Why Data Driven Tests important?

There are two main benefits to this:

- They reduce the cost of adding new tests and changing them when your business rules change. This is done by creating parameters for different scenarios, using data sets that the same code can run against.
- They help to identify what data is most important for tested behavior. By separating first-class scenario data into parameters, it becomes clear what matters most to the test. This makes it easy to remember how something works when developers need to change it.

Data Driven Testing

Data Driven Testing is a software testing method in which test data is stored in table or spreadsheet format. Data driven testing allows testers to input a single test script that can execute tests for all test data from a table and expect the test output in the same table. It is also called table-driven testing or parameterized testing.

How to work on Data Driven Framework in Selenium Using Apache POI?

Data Driven Framework is one of the popular Automation Testing Framework in the current market. Data Driven automated testing is a method in which the test data set is created in the excel sheet, and is then imported into automation testing tools to feed to the software under test.

Selenium Webdriver is a great tool to automate web-based applications. But it does not support read and write operations on excel files.
Therefore, we use third party APIs like Apache POI.

What is Apache POI?

Apache POI (Poor Obfuscation Implementation) is an API written in Java to support read and write operations – modifying office files. This is the most common API used for Selenium data driven tests.

Data Driven Framework in Selenium using Apache POI

There are several ways to implement a Data Driven Framework, and each differs in the effort required to develop the framework and maintenance.
Developing Data Driven framework in Selenium using POI helps reduce maintenance, improve test coverage thus providing a good return on investment.

The Apache POI library provides two implementations for reading excel files:

- **HSSF (Horrible SpreadSheet Format) Implementation:** It denotes an API that is working with Excel 2003 or earlier versions.
- **XSSF (XML SpreadSheet Format) Implementation:** It denotes an API that is working with Excel 2007 or later versions.

Interfaces and Classes in Apache POI

Interfaces

- **Workbook:** It represents an **Excel Workbook**. It is an interface implement by **HSSFWorkbook** and **XSSFWorkbook**.
- **Sheet:** It is an interface that represents an **Excel worksheet**. A sheet is a central structure of a workbook, which represents a grid of cells. The Sheet interface extends **java.lang.Iterable**.
- **Row:** It is also an interface that represents the **row** of the spreadsheet. The Row interface extends **java.lang.Iterable**. There are two concrete classes: **HSSFRow** and **XSSFRow**.
- **Cell:** It is an interface. It is a high-level representation of a **cell** in a row of the spreadsheet. **HSSFCell** and **XSSFCell** implement Cell interface.

