default value of retry count is 0
	int value() default 0;
}
```

You can see that there is only one variable "***value***" inside the ***RetryCountIfFailed*** annotation. This value specifies how many times a test needs to be re-executed in case of failures. Note that the default value is set to 0. Also, one key point to note is that its a runtime annotation. Hence we have specified

```
@Retention(RetentionPolicy.RUNTIME)
```


Step to Use the Custom Created Java Annotation in TestNG Automation Test

One you have the annotation its time to use the annotation in the test code. This is the update Test code that uses this annotation.

```
package Tests;

import org.testng.Assert;
import org.testng.annotations.Test;

import CustomAnnotations.RetryCountIfFailed;

public class Test001 {

	@Test
	@RetryCountIfFailed(10)
	public void Test1()
	{
		Assert.assertEquals(false, true);
	}

	@Test
	public void Test2()
	{
		Assert.assertEquals(false, true);
	}
}
```

Pay attention to the the usage of the ***RetryCountIfFailed*** annotation. First, it is used just like any other annotation by using the ***@*** in front of the name (`@RetryCountIfFailed`). Second a numeric value is passed to the annotation which specifies the number of times a failed test needs to be rerun. So the actual usage of the annotation becomes `@RetryCountIfFailed(5)`, where 5 is the number of times you want this test to be rerun in case of failure. This value can be Integer value.

Implement IRetryAnalyzer to Retry Failed Test in TestNG Framework

With this annotation being used in tests its time to update the implementation of *IRetryAnalyzer* interface. There are the two things that we will need to do once a call to ***IRetryAnalyzer::retry*** method comes

1. Check if the Test method for which retry is called has RetryCountIfFailed annotation
2. Then compare current retry attempt with value of this annotation.

Here is the new implementation of ***RetryAnalyzer Class***

```
package Listeners;

import org.testng.IRetryAnalyzer;
import org.testng.ITestResult;

import CustomAnnotations.RetryCountIfFailed;

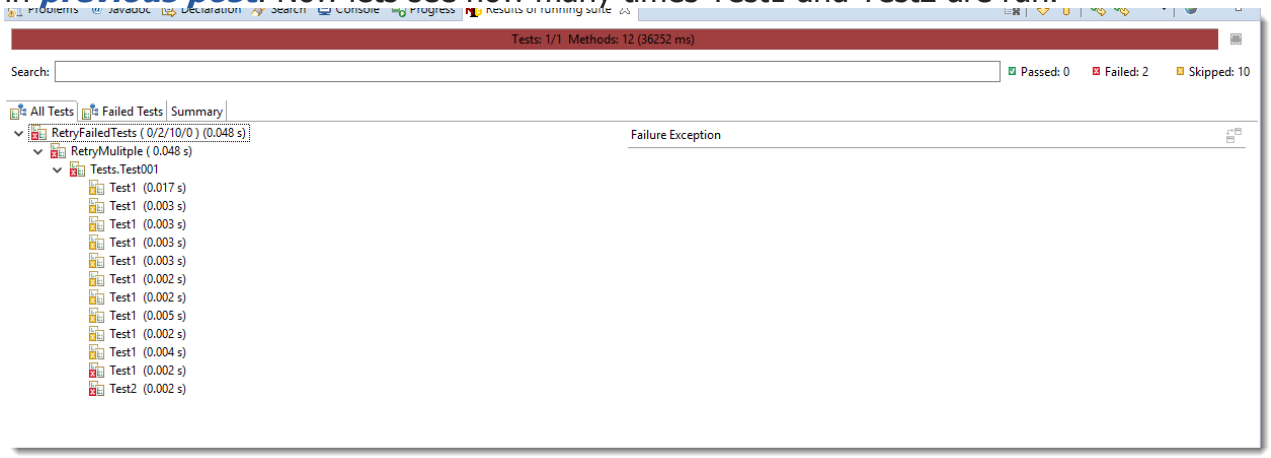
public class RetryAnalyzer implements IRetryAnalyzer {

        int counter = 0;
        /*
         * (non-Javadoc)
         *
         * @see org.testng.IRetryAnalyzer#retry(org.testng.ITestResult)
         *
         * This method decides how many times a test needs to be rerun. TestNg will
         * call this method every time a test fails. So we can put some code in here
         * to decide when to rerun the test.
         *
         * Note: This method will return true if a tests needs to be retried and
         * false it not.
         *
         */

