Page Object Model (POM) Important Questions

Question 1: Explain Page Object Model

Answer:

- The Page Object Model (POM) is a design pattern in Selenium.
- It helps create an object repository for storing web elements. Each page of the web application has a corresponding Page Class.
- Enhances test maintenance by organising code.
- Reduces code duplication by separating classes for each web page. If a web element or functionality on a page changes, you only need to update the relevant page class, not every test case that uses it.

Question 2: Explain the flow in POM when using BaseClass, TestClass, PageClass

Answer:

Base Class:

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import java.util.concurrent.TimeUnit;

public class Base {
    public static WebDriver driver;
    public static String url = "http://example.com"; // Define the
URL

    @BeforeTest
    public void invokedriver() {
        WebDriverManager.chromedriver().setup();
        driver = new ChromeDriver();
        driver.get(url);
        driver.manage().window().maximize();
    }
```

```
	@AfterTest
	public void terminateflow() {
		if (driver != null) {
			driver.quit();	// Ensure driver.quit() is only called
if the driver instance is not null
		}
	}
}
```

Page Class:

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;

public class Page {
	WebDriver driver;

	@FindBy(id = "userId")
	WebElement username;

	@FindBy(xpath = "//input[@id="password"]")
	WebElement password;

	@FindBy(id = "signBtn")
	WebElement signBtn;

	public Page(WebDriver driver) {
		this.driver = driver;
		PageFactory.initElements(driver, this);
	}

	public void enterCred() {
		username.clear();
		username.sendKeys("abc@gmail.com");
		password.clear();
		password.sendKeys("xyz12345");
		signBtn.click();
	}
```

```
}
```

Test Class:

```
import org.testng.Assert;
import org.testng.annotations.Test;

public class Test extends Base {
	public Test() {
		super();
	}

	@Test
	public void signIn() {
		Page obj = new Page(driver);
		obj.enterCred();
	}

	@Test
	public void validateDashboard() {
		String act = "actualValue";
		String exp = "expectedValue";
		Assert.assertEquals(act, exp);

		WebElement newWireBtn =
driver.findElement(By.id("newWireBtn"));
		boolean newWire = newWireBtn.isDisplayed();
		Assert.assertTrue(newWire);
	}
}
```

Question 3: Flow Summary

Answer:

- **Setup:** TestClass starts by calling `invokedriver()` from BaseClass to initialize the WebDriver, open the specified URL, and maximize the browser window.
- **BaseClass:** Initializes WebDriver, manages browser setup and teardown, and sets up logging.

- **LoginPage:** Implements Page Object Model for the login page, interacts with web elements, and logs actions.
- **Login Test:** Contains TestNG test methods for login functionality, uses assertions, and logs test actions.
- **Testng.xml:** Configures TestNG suite for executing tests and includes a listener for reporting.

Question 4: What are the advantages of using POM?

Answer:

- Reusability
- Better code maintainability
- Separation of test logic from page structure
- Less duplication of code
- If a web element or functionality on a page changes, you only need to update the relevant page class, not every test case that uses it.

Question 5: How is POM different from Page Factory?

Answer:

- POM is a design pattern, while Page Factory is a class in Selenium that provides an easier way to initialise web elements.
- Page Factory uses the `@FindBy` annotation to locate elements.

Question 6: What is the role of constructors in POM?

Answer:

- Constructors in POM are used to initialise the WebDriver instance and other necessary configurations for the page class, such as initialising elements using Page Factory.
- Web elements in POM can be initialised using `PageFactory.initElements(driver, this)` or by creating a constructor and passing the driver instance.

Question 7: Can we use POM without Page Factory?

Answer:

- Yes.

- Example:

```
public class LoginPage {
    WebDriver driver;

    // Define elements using locators
    By usernameField = By.id("username");
    By passwordField = By.id("password");
    By loginButton = By.id("loginBtn");

    // Constructor to initialise driver
    public LoginPage(WebDriver driver) {
        this.driver = driver;
    }

    // Method to perform login action
    public void login(String username, String password) {
        driver.findElement(usernameField).sendKeys(username);
        driver.findElement(passwordField).sendKeys(password);
        driver.findElement(loginButton).click();
    }
}
```

Question 8: What are the challenges faced while implementing POM?

Answer:

- Initial setup effort
- Need for clear organisation of classes for complex applications
- Managing dependencies between page classes

Question 9: What are the disadvantages of POM?

Answer:

- POM can lead to the creation of too many classes for large applications
- Increased complexity in managing them

Question 10: Can you use multiple drivers in POM?

Answer:

- Yes, but it's essential to ensure that each page class or test class has its own driver instance to avoid conflicts.