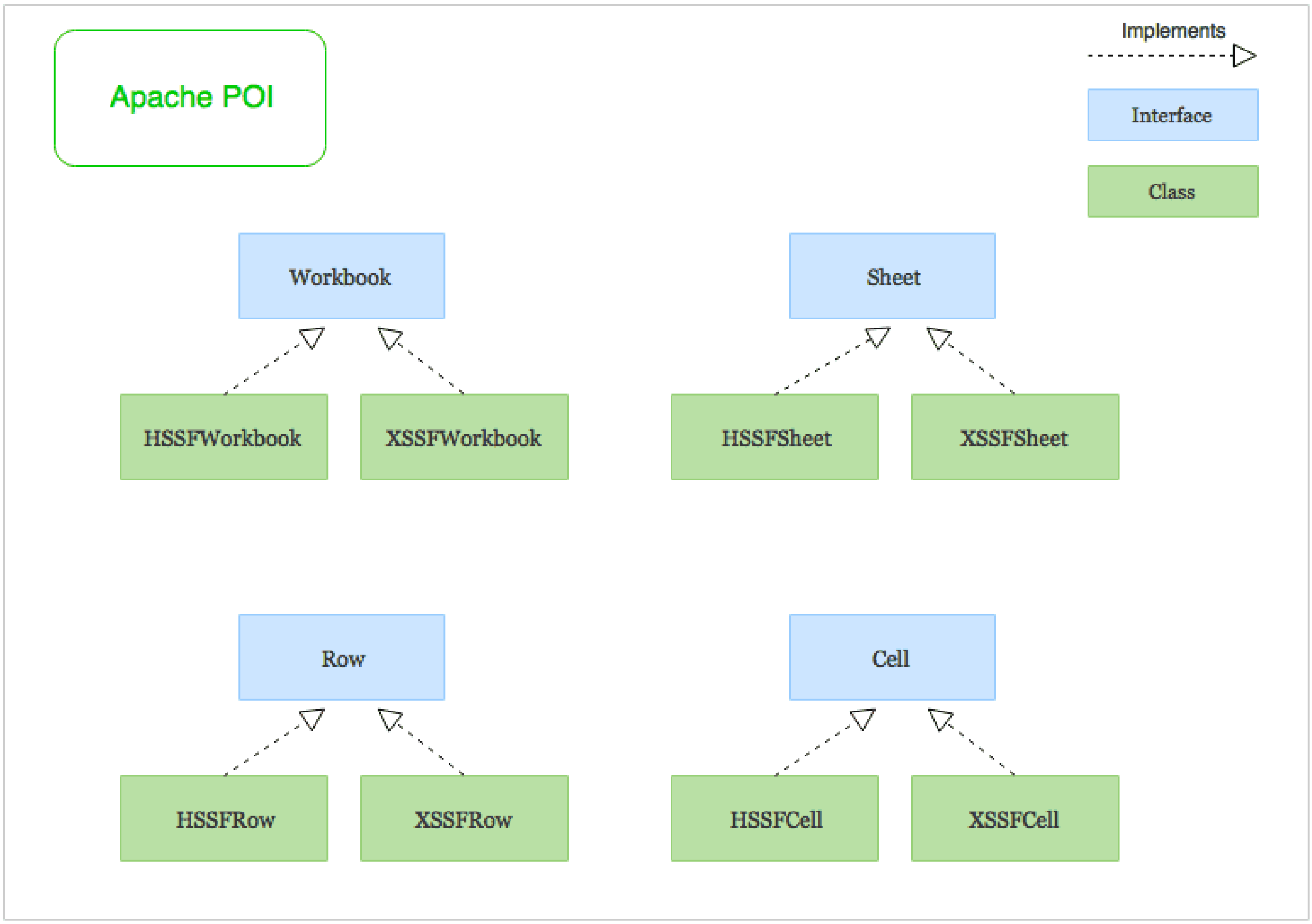
Classes

XLS Classes

- **HSSFWorkbook:** It is a class representing the XLS file.
- **HSSFSheet:** It is a class representing the sheet in an XLS file.
- **HSSFRow:** It is a class representing a row in the sheet of XLS file.
- **HSSFCell:** It is a class representing a cell in a row of XLS file.

XLSX Classes

- **XSSFWorkbook:** It is a class representing the XLSX file.
- **XSSFSheet:** It is a class representing the sheet in an XLSX file.
- **XSSFRow:** It is a class representing a row in the sheet of XLSX file.
- **XSSFCell:** It is a class representing a cell in a row of XLSX file.



Apache POI – Dependencies

If you are working on a maven project, you can include the POI dependency in `pom.xml` file using this:

pom.xml

```
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi</artifactId>
    <version>3.9</version>
</dependency>
```

If you are not using maven, then you can download maven jar files from POI download page. Include following jar files to run the sample code given in this tutorial.

- poi-3.9-20121203.jar
- poi-ooxml-3.9-20121203.jar
- poi-ooxml-schemas-3.9-20121203.jar

Apache POI – Write an excel file

I am taking this example first so that we can reuse the excel sheet created by this code to read back in next example.

Writing excel using POI is very simple and involve following steps:

1. Create a workbook
2. Create a sheet in workbook
3. Create a row in sheet
4. Add cells in sheet
5. Repeat step 3 and 4 to write more data

Apache POI – Read an excel file

Reading an excel file using POI is also very simple if we divide this in steps.

1. Create workbook instance from excel sheet
2. Get to the desired sheet
3. Increment row number

4. iterate over all cells in a row
5. repeat step 3 and 4 until all data is read

Lets see all above steps in code. I am writing the code to read the excel file created in above example. It will read all the column names and the values in it – cell by cell.

Java program to read excel file using apache POI library.

```
package com.demo.poi;

//import statements

public class ReadExcelDemo

{

    public static void main(String[] args)

    {

        try

        {

            FileInputStream file = new FileInputStream("testData_demo.xlsx");


            //Create Workbook instance holding reference to .xlsx file

            XSSFWorkbook workbook = new XSSFWorkbook(file);


            //Get first/desired sheet from the workbook

            XSSFSheet sheet = workbook.getSheetAt(0);


            //Iterate through each rows one by one

            Iterator<Row> rowIterator = sheet.iterator();

            while (rowIterator.hasNext())

            {

                Row row = rowIterator.next();

                //For each row, iterate through all the columns

                Iterator<Cell> cellIterator = row.cellIterator();


                while (cellIterator.hasNext())

                {

                    Cell cell = cellIterator.next();

                    //Check the cell type and format accordingly
                    switch (cell.getCellType())
                    {
                        case Cell.CELL_TYPE_NUMERIC:
                            System.out.print(cell.getNumericCellValue() + "t");
                            break;
                        case Cell.CELL_TYPE_STRING:
                            System.out.print(cell.getStringCellValue() + "t");
                            break;
                    }
                }
                System.out.println("");
            }
            file.close();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

Output:

