**1.**

**2. Minimal Java Code for Simple Login Automation:**

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

public class SimpleLogin {

public static void main(String[] args) {

// Set the path for the ChromeDriver

System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");

// Create an instance of ChromeDriver

WebDriver driver = new ChromeDriver();

try {

// Open the login page

driver.get("https://example.com/login");

// Find and fill the username field

driver.findElement(By.id("username")).sendKeys("myUsername");

// Find and fill the password field

driver.findElement(By.id("password")).sendKeys("myPassword");

// Find and click the login button

driver.findElement(By.id("loginButton")).click();

// Print confirmation

System.out.println("Login attempted.");

} catch (Exception e) {

e.printStackTrace();

} finally {

// Close the browser

driver.quit();

}

}

}

**Q3.** Using GIT, create a local repository and perform various operations on it.

###### ChatGPT said:

ChatGPT

Here's a step-by-step guide on how to create a local Git repository and perform basic operations on it:

### 1. Install Git

Ensure Git is installed on your system. You can check by running:

bash

Copy code

git --version

If not installed, download and install Git from [git-scm.com](https://git-scm.com/).

### 2. Create a Local Repository

1. **Create a new project folder:**

bash

Copy code

mkdir my-git-project

cd my-git-project

1. **Initialize the Git repository:**

bash

Copy code

git init

This will create a .git directory, making my-git-project a Git repository.

### 3. Create a New File and Add Content

1. **Create a new file:**

bash

Copy code

echo "Hello, Git!" > hello.txt

1. **Check the status of the repository:**

bash

Copy code

git status

This shows that hello.txt is an untracked file.

### 4. Stage the File

1. **Add the file to the staging area:**

bash

Copy code

git add hello.txt

1. **Verify the status again:**

bash

Copy code

git status

hello.txt should now be listed under "Changes to be committed."

### 5. Commit the File

1. **Commit the file to the repository:**

bash

Copy code

git commit -m "Initial commit with hello.txt"

### 6. Make Changes and Commit Again

1. **Modify the hello.txt file:**

bash

Copy code

echo "This is an update to the file." >> hello.txt

1. **Check the status:**

bash

Copy code

git status

1. **View the changes:**

bash

Copy code

git diff

1. **Stage and commit the changes:**

bash

Copy code

git add hello.txt

git commit -m "Updated hello.txt with additional content"

### 7. View Commit History

bash

Copy code

git log

This shows a history of commits, including commit hashes, author names, dates, and messages.

### 8. Create and Switch to a New Branch

1. **Create a branch:**

bash

Copy code

git branch feature-branch

1. **Switch to the new branch:**

bash

Copy code

git checkout feature-branch

1. **Create a new file on this branch:**

bash

Copy code

echo "Feature branch content" > feature.txt

git add feature.txt

git commit -m "Added feature.txt on feature branch"

### 9. Merge Branches

1. **Switch back to main or master branch:**

bash

Copy code

git checkout main

1. **Merge the feature branch:**

bash

Copy code

git merge feature-branch

### 10. Delete the Merged Branch

bash

Copy code

git branch -d feature-branch

### 11. Push to a Remote Repository (Optional)

1. **Add a remote repository:**

bash

Copy code

git remote add origin https://github.com/yourusername/your-repo.git

1. **Push the commits:**

bash

Copy code

git push -u origin main

#### Example of Pull Up Method:

* **Before**:

java

Copy code

public class Student extends Person {

public void printInfo() {

System.out.println("Name: John");

}

}

public class Teacher extends Person {

public void printInfo() {

System.out.println("Name: Jane");

}

}

* **After**:

java

Copy code

public class Person {

public void printInfo() {

System.out.println("Name: [Generic Name]");

}

}

public class Student extends Person {

// Additional specific methods for Student, if needed

}

public class Teacher extends Person {

// Additional specific methods for Teacher, if needed

}

### 2. Pull Down Method/Field

**Pull Down** is used when a method or field in a superclass is only relevant for certain subclasses and needs to be moved down to those subclasses.

#### Steps for Pull Down:

1. **Identify Specific Code**:
   * Locate the methods or fields in the superclass that are only needed in certain subclasses.
2. **Move Method/Field**:
   * Copy or cut the method/field from the superclass and paste it into the relevant subclasses.
3. **Remove from Superclass**:
   * Delete the method/field from the superclass after confirming it has been moved correctly.
4. **Test the Code**:
   * Ensure that the new structure works as expected and that the subclasses behave correctly.

#### Example of Pull Down Method:

* **Before**:

java

Copy code

public class Person {

public void teach() {

System.out.println("Teaching a class");

}

}

public class Student extends Person {

// Doesn't need the `teach` method

}

public class Teacher extends Person {

// Needs the `teach` method

}

* **After**:

java

Copy code

public class Person {

// Generic properties/methods common to all

}

public class Student extends Person {

// Specific methods for Student

}

public class Teacher extends Person {

public void teach() {

System.out.println("Teaching a class");

}

}

#### **Refactoring Goal: to Rename:**

* Rename OldClassName to NewClassName.

#### Steps:

1. Right-click on OldClassName.
2. Choose Refactor > Rename.
3. Type NewClassName and confirm.