        @Override
        public boolean retry(ITestResult result) {

                // check if the test method had RetryCountIfFailed annotation
                RetryCountIfFailed annotation =
result.getMethod().getConstructorOrMethod().getMethod()
                                .getAnnotation(RetryCountIfFailed.class);
                // based on the value of annotation see if test needs to be rerun
                if((annotation != null) && (counter < annotation.value()))
                {
                        counter++;
                        return true;
                }
                return false;
        }
}
```

This way retry decision is taken based on the count specified in the *RetryCountIfFailed* annotation. Rest of the code remains same as explained in ***previous post***. Now lets see how many times Test1 and Test2 are run.

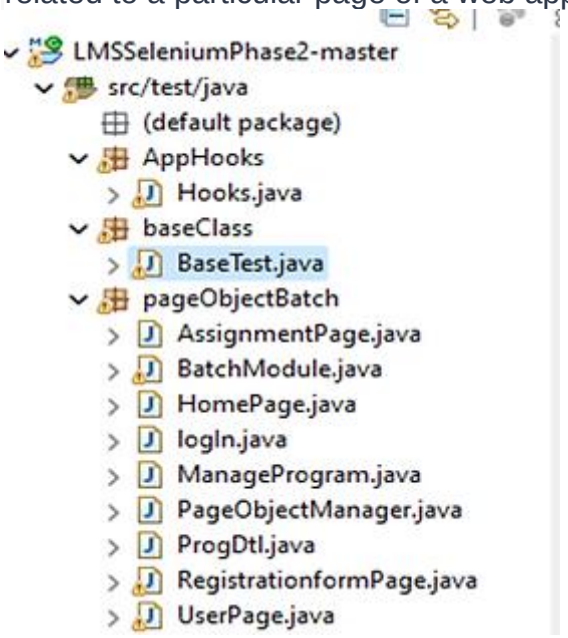


Here you can see Test1 is run 10 times and Test2 is run just once.

JAVA OOPS CONCEPTS - IMPLEMENTATION OF OOPS CONCEPT IN SELENIUM FRAMEWORK

Page object Design Pattern

In Selenium, we call objects as locators (such as ID, Name, Class Name, Tag Name, Link Text, Partial Link Text, XPath, and CSS). Object repository is a collection of objects. One of the ways to create Object Repository is to place all the locators in a separate file (i.e., properties or a PageObject file). But the best way is to use Page Object Model. In the Page Object Model Design Pattern, each web page is represented as a class. All the objects related to a particular page of a web application are stored in a class.



1. ABSTRACTION

Abstraction is the methodology of hiding the implementation of internal details and showing the functionality to the users.
Let's see an example of data abstraction in Selenium Automation Framework.
In Page Object Model design pattern, we write locators (such as id, name, xpath etc.,) and the methods in a Page Class. We utilize these locators in tests but we can't see the implementation of the methods. Literally we hide the implementations of the locators from the tests.
Example : Login function of LMS portal
A page Class "LoginPage" was created to store all the objects or locators of LoginPage Module .All functions performed on the UI of Login page are stored as methods in the same Page.

```
public class LoginPage  {

    private WebDriver driver;
    //1. by locators
    public String loginScreenUrl = "https://LMS-UI.com/login";
    public String LMSHomePage="https://LMS-UI.com/home";
    //Login Name and Password
    @FindBy (xpath="//input[@id='txtUserLoginName']")    @CacheLookup    WebElement loginName;
    @FindBy (xpath="//input[@id='txtUserPassword']")     @CacheLookup    WebElement password;
    @FindBy (xpath="//input[@id='btnLoginSubmit']") @CacheLookup    WebElement SubmitBtn;

    //2.constructor of the page class
    public LoginPage (WebDriver driver) {
        this.driver=driver;
        try {
            PageFactory.initElements(driver, this);
            } catch (Exception e) {
            System.out.println(e.getMessage());
            }
            }

    //3. page actions
        //User enters Login Name and Password
        public void loginName(String UsrName) {
        loginName.sendKeys(UsrName);
        }
        public void password(String pwd) {
        password.sendKeys(pwd);
            }
        public void loginfunction() {
            loginName.sendKeys("admin");
            password.sendKeys("password");
            SubmitBtn.click();
            alertmessageVerification();
            driver.navigate();
            System.out.println(driver.getTitle());