```
 ID      NAME        LASTNAME
1.0   Amit        Shukla
2.0   Lokesh      Gupta
3.0   John        Adwards
4.0   Brian       Schultz
```

Apache POI – Write an excel file

I am taking this example first so that we can reuse the excel sheet created by this code to read back in next example.

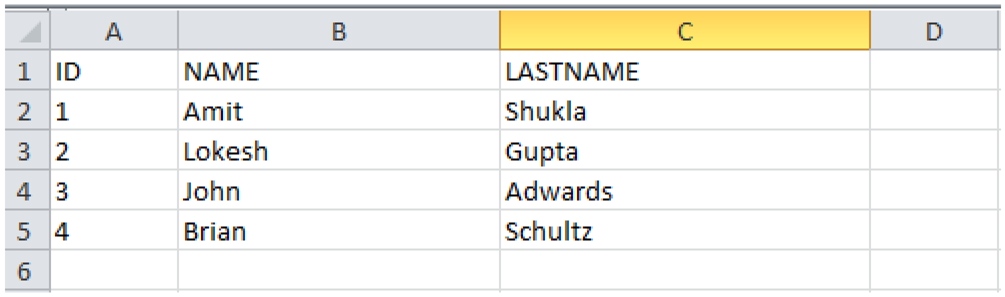
Writing excel using POI is very simple and involve following steps:

1. Create a workbook
2. Create a sheet in workbook
3. Create a row in sheet
4. Add cells in sheet
5. Repeat step 3 and 4 to write more data

It seems very simple, right? Lets have a look at the code doing these steps.

Java program to write excel file using apache POI library.

```
package com.demo.poi;

//import statements

public class WriteExcelDemo

{

    public static void main(String[] args)

    {

        //Blank workbook

        XSSFWorkbook workbook = new XSSFWorkbook();


        //Create a blank sheet

        XSSFSheet sheet = workbook.createSheet("Employee Data");


        //This data needs to be written (Object[])

        Map<String, Object[]> data = new TreeMap<String, Object[]>();

        data.put("1", new Object[] {"ID", "NAME", "LASTNAME"});

        data.put("2", new Object[] {1, "Amit", "Shukla"});

        data.put("3", new Object[] {2, "Lokesh", "Gupta"});

        data.put("4", new Object[] {3, "John", "Adwards"});

        data.put("5", new Object[] {4, "Brian", "Schultz"});
        //Iterate over data and write to sheet
        Set<String> keyset = data.keySet();
        int rownum = 0;
        for (String key : keyset)
        {
            Row row = sheet.createRow(rownum++);
            Object [] objArr = data.get(key);
            int cellnum = 0;
            for (Object obj : objArr)
            {
               Cell cell = row.createCell(cellnum++);
               if(obj instanceof String)
                    cell.setCellValue((String)obj);
                else if(obj instanceof Integer)
                    cell.setCellValue((Integer)obj);
            }
        }
        try
        {
            //Write the workbook in file system
            FileOutputStream out = new FileOutputStream("testData_demo.xlsx");
            workbook.write(out);
            out.close();
            System.out.println("testData_demo.xlsx written successfully on disk.");
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```



