Have you used the OOPS concept in your Framework ? Where and How?

1. Classes and Objects:
i. Page Object Classes were created for each unique ui component.
2. Encapsulations:
i. As Instance variables are the most important variable in your entire app code. We ensure that instance variables are private and are only accessed via getters and setters.
ii. Example POJO Classes.
3. Inheritance:
i. In order to make code reusable we implement Child and Parent Class relationship.
ii. Test Base class is parent class for test classes
iii. IJson Is an interface implement by all api pojo which helped in achieving loose coupling
4. Abstraction:
i. We have hidden the complexity of selenium, rest assured and Appium by creating browser utility classes/android utility and api helper classes
ii. These classes have wrapped the complex selenium/restassured and APPIUM code so that people who are writing test need not need to learn or deal with any of these libraries. Hence in this way abstraction was achieved.
5. Polymorphism:
i. We have ensured that function names are reusable because it helps in remembering less names and functionalities.
ii. So, for that we did method overloading. That's the compile time polymorphism
6. PRO Tip
In order to use OOPs effectively we have ensured to use Design Patterns for our classes.

Design Pattern	Comment
Singleton Design pattern	For DB Connectivity and Logger
Transfer Object Design Pattern	POJO classes
Fluent Design Pattern	For making scripts more human readable
Factory Design Pattern	For producing certain data or specify objects for our classes

Japneet Sachdeva

What are object classes and name some object class methods?

• Object class is parent class or superclass for all Classes in Java.
• Some popular functions that are present in Object class:
i. toString(): returns the string representation of this object.
ii. hashCode():returns the hashcode number for this object.
iii. equals(Object obj): compares the given object to this object.
iv. finalize(): is invoked by the garbage collector before object is being garbage
Collected.

Difference between Compile time polymorphism and run time polymorphism?

• Compile Time Polymorphism happens before the execution of the program
o Method Overloading is an example of Compile Time Polymorphism
o Method Overloading will involve only one class.
o Compile Time Polymorphism is also called as Static Polymorphism
• Runtime Polymorphism happens during the execution of a program.
o Method Overriding is an example of Run Time Polymorphism.
o Method Overriding involves 2 classes and these classes needs to have child
parent relationship.
o Runtime Polymorphism is also called Dynamic Polymorphism.

What is Serialization and Deserialization?

• When we convert a java object to a JSON object it is called as Serialization
• The reverse (When we convert a JSON object to Java Object its called as Deserialization)
• There are 2 popular 3rd party libraries that can help us in achieving this
i. JACKSON
ii. GSON

Japneet Sachdeva

Crack Test Automation interviews with Java coding: [Link]

How do you convert the API JSON Response to Java Object for validation?

• This is the purpose of Deserialization. We need to deserialize the JSON response to Java

Object and then we can do the field validation using getters or equals() functions.
Gson g = new Gson();
Customer customerReference = g.fromJson(json, Customer.class);

How to convert a Java Object to JSON object using Jackson?

• Note: Although we have used gson library for the Serialization and Deserialization.
The same result can be achieved with the JACKSON library too.
• You need to add the dependency in the pom.xml
<dependency>
<groupId>org.codehaus.jackson</groupId>
<artifactId>jackson-mapper-asl</artifactId>
<version>1.9.13</version>
</dependency>

How to convert a JSON object to a java object using Jackson?

ObjectMapper mapper = new ObjectMapper();
String response = "{'name' : abc}";
//JSON from String to Object
User user = mapper.readValue(jsonInString, User.class);
ObjectMapper mapper = new ObjectMapper();
Customer customer = new Customer();
//Object to JSON in String
String jsonInString = mapper.writeValueAsString(customer);

What is the difference between multilevel and multiple Inheritance?

a. Multilevel inheritance is consist of more than one single inheritance
b. Example GrandParent – Parent – Child
i. GrandParent- Parent is Single Inheritance
ii. Parent- Child is single Inheritance
c. When a child class tries to access the properties of 2 parent simultaneous it called as Multiple Inheritance
d. Multiple Inheritance is not directly supported in Java