        }
```

Step Definitions in Login page
Here the loginpage is the object of LoginPage class and the methods are directly called.
Abstraction achieved as code and locator details are hidden

```
@Given("User is on LMS Log-In Screen")
public void user_is_on_lms_log_in_screen() {
    DriverFactory.getDriver().get("https://LMSPortal.com/login");
}

@When("User enters Login Name and Password")
public void user_enters_login_name_and_password() {
   loginpage.loginName("admin");
   loginpage.password("password");
   }
```

2. INTERFACE

WebDriver is an Interface.

WebDriver driver = new ChromeDriver();

Here, we are initializing Chrome browser using Selenium WebDriver. It means we are creating a *reference variable (driver)* of the *interface (WebDriver)* and creating an *Object*. Here *WebDriver* is an *Interface* as mentioned earlier and Chromedriver is a *class*.


```
public class BaseTest {

    public WebDriver driver;
    public Properties prop;
```


3. INHERITANCE

The mechanism in Java by which one class acquires the properties (instance variables) and functionalities of another class is known as Inheritance.

We create a Base Class in the Automation Framework to initialize WebDriver interface, WebDriver waits, Property files, Excels, etc., in the Base Class. We extend the Base Class in other classes such Tests and Utility Class.

```
public class BaseTest {
    public WebDriver driver;
    public Properties prop;
    // public static ThreadLocal<WebDriver> tdriver=new ThreadLocal<WebDriver>();
    public WebDriver driverlaunchApp() throws InterruptedException, IOException {
            prop = new Properties();
            String conPath = System.getProperty("user.dir")+"/src/test/resources/configuration/Config.properties";
            FileInputStream input = new FileInputStream(conPath);
            prop.load(input);

            String browserName = prop.getProperty("browser");
            System.out.println(browserName);

            if(driver==null)
            {
                if (browserName.equalsIgnoreCase("Chrome")) {
                    WebDriverManager.chromedriver().setup();
                    driver=new ChromeDriver();
                } else if (browserName.equalsIgnoreCase("FireFox")) {
                    WebDriverManager.firefoxdriver().setup();
                    driver = new FirefoxDriver();
                } else if (browserName.equalsIgnoreCase("Edge")) {
                    WebDriverManager.edgedriver().setup();
                    driver = new EdgeDriver();
                    }

                driver.manage().window().maximize();// Maximize the screen
                driver.manage().deleteAllCookies();//Delete all the cookies
                driver.get(prop.getProperty("url"));    // Launching the URL

                //Global wait for 10sec
                driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
            //tdriver.set(driver);
            }
        //return getDriver();
        return driver;
    }
```


4. POLYMORPHISM

Polymorphism allows us to perform a task in multiple ways.

METHOD OVERLOADING

We use **Implicit wait** in Selenium. Implicit wait is an example of overloading. In Implicit wait we use different time stamps such as SECONDS, MINUTES, HOURS etc.

```
driver.manage().window().maximize();// Maximize the screen
driver.manage().deleteAllCookies();//Delete all the cookies
driver.get(prop.getProperty("url"));// Launching the URL
driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));//Global wait for 10sec
```

Action class in TestNG is also an example of overloading.
Assert class in TestNG is also an example of overloading.
A class having multiple methods with same name but different parameters is called Method Overloading

```
@Then("Newly Added Class details should display in Manage Classes Page -Admin")
public void newly_Added_Class_details_should_display_in_Manage_Classes_Page_Admin() {
    newSize = pageObj.classSerialNo().size();
    if(newSize>classSize) {
        Assert.assertTrue(true);
    }
}
```

METHOD OVERRIDING

We use a method which was already implemented in another class by changing its parameters. To understand this you need to understand Overriding in Java.
Declaring a method in child class which is already present in the parent class is called Method Overriding. Examples are **get** and **navigate** methods of different drivers in Selenium.

5. ENCAPSULATION

All the classes in a framework are an example of Encapsulation.
In POM classes, we declare the data members using @**FindBy** and initialization of data members will be done using ***Constructor*** to utilize those in methods. Encapsulation is a mechanism of binding ode and data (variables) together in a single unit.