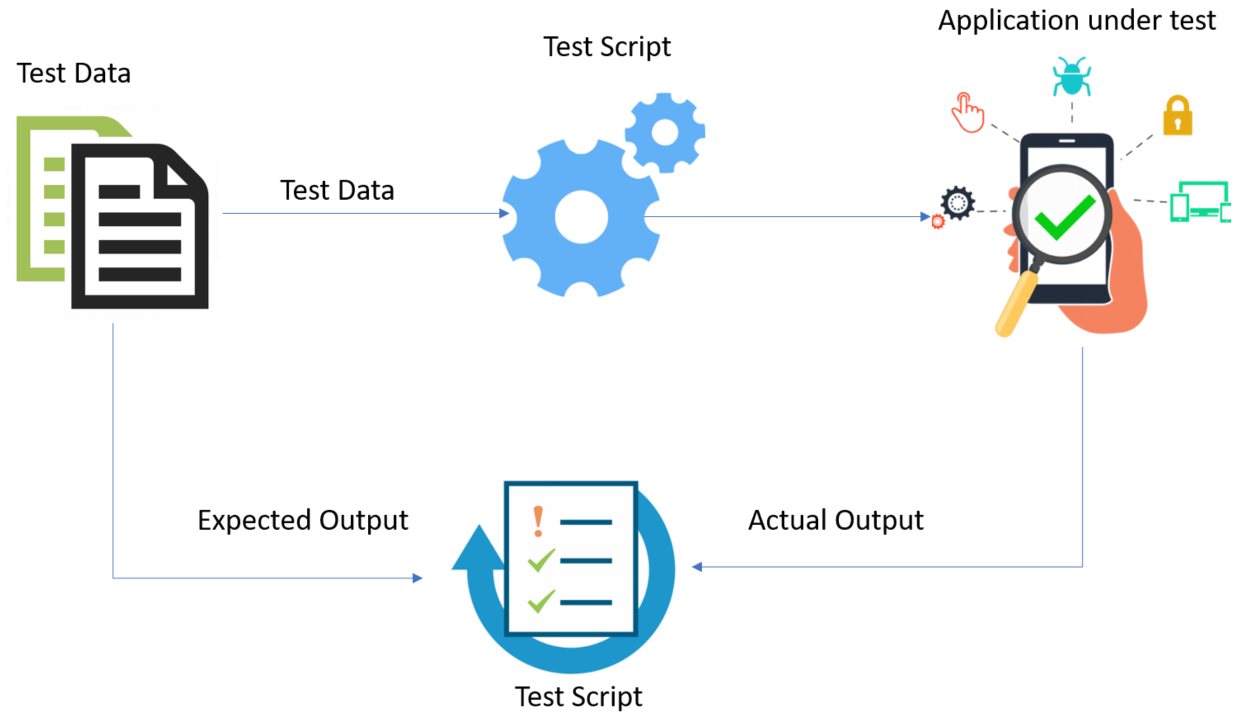
	A	B	C	D
1	ID	NAME	LASTNAME	
2	1	Amit	Shukla	
3	2	Lokesh	Gupta	
4	3	John	Adwards	
5	4	Brian	Schultz	
6				

Apache POI imports data from the excel sheet and uses it to log into our application. Now that we saw how to read data from the excel sheet, let's look at how to write to the sheet.

```
package automationFramework;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.concurrent.TimeUnit;

import org.apache.poi.hssf.usermodel.HSSFCell;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.apache.poi.ss.usermodel.Cell;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.WebDriverWait;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

/**
 * @author Admin
 *
 */
public class ReadWriteExcel
{
    WebDriver driver;
    WebDriverWait wait;
    HSSFWorkbook workbook;
    HSSFSheet sheet;
    HSSFCell cell;

 @BeforeTest
 public void TestSetup()
{
    // Set the path of the Firefox driver.
    WebDriverManager.chromedriver().setup();
    driver = new FirefoxDriver();

    // Enter url.
    driver.get("http://www.linkedin.com/");
    driver.manage().window().maximize();

    wait = new WebDriverWait(driver,30);
    driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
}

 @Test
 public void ReadData() throws IOException
 {
     // Import excel sheet.
     File src=new File("C:\\Users\\Admin\\Desktop\\TestData.xls");

     // Load the file.
     FileInputStream finput = new FileInputStream(src);

     // Load he workbook.
    workbook = new HSSFWorkbook(finput);

     // Load the sheet in which data is stored.
     sheet= workbook.getSheetAt(0);

     for(int i=1; i&amp;lt;=sheet.getLastRowNum(); i++)
     {
         // Import data for Email.
         cell = sheet.getRow(i).getCell(1);
         cell.setCellType(Cell.CELL_TYPE_STRING);
         driver.findElement(By.id("login-email")).sendKeys(cell.getStringCellValue());

         // Import data for password.
         cell = sheet.getRow(i).getCell(2);
         cell.setCellType(Cell.CELL_TYPE_STRING);
         driver.findElement(By.id("login-password")).sendKeys(cell.getStringCellValue());

         // Write data in the excel.
       FileOutputStream foutput=new FileOutputStream(src);

        // Specify the message needs to be written.
        String message = "Data Imported Successfully.";

        // Create cell where data needs to be written.
        sheet.getRow(i).createCell(3).setCellValue(message);

        // Specify the file in which data needs to be written.
        FileOutputStream fileOutput = new FileOutputStream(src);

        // finally write content
        workbook.write(fileOutput);

         // close the file
        fileOutput.close();

     }
 }
}
```

What is Data Driven Testing Framework in Selenium?

