**1. Introduction to Reading External Files in Selenium**

In Selenium, storing and retrieving data from external files is a common practice for maintaining test configuration, test data, and other project settings. Two common types of files you will interact with are:

* **Configuration files** (usually in .properties format).
* **Excel files** (for managing large datasets for testing).

**2. Reading Configuration Files**

Configuration files (usually .properties) are used to store key-value pairs for test configurations, such as browser types, URLs, timeouts, and other settings. They provide an easy way to manage and change test settings without modifying the actual code.

**Steps to Read a Configuration File:**

1. Create a .properties file.
2. Load the file using FileInputStream.
3. Read the key-value pairs using the Properties class.

**Example of a Configuration File (config.properties):**

properties

browser=chrome

url=https://www.google.com

timeout=30

**Code to Read Configuration File:**

java

import java.io.FileInputStream;

import java.io.IOException;

import java.util.Properties;

public class ConfigReader {

public static void main(String[] args) throws IOException {

// Create a Properties object

Properties prop = new Properties();

// Load the config.properties file

FileInputStream fis = new FileInputStream("config.properties");

prop.load(fis);

// Reading values from the config file

String browser = prop.getProperty("browser");

String url = prop.getProperty("url");

String timeout = prop.getProperty("timeout");

// Output the read values

System.out.println("Browser: " + browser);

System.out.println("URL: " + url);

System.out.println("Timeout: " + timeout);

}

}

**Explanation:**

* FileInputStream is used to load the file.
* Properties class is used to read key-value pairs from the .properties file.

**3. Reading Data from Excel Files**

For managing large datasets, Excel is widely used. Selenium doesn’t have direct support for Excel, but we can use the Apache POI library to interact with Excel files.

**Steps to Read Excel Files:**

1. Add the Apache POI library to your project.
2. Use FileInputStream to load the Excel file.
3. Use classes like XSSFWorkbook, XSSFSheet, XSSFRow, and XSSFCell to read data.

**Maven Dependency for Apache POI:**

xml

<dependency>

<groupId>org.apache.poi</groupId>

<artifactId>poi-ooxml</artifactId>

<version>5.0.0</version>

</dependency>

**Example Excel Sheet (TestData.xlsx):**

| **Test Case** | **URL** | **Username** | **Password** |
| --- | --- | --- | --- |
| TC\_01 | <https://example.com> | user1 | pass1 |
| TC\_02 | <https://another.com> | user2 | pass2 |

**Code to Read Data from Excel:**

java

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.xssf.usermodel.XSSFSheet;

import org.apache.poi.xssf.usermodel.XSSFWorkbook;

**public class ExcelReader {**

**public static void main(String[] args) throws IOException {**

**// Load the Excel file**

**FileInputStream fis = new FileInputStream("TestData.xlsx");**

**XSSFWorkbook workbook = new XSSFWorkbook(fis);**

**XSSFSheet sheet = workbook.getSheetAt(0);**

**// Reading data from the first row and first cell (A1)**

**Row row = sheet.getRow(0);**

**Cell cell = row.getCell(0);**

**// Output the cell value**

**System.out.println("First cell value: " + cell.getStringCellValue());**

**// Closing the workbook and file stream**

**workbook.close();**

**fis.close();**

**}**

**}**

**Explanation:**

* XSSFWorkbook is used to open the Excel workbook.
* XSSFSheet accesses the sheet within the Excel file.
* XSSFRow and XSSFCell are used to access individual rows and cells.

**4. Example: Reading Both Configuration and Excel Files in Selenium**

Here is a practical example of how you can use both configuration and Excel files together in a Selenium test.

**Code Example:**

java

import java.io.FileInputStream;

import java.io.IOException;

import java.util.Properties;

import org.apache.poi.ss.usermodel.Row;

import org.apache.poi.ss.usermodel.Sheet;

import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

public class ConfigAndExcelTest {

public static void main(String[] args) throws IOException {

// Load configuration properties

Properties prop = new Properties();

FileInputStream fisConfig = new FileInputStream("config.properties");

prop.load(fisConfig);

String browser = prop.getProperty("browser");

String url = prop.getProperty("url");

// Load test data from Excel

FileInputStream fisExcel = new FileInputStream("TestData.xlsx");

XSSFWorkbook workbook = new XSSFWorkbook(fisExcel);

Sheet sheet = workbook.getSheetAt(0);

// Reading data from Excel

Row row = sheet.getRow(1);

String testURL = row.getCell(1).getStringCellValue();

String username = row.getCell(2).getStringCellValue();

String password = row.getCell(3).getStringCellValue();

// Initialize WebDriver (example with ChromeDriver)

WebDriver driver = new ChromeDriver();

// Use URL from configuration file

driver.get(url);

// You can use the test data (username, password) for logging in, etc.

System.out.println("Test URL: " + testURL);

System.out.println("Username: " + username);

System.out.println("Password: " + password);

// Close resources

workbook.close();

fisExcel.close();

fisConfig.close();

driver.quit();

}

}

**5. Best Practices:**

* **Configuration File:** Use .properties files for any setting that you expect might change (like browser types, URLs, timeouts, etc.).
* **Excel File:** Use Excel for larger sets of data, such as multiple sets of test inputs, test cases, and expected outputs.

**6. Questions and Discussion:**

* When should you use a configuration file vs an Excel file?
* How can you handle large test datasets more efficiently using external files?

**7. Conclusion:**

* Reading configuration files helps to manage project settings in a centralized manner.
* Reading Excel files allows you to manage large sets of test data effectively.
* This combination provides a flexible and scalable solution for data-driven testing in Selenium.

**Homework:**

* Create a .properties file for your current project and use it to store browser types, base URLs, and timeouts.
* Implement an Excel-based data-driven testing approach where you can load multiple test cases from an Excel file.