Japneet Sachdeva

e. To achieve multiple inheritance, we need to use Interface
f. A class can only extend one class at a time
g. A class can implement N numbers of interfaces

Explain Singleton Design Pattern

a. Singleton Design Pattern is a way of creating only one instance of a class.
b. In Singleton design Pattern the constructor will be private.
c. There will be static reference variables of the same class.
d. We use Singleton Design Pattern for DB connectivity and Logger Creation for the our Framework

Explain POJO ?

a. POJO stands for plain old java object
b. In POJO the instance variables of the class will be private
c. There will one or more parameterized constructor
d. There will be getters and setters for the setting and retrieving the values of the instance variables
e. There will be a toString() to get the entire state of Object. (what are the current values of the instance variables)

What is the difference between List and Set?

a. Both List and Set are part of the Collection Series
b. Both of List and Set are Interface and extends the Super Interface Collection
c. List Interface is implemented by 3 classes
i. ArrayList :
1. It's a resizable array.
2. Stores the elements in continuous memory
3. Uses the concept of capacity to increase and decrease the size of the arraylist.
4. ArrayList is non-synchronized
5. Hence ArrayList is Fast in a MultiThreaded Program
ii. LinkedList:
1. Linked List is another which implements List Interface
2. Linked List also implements Queue Interface

Japneet Sachdeva

3. Linked List stores the data in a concept of node
4. A node consist of 3 things
a. Address of previous node
b. Data
c. Address to next node
5. Every node is connected with other node and the data can be stored
in different memory addresses.
• Which means they can store duplicate values in them!
d. Set:
i. Set is a data structure which does not store duplicate values in them
1. Set Interface is implemented by 2 classes
a. HashSet
i. HashSet is a class which will store the data based on
the HashCode of the elements
ii. These HashCode helps in faster retrieval of values
iii. Duplicate Values cannot be stored inside the
HashSet
b. TreeSet:
i. TreeSet is a class which also implements Set
Interface
ii. TreeSet retrieves the value either in
1. Alphabetical or Ascending order of the value
of elements

Become SDET and Future SDET Manager: Link

Architecture of Exceptions in Java?

Exception architecture in Java refers to the principles and practices used to design and implement exception handling mechanisms in your applications. It's crucial for building robust, maintainable, and user-friendly software. Here are some key aspects to consider:

1. Exception Hierarchy:

Throwable: The base class for all exceptions and errors in Java.

Japneet Sachdeva

Exception: Represents recoverable issues within the application's control, like NullPointerException or FileNotFoundException.
Error: Indicates more severe, unexpected problems outside your control, like OutOfMemoryError or StackOverflowError.

2. Exception Types:

Checked vs Unchecked:
Checked: Compiler forces you to handle them in the method signature (throws). Often related to I/O or external dependencies.
Unchecked: Compiler doesn't force handling, but ignoring them can lead to unpredictable behavior. Usually represent logic errors or invalid inputs.
Specific vs Generic:
Specific: More precise exceptions like IOException, providing detailed information about the error.
Generic: Less specific exceptions like RuntimeException, used when the exact cause is unknown.

3. Exception Handling Techniques:

Try-Catch-Finally: Blocks used to handle specific exceptions or provide cleanup logic regardless of exceptions.
Exception Chaining: Wrapping lower-level exceptions with higher-level ones for clearer context.
Custom Exceptions: Creating your own exception types to represent specific application errors.

4. Best Practices:

Avoid Over-Catch: Catching broad exceptions like Exceptions hides specific errors.
Log Exceptions: Use logging libraries to track and analyze exception occurrences.
Provide Informative Error Messages: Clearly explain the error to users and developers.
Test Exception Handling: Ensure your code handles expected exceptions gracefully.