Data Driven framework is used to drive test cases and suites from an external data feed. The data feed can be data sheets like xls, xlsx, and csv files.



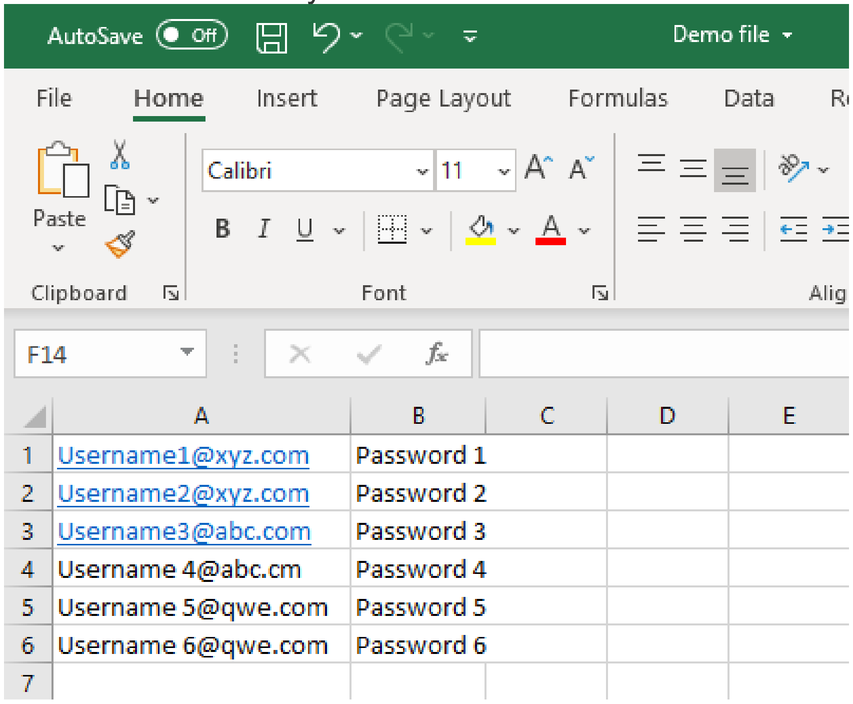
A Data Driven Framework in Selenium is a technique of separating the “data set” from the actual “test case” (code). Since the test case is separated from the data set, one can easily modify the test case of a particular functionality without making changes to the code.
For example, if one has to modify the code for login functionality, they can modify just the login functionality instead of having to modify any other feature dependent on the same code.
One can easily increase the number of test parameters by adding more username and password fields to the excel file (or other sources).

Data Driven Testing Example

This example will demonstrate how to read the data from excel files and perform data driven testing using Selenium. WebDriver does not directly support data reading of excel files. Therefore, one needs to use a plugin such as **Apache POI** for reading/writing on any Microsoft office document.

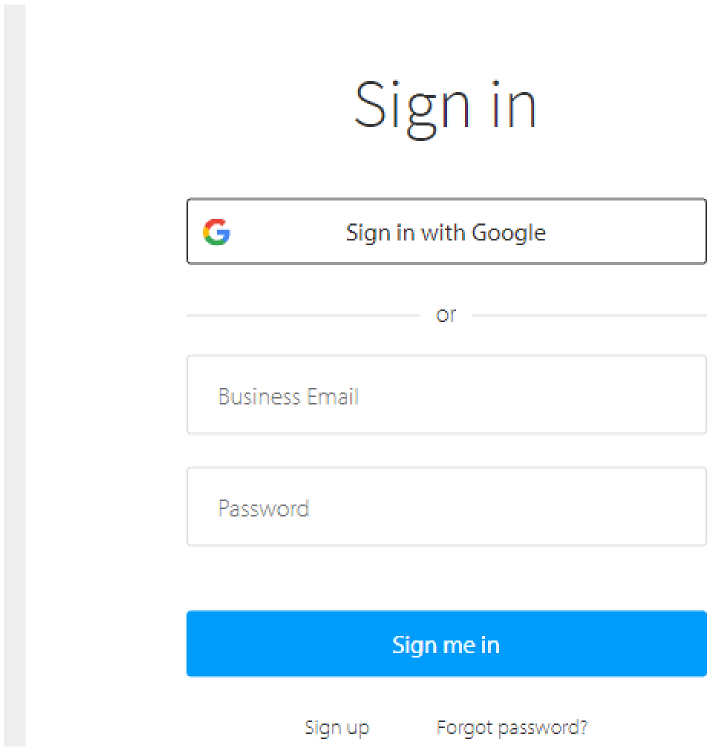
- To download Apache POI Jar files click here. Download the zip file or tar file as per requirement and place them along with the set of Selenium JARs and configure your build path.

