[**Core Java All Interview Q&A:**](https://drive.google.com/file/d/1wOzYOz0FGKBqknYXT0_jXRnLStHOGDp8/view?usp=drive_link)[**JAVA CODING INTERVIEW Q&A BANK: 240 Coding Q&A**](https://drive.google.com/drive/folders/1x8CFUXmC7_3UZlsPuqa3QO2JwScM-YFL?usp=sharing)[**Top 50 JAVA Interview Q&A:**](https://drive.google.com/file/d/17gBYGa8WvKnsR4bUJsq8vD_eeotbO7o5/view?usp=sharing)

[Top 20 Java Coding Q&A](https://drive.google.com/drive/folders/1ZSqomXjYT0V0Up6dMlyi75Y0ZJm0-krO?usp=drive_link) : Most Frequently Asked

1. **How to use Java Concepts like overloading, overriding, interface, multithreading,exception handling  in Test Automation ?**  
[Check the answers here](https://www.linkedin.com/pulse/how-use-java-concepts-test-automation-sidharth-shukla-gpuhc/?trackingId=Ev6HeyKRSjmRG3MEZK%2BSjg%3D%3D)

**Basics of Object-Oriented Programming (OOPS)**

Object-Oriented Programming (OOP) is a fundamental programming paradigm used in software development, defined by its use of classes and objects.

It’s built on four main principles: Inheritance, Polymorphism, Abstraction, and Encapsulation.

These principles not only help in creating structured and reusable code but also make it easier to understand, maintain, and modify.

**Inheritance**

Inheritance allows one class to inherit the properties and methods of another class. It's a way to form a hierarchy between classes, promoting code reusability.

**Example:**

class Vehicle {

    public void startEngine() {

        System.out.println("Engine started");

    }

}

class Car extends Vehicle {

    public void openTrunk() {

        System.out.println("Trunk opened");

    }

}

public class Main {

    public static void main(String[] args) {

        Car myCar = new Car();

        myCar.startEngine(); // Inherited method

        myCar.openTrunk(); // Own method

    }

}

In this Java example, Car inherits from Vehicle.

Car can use the startEngine method from Vehicle, demonstrating inheritance.

**Polymorphism**

Polymorphism allows objects of different classes to be treated as objects of a common superclass. It’s the ability of multiple object types to implement the same functionality, which can be achieved either by method overloading or method overriding.

**Example:**

class Bird {

    public void sing() {

        System.out.println("Bird is singing");

    }

}

class Sparrow extends Bird {

    public void sing() {

        System.out.println("Sparrow is singing");

    }

}

public class Main {

    public static void main(String[] args) {

        Bird myBird = new Sparrow();

        myBird.sing(); // Outputs: Sparrow is singing

    }

}

Here, Sparrow overrides the sing method of Bird. Despite referring to Sparrow with a Bird reference, the overridden method in Sparrow is called.

**Abstraction**

Abstraction is the concept of hiding complex implementation details and showing only the necessary features of an object. It can be achieved using abstract classes and interfaces.

**Example:**

abstract class Animal {

    abstract void makeSound();

    public void eat() {

        System.out.println("Animal is eating");

    }

}

class Dog extends Animal {

    public void makeSound() {

        System.out.println("Bark");

    }

}

public class Main {

    public static void main(String[] args) {

        Animal myDog = new Dog();

        myDog.makeSound(); // Outputs: Bark

        myDog.eat(); // Inherited method

    }

}

Animal is an abstract class that provides a method makeSound().

Dog provides the specific implementation of this method.

**Encapsulation**

Encapsulation is the technique of bundling data (variables) and methods that act on the data into a single unit, often called a class, and restricting access to some of the object’s components.

class BankAccount {

    private double balance;

    public void deposit(double amount) {

        if (amount > 0) {

            balance += amount;

        }

    }

    public void withdraw(double amount) {

        if (amount <= balance) {

            balance -= amount;

        }

    }

    public double getBalance() {

        return balance;

    }

}

public class Main {

    public static void main(String[] args) {

        BankAccount account = new BankAccount();

        account.deposit(1000);

        account.withdraw(500);

        System.out.println("Balance: " + account.getBalance());

    }

}

In this example, the balance of the BankAccount is kept private. It can only be modified through the deposit and withdraw methods and read through the getBalance method, showcasing encapsulation.  
 **JAVA Scenario Based Interview Q&A:**  
  
Overloading:

Question: How would you create overloaded methods for finding elements using Selenium WebDriver?

Answer:

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

public class ElementFinder {

 public WebElement findElement(WebDriver driver, String locator) {

 return driver.findElement(By.xpath(locator));

 }

 public WebElement findElement(WebDriver driver, By locator) {

 return driver.findElement(locator);

 }

}

Overriding:

Question: Describe a scenario where you would override the toString() method in a custom WebElement class for logging purposes in Selenium.