5. Architectural Patterns:

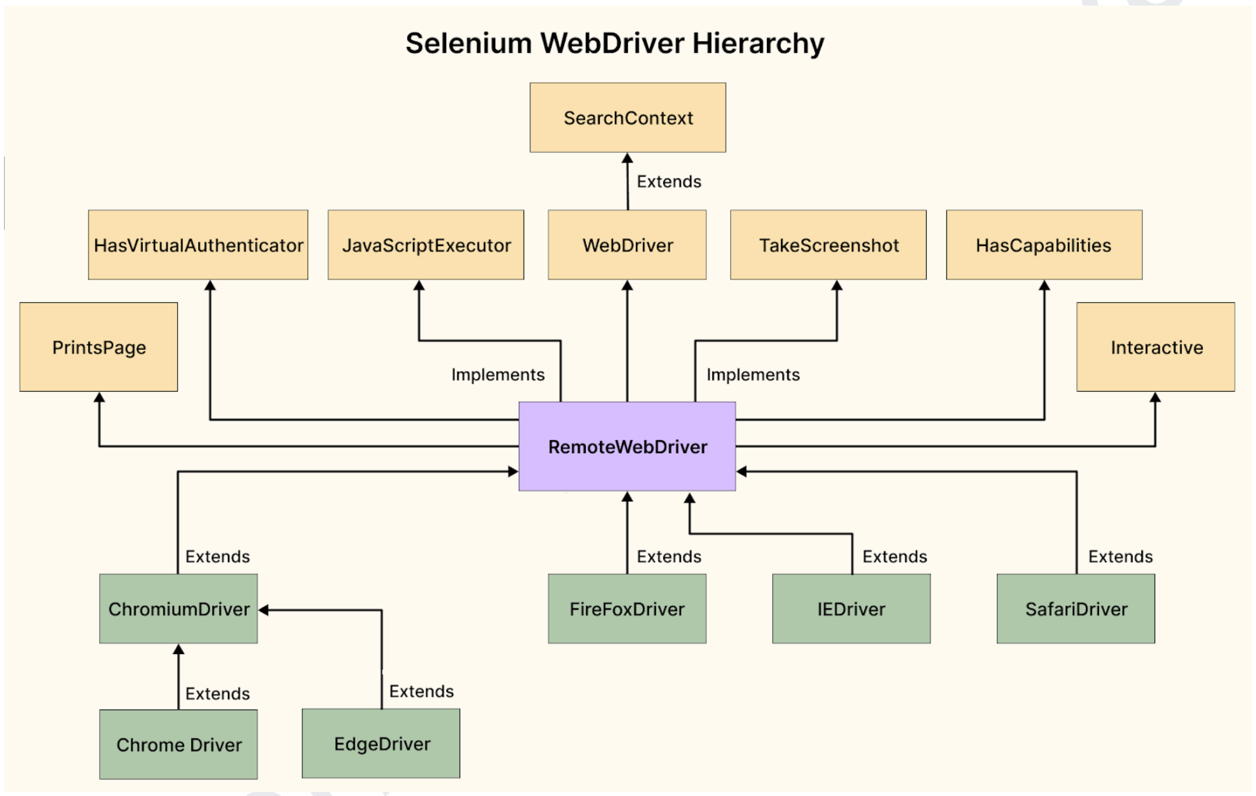
Clean Architecture: Separates business logic from external dependencies, promoting better testability and exception handling.
Layered Architecture: Each layer handles specific exceptions relevant to its functionality, minimizing error propagation.

Japneet Sachdeva

I want my class to be developed in such a way that no other class (even derived class) can create its objects. How can I do so?

If we declare the constructor of a class as private, it will not be accessible by any other class and hence, no other class will be able to instantiate it and formation of its object will be limited to itself only.

Explain Selenium WebDriver Hierarchy?



Japneet Sachdeva

Become Test Automation Architect with Full Stack QA: [Link]

What is Implicit wait?

A Simple Global Delay

What it is:

Implicit wait is a global setting that instructs Selenium WebDriver to wait for a certain amount of time before throwing a NoSuchElementException if an element is not immediately available.

It's a simple, built-in mechanism to handle potential synchronization issues during test execution.