Now let's understand how to write the first test case. An excel file to read the data from the sheet. The user has entered different combinations of username and password in the sheet.
The task here is to enter all the combinations of username and passwords into the login field in order to test the functionality. Let's see how to do that.



	A	B	C	D	E
1	Username1@xyz.com	Password 1			
2	Username2@xyz.com	Password 2			
3	Username3@abc.com	Password 3			
4	Username 4@abc.cm	Password 4			
5	Username 5@qwe.com	Password 5			
6	Username 6@qwe.com	Password 6			
7					

Here, the target is to enter all these combinations of username and password into the **Browserstack Sign in** page as shown below.



Let’s write a code snippet to read the data files.

Step 1: Go to the Eclipse IDE and create a project. Add all the dependencies for TestNG, Selenium and Apache POI.

Step 2: Create a class file to write the functionality.

```
import org.openqa.selenium.By;

import org.testng.Assert;

import org.testng.annotations.AfterMethod;

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

public class ExcelExample{

@Test(dataProvider="testdata")

public void demoClass(String username, String password) throws InterruptedException {

System.setProperty("webdriver.chrome.driver", "Path of Chrome Driver");

Webdriver driver = new ChromeDriver();

driver.get("<a href="https://www.browserstack.com/users/sign_in</a>");

driver.findElement(By.name("user[login]")).sendKeys(username);

driver.findElement(By.name("user[password]")).sendKeys(password);

driver.findElement(By.name("commit")).click();

Thread.sleep(5000);

Assert.assertTrue(driver.getTitle().matches("BrowserStack Login | Sign Into The Best Mobile &
Browser Testing Tool"), "Invalid credentials");

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

}

@AfterMethod

void ProgramTermination() {

driver.quit();

}
@DataProvider(name="testdata")

public Object[][] testDataExample(){

ReadExcelFile configuration = new ReadExcelFile("Path_of_Your_Excel_File");

int rows = configuration.getRowCount(0);

Object[][]signin_credentials = new Object[rows][2];



for(int i=0;i<rows;i++)

{

signin_credentials[i][0] = config.getData(0, i, 0);

signin_credentials[i][1] = config.getData(0, i, 1);

}

return signin_credentials;

}

}
```

In the above code, there is a “TestDataExample() method” in which the user has created an object instance of another class named “ReadExcelFile”. The user has mentioned the path to the excel file. The user has further defined a *for loop* to retrieve the text from the excel workbook. But to fetch the data from the excel file, one needs to write a class file for the same.
Try Running Selenium Tests on Cloud for Free

```
import java.io.File;

import java.io.FileInputStream;

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

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

public class ReadExcelFile{

XSSFWorkbook work_book;

XSSFSheet sheet;

public ReadExcelFile(String excelfilePath) {

try {
```

```
File s = new File(excelfilePath);

FileInputStream stream = new FileInputStream(s);

work_book = new XSSFWorkbook(stream);

}

catch(Exception e) {

System.out.println(e.getMessage());

}

}

public String getData(int sheetnumber, int row, int column){

sheet = work_book.getSheetAt(sheetnumber);

String data = sheet.getRow(row).getCell(column).getStringCellValue();

return data;

}

public int getRowCount(int sheetIndex){

int row = work_book.getSheetAt(sheetIndex).getLastRowNum();

row = row + 1;

return row;

}
```

In the code above, the user has used Apache POI libraries to fetch the data from the excel file. Next, it will point to the data present in the excel file and then enter the relevant username and password to the sign in page.
Note: The same thing can be done using a Data provider in TestNG. But to fetch the data from the Excel sheet, the user needs Apache POI jar files.
Note: Please enter one valid credential to test.

Advantages of Data Driven Testing Framework

1. Allows testing of the application with multiple sets of data values during regression testing

2. Separates the test case data from the executable test script
3. Allows reusing of Actions and