Answer:

import org.openqa.selenium.WebElement;

public class CustomWebElement extends WebElement {

 @Override

 public String toString() {

 return "Custom element with tag name: " + this.getTagName();

 }

}\

Encapsulation:

Question: How can encapsulation be applied to manage WebDriver instances in Selenium tests?

Answer:

import org.openqa.selenium.WebDriver;

public class WebDriverManager {

 private WebDriver driver;

 public WebDriver getDriver() {

 if (driver == null) {

 // Initialize WebDriver here

 }

 return driver;

 }

}

**Inheritance:**

Question: How does the Page Object Model (POM) utilize inheritance in Selenium tests?

Answer:

public class BasePage {

 // Common page elements and methods

}

public class HomePage extends BasePage {

 // Page-specific elements and methods

}

**Enums:**

Question: Explain how you could use enums to define browser types for cross-browser testing in Selenium.

Answer:

public enum BrowserType {

 CHROME, FIREFOX, SAFARI, EDGE

}

**Generics:**

Question: How would you create a generic method to handle dynamic waits in Selenium?

Answer:

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.support.ui.WebDriverWait;

public class WaitUtils {

 public static <T> T waitFor(WebDriver driver, long timeoutInSeconds, T condition) {

 WebDriverWait wait = new WebDriverWait(driver, timeoutInSeconds);

 return wait.until(condition);

 }

}

Strings:

Question: Provide an example of using strings in Selenium to verify page titles.

Answer:

import org.openqa.selenium.WebDriver;

public class PageTitleChecker {

 public boolean isPageTitleCorrect(WebDriver driver, String expectedTitle) {

 return driver.getTitle().equals(expectedTitle);

 }

}

Array:

Question: How could you use arrays in Selenium to store multiple element locators?

Answer:

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

public class ElementLocator {

 private By[] locators;

 public ElementLocator(By... locators) {

 this.locators = locators;

 }

 public WebElement findElement(WebDriver driver) {

 for (By locator : locators) {

 WebElement element = driver.findElement(locator);

 if (element != null) {

 return element;

 }

 }

 return null;

 }

}

List:

Question: How would you use lists in Selenium to store a collection of WebElements?

Answer:

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import java.util.List;

public class ElementList {

 public List<WebElement> findElements(WebDriver driver, By locator) {

 return driver.findElements(locator);

 }

}

Map:

Question: Describe a scenario where you would use a map in Selenium to store test data for data-driven testing.

import org.openqa.selenium.WebDriver;

import java.util.Map;

public class TestData {

 public void performTest(WebDriver driver, Map<String, String> testData) {

 // Use test data for test execution

 }

}

These scenarios demonstrate how core Java concepts can be applied in the context of Selenium WebDriver for effective test automation.

INTERFACE:

**How to use Java Interface in Test Automation ?**

<https://www.linkedin.com/posts/sidharth-shukla-77b53145_sidpost-testautomation-automation-activity-7188447406403784705-Phgd?utm_source=share&utm_medium=member_desktop>

**Commonly Asked Java Interview Q&A 2024**

👉Java program to remove duplicates characters from given String.

👉Program Remove the second highest element from the HashMap.

👉Java program to Generate prime numbers between 1 & given 4 number

👉How to find the missing values from a sorted array.

👉Java program to input name, middle name and surname of a person and print only the initials.

👉Program to Print all Treemap elements?

👉What is a singleton Design Pattern? How do you implement that in your framework?

👉Write the Top 5 test cases for Booking Coupons.

👉What is serialization and deserialization?

👉What is the Difference between status codes 401 and 402?

👉Difference between selenium 3 and selenium 4?

👉What is delegate in Java and where do you use Delegate in your Framework?

👉How many maximum thread-pool can you open in the TestNG?

👉What are the Major challenges that come into the picture when you do parallel testing using TestNG and Grid?

👉How do you integrate your automation framework with the Jenkins pipeline?

👉What will happen if we remove the main method from the java program?

👉What is the component of your current Project?

👉How do you pass parameters in TestNG?

👉Write the logic of retrying the failed test case with a minimum 3 numbers of time in

Automation Testing. Which Interface do you use for it?

👉What is the OOPs concept in java?

👉Difference Between Classes and Objects?

👉What is collection in Java?

👉In How many ways can we create an object?

👉Why is Java not 100% Object-oriented?

👉Can we make a constructor as Static?

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

👉What is the difference between Abstraction Class and Interfaces?

👉Difference between String, StringBuilder, and Stringbuffer?

👉What are other immutable classes in Java apart from String?

👉Difference between TreeMap and HashMap?

👉How do you set priorities for test automation, which test needs to be automated first?

👉How do you set test case priorities for your team?

👉What are the functional things you need to test on e-commerce sites?