How it works:

You set a timeout value (in seconds) using driver.manage().timeouts().implicitlyWait(time, TimeUnit.SECONDS).
Before each findElement or findElements call, WebDriver pauses for the specified time if the element isn't found immediately.
If the element still isn't found after the timeout, the exception is thrown.

Key points:

It applies to all subsequent findElement calls within the current WebDriver session.
It only waits for elements to be present, not necessarily visible or clickable.
It can slow down test execution if elements load quickly.
It can be overridden by explicit waits for specific conditions.

When to use it:

For simple scenarios where you expect minor delays in element loading.
To provide a basic level of synchronization for your tests.

When to avoid it:

When you need precise control over waiting for specific conditions like element visibility or clickability.
When you want to avoid unnecessary delays in test execution.
When you're dealing with dynamic elements or unpredictable loading times.

Best practices:

Use a moderate timeout value (e.g., 5-10 seconds).
Avoid using it alongside explicit waits as it can lead to unpredictable behavior.
Consider Fluent Wait for more granular control over waiting conditions.

Example:

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

// Subsequent findElement calls will wait up to 10 seconds
driver.findElement(By.id("myElement"));
```

Japneet Sachdeva

How to use a headless browser in selenium?

Key Points:

Headless browsers run without a visible UI, making them ideal for automated testing, web scraping, and performance optimization.
They offer faster execution, reduced resource consumption, and compatibility with environments without display capabilities (e.g., servers).

Setting Up:

Download WebDriver for Headless Mode:
Chrome: Download chromedriver for your OS and specify --headless option when creating the WebDriver instance.
Firefox: Use geckodriver and enable headless mode with options.setHeadless(true).

Creating a Headless WebDriver Instance: (JAVA)

```
System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
ChromeOptions options = new ChromeOptions();
options.addArguments("--headless");
WebDriver driver = new ChromeDriver(options);
```

Using the Headless Browser:

Use the WebDriver instance as you would with a regular browser, interacting with elements and navigating pages.
Remember, you won't see visual feedback due to the headless nature.

Benefits:

Faster Execution: No rendering of UI elements, leading to quicker test runs.
Reduced Resource Usage: Less memory and CPU consumption.
Cross-Platform Compatibility: Run tests on systems without displays.
Testing Sensitive Data: Avoid visual exposure of sensitive information.
Additional Considerations:

Debugging: Use logging or browser extensions to inspect events in the headless context.
Troubleshooting: Headless mode might have unique issues; refer to documentation for troubleshooting.
Feature Compatibility: Some browser features might not work consistently in headless mode.

Japneet Sachdeva

How to handle window based popups?

Yes, it is possible to handle Windows based pop-ups in Selenium webdriver. Sometimes on clicking a link or a button, another window gets opened. It can be a pop up with information or an advertisement.

The methods getWindowHandles and getWindowHandle are used to handle child windows. The getWindowHandles method stores all the handle ids of the opened windows in the form of Set data structure.

```
public class FirstAssign {
  public static void main(String[] args) {
    System.setProperty("webdriver.chrome.driver", "chromedriver");
    WebDriver driver = new ChromeDriver();
    //implicit wait
    driver.manage().timeouts().implicitlyWait(15, TimeUnit.SECONDS);
    //url launch
    driver.get("https://secure.indeed.com/account/login");
    driver.findElement(By.id("login-google-button")).click();
    //hold window handles
    Set<String> s = driver.getWindowHandles();
    // iterate handles
    Iterator<String> i = s.iterator();
    //child window handle id
    String c = i.next();
    //parent window handle id
    String p = i.next();
    // child window switch
    driver.switchTo().window(c);
    System.out.println("Page title of child window: "+ driver.getTitle());
    // switch to parent window
    driver.switchTo().window(p);
    System.out.println("Page title of parent window: "+ driver.getTitle());
    //browser quit
    driver.quit();
  }
```

How to handle shadow DOM elements in selenium?

Japneet Sachdeva

Shadow DOM is a functionality that allows the web browser to render DOM elements without putting them into the main document DOM tree. This creates a barrier between what the developer and the browser can reach; the developer cannot access the Shadow DOM the same way they would with nested elements, while the browser can render and modify that code the same way it would with nested elements

There are some bits of Shadow DOM terminology to be aware of:

Shadow host: The regular DOM node to which the Shadow DOM is attached.
Shadow tree: The DOM tree inside the Shadow DOM.
Shadow boundary: The place where the Shadow DOM ends and the regular DOM begins.
Shadow root: The root node of the Shadow tree.

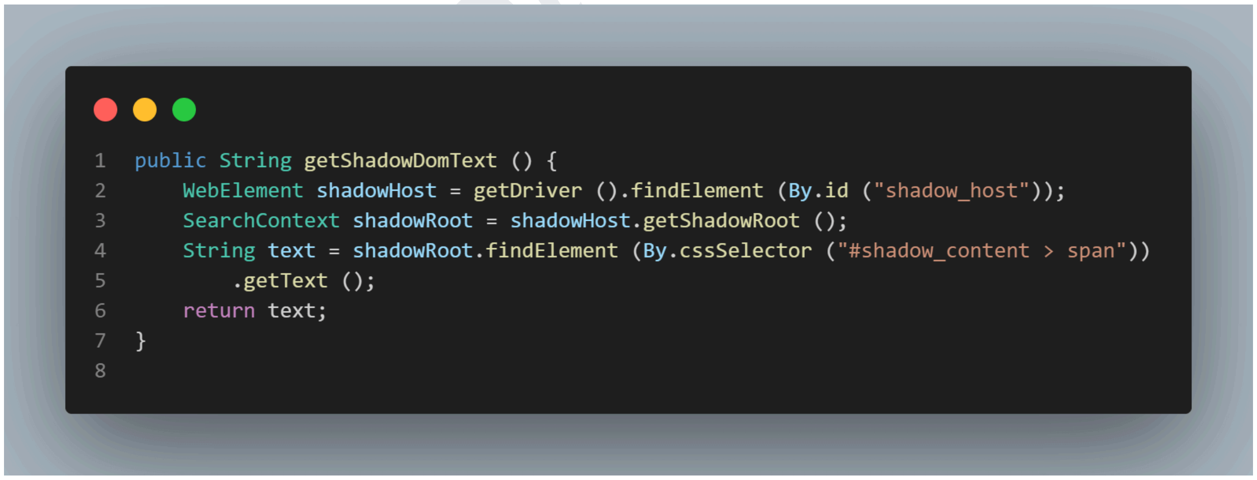
What are the ways to handle shadow DOM elements?

When we try to find the Shadow DOM elements using Selenium locators, we get NoSuchElementException as it is not directly accessible to the DOM.

Selenium WebDriver's version 4.0.0 and above, the **getShadowRoot**() method was introduced and helped locate Shadow root elements.

In case Shadow root is not found, it will throw **NoSuchShadowRootException**

How to Use getShadowRoot() →



```java
public String getShadowDomText () {
    WebElement shadowHost = getDriver ().findElement (By.id ("shadow_host"));
    SearchContext shadowRoot = shadowHost.getShadowRoot ();
    String text = shadowRoot.findElement (By.cssSelector ("#shadow_content > span"))
        .getText ();
    return text;
}
```

Use Generative AI in your Testing and Automation: Link

LinkedIn: Japneet Sachdeva

Japneet Sachdeva

Coding Questions with Solutions

How is JAVA Code compiled and executed?

Writing Code

You write the Java code using a text editor, just like writing a recipe with instructions for the computer to follow.

This code is saved as a **.java file**, containing ingredients (variables), steps (instructions), and tools (functions).

Compilation

The compiler acts as your kitchen assistant. It takes the **.java file** and checks for errors, like missing ingredients or mixing instructions.

If everything's good, it translates the recipe into a machine-readable language, creating a **.class file** (like prepared ingredients).

Execution

The Java Virtual Machine (JVM) is like your oven. It takes the **.class file** and executes the instructions step by step.

It brings your code to life, performing the actions you specified and producing the desired output (like a delicious dish!).

Bytecode: The .class file contains bytecode, a special format that the JVM understands

Just-in-Time (JIT) Compilation: The JVM often optimizes the code as it runs on the fly for better results

Notes:

Compilation happens before execution. You need to compile your code before running it
The JVM is essential for running Java code

(Setup JAVA in your system if not already)

Japneet Sachdeva

Java program to find perfect numbers

Perfect Number: a perfect number is a positive integer that is equal to the sum of its proper positive divisors, that is, the sum of its positive divisors excluding the number itself.

```
public class PerfectNumber
{
public static void main(String[] args)
{
Scanner sc = new Scanner(System.in);
int n, i = 1, sum = 0;
System.out.print("Enter a number: ");
n = sc.nextInt();
while (i < n) {
if (n % i == 0) {
sum = sum + i;
}
i++;
}
if (sum == n) {
System.out.print(i + " is a perfect number");
} else {
System.out.print(i + " is not a perfect number");
}
}
}
```

Output:

```
Enter a number: 6
6 is a perfect number
```

Japneet Sachdeva

Java program to split numbers into digits

```
public class SplitNumber
{
public static void main(String[] args)
{
```

```
int num, temp, factor = 1;
Scanner sc = new Scanner(System.in);
System.out.println("Enter a number: ");
num = sc.nextInt();
temp = num;
while (temp != 0) {
temp = temp / 10;
factor = factor * 10;
}
System.out.print("Each digits of given number are: ");
while (factor > 1) {
factor = factor / 10;
System.out.print((num / factor) + " ");

num = num % factor;
}
}
}
```

Output:

```
Enter a number: 4732
Each digits of given number are: 4 7 3 2
```

Japneet Sachdeva

Java program to add two number without using the addition operator.

```
public class Addition
{
public static void main(String[] args)
{
int a, b;
int sum;
Scanner sc = new Scanner(System.in);
System.out.print("Enter any two integers: ");
a = sc.nextInt();
b = sc.nextInt();
sum = a - ~b - 1;
System.out.print("Sum of two integers: " + sum);
}
}
```

Output:

```
Enter any two integers: 20
50
Sum of two integers: 70
```

find the frequency of each element in the array

```
public class Frequency {
    public static void main(String[] args) {
        //Initialize array
        int [] arr = new int [] {1, 2, 8, 3, 2, 2, 2, 5, 1};
        //Array fr will store frequencies of element
        int [] fr = new int [arr.length];
        int visited = -1;
        for(int i = 0; i < arr.length; i++){
            int count = 1;
            for(int j = i+1; j < arr.length; j++){
                if(arr[i] == arr[j]){
                    count++;
                    //To avoid counting same element again
                    fr[j] = visited;
                }
            }
            if(fr[i] != visited)
                fr[i] = count;
        }

        //Displays the frequency of each element present in array
        System.out.println(" Element | Frequency");
        for(int i = 0; i < fr.length; i++){
            if(fr[i] != visited)
                System.out.println("    " + arr[i] + "    |    " + fr[i]);
        }
    }}
```

Output:

Japneet Sachdeva

```
1        |       2
2        |       4
8        |       1
3        |       1
5        |       1
```

print the duplicate elements of an array

```
public class DuplicateElement {
public static void main(String[] args) {
        //Initialize array
        int [] arr = new int [] {1, 2, 3, 4, 2, 7, 8, 8, 3};
        System.out.println("Duplicate elements in given array: ");
        //Searches for duplicate element
        for(int i = 0; i < arr.length; i++) {
            for(int j = i + 1; j < arr.length; j++) {
                if(arr[i] == arr[j])
                    System.out.println(arr[j]);
            }
        }
    }
}
```

Output:

```
Duplicate elements in given array:
2
3
8
```

Use Generative AI in your Testing and Automation: Link
Become Test Automation Architect with Full Stack QA: Link
Become SDET and Future SDET Manager: Link
Crack Test Automation interviews with Java coding: Link

Japneet Sachdeva