```
public class LoginPage  {

    private WebDriver driver;
    //1. by locators
    public String loginScreenUrl = "https://LMS-UI.com/login";
    public String LMSHomePage="https://LMS-UI.com/home";
    //Login Name and Password
    @FindBy (xpath="//input[@id='txtUserLoginName']")    @CacheLookup    WebElement loginName;
    @FindBy (xpath="//input[@id='txtUserPassword']")     @CacheLookup    WebElement password;
    @FindBy (xpath="//input[@id='btnLoginSubmit']") @CacheLookup    WebElement SubmitBtn;

    //2.constructor of the page class
    public LoginPage (WebDriver driver) {
        this.driver=driver;
        try {
            PageFactory.initElements(driver, this);
            } catch (Exception e) {
            System.out.println(e.getMessage());
            }
            }

    //3. page actions
        //User enters Login Name and Password
        public void loginName(String UsrName) {
        loginName.sendKeys(UsrName);
        }
        public void password(String pwd) {
        password.sendKeys(pwd);
            }
        public void loginfunction() {
            loginName.sendKeys("admin");
            password.sendKeys("password");
            SubmitBtn.click();
            alertmessageVerification();
            driver.navigate();
            System.out.println(driver.getTitle());

        }
```


MANAS JHA

Programs

Nth Highest Number In an Array

Certainly! Let's check the program with the given input in Java:

```
package practiceSet2;
import java.util.Arrays;


public class NthHighestNumberInArray {
    public static int findNthHighestNumber(int[] arr, int n) {
        // Sort the array in descending order
        Arrays.sort(arr);

        // Initialize count and index variables
        int count = 0;
        int index = 0;

        // Traverse the array from end to start
        for (int i = arr.length - 1; i > 0; i--) {
            // If the current element is not equal to the next element, increment
the count
            if (arr[i] != arr[i - 1]) {
                count++;
            }

            // If the count is equal to the given N, store the index and break
the loop
            if (count == n) {
                index = i;
                break;
            }
        }

        // Return the Nth highest number
        return arr[index];
    }

    public static void main(String[] args) {
        int[] arr = { -1, -1, -1, -1, -1, -3, 1, -2,-2,-2 }; // Example array {
5, 2, 8, 9, 2, 4, 7, 5,5,9 }
        int n = 3; // Find the 3rd highest number

        int nthHighest = findNthHighestNumber(arr, n);
        System.out.println("The " + n + "th highest number is: " + nthHighest);
    }
}
```

Second highest number in an array

```
package practiceSet2;


public class SecondHighestNumberInArray {
	public static int findSecondHighest(int[] arr) {

		//Integer.MIN_VALUE = -2147483648
		int largest =Integer.MIN_VALUE;
		int second_Largest = Integer.MIN_VALUE;
		//{-1, 7, 1, 34, 18}

		for(int i = 0; i < arr. length; i++) {
			if(arr[i]> largest) {
				second_Largest = largest;
				largest = arr[i];
			}

			// it will check largest ke barabar to nahi hai
			//and second largest se bada to nahi hai

			// agar hai to number ko second largest me daal do
			if(arr[i] != largest && arr[i] > second_Largest ) {
				second_Largest = arr[i];
			}
		}
		if(second_Largest == Integer. MIN_VALUE) {
			System.out.println(" Second largest element does not exist ");
		}
		else {
			System.out.println(" Second largest" + second_Largest);

		}
		return second_Largest;
	}

	public static void main(String[] args) {
		int[] numbers = {-2, -4,10, -2,-2, -2,-2, -2 };
		int secondHighest = findSecondHighest(numbers);
		System.out.println("The second highest number is: " + secondHighest);
	}
}
```

Largest Number in an Array

```
public class Testing {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		int[] arr = {2, 5, 8, 3, 1,11};

		int max = arr[0];
		// assume first element is the max

		for(int i=1; i<arr.length; i++) {
		   if(arr[i] > max) {
		      max = arr[i];
		   }
		}
		System.out.println(max);

	}

}
```


Fibonacci

```
	public static void printFibonacciSeries(int n) {
	    int first = 0, second = 1;
	    System.out.print(first + " "  + second + " ");
	    for (int i = 2; i < n; i++) {
	        int sum = first + second;
	        System.out.print(sum + " " );
	        first = second;
	        second = sum;
	    }
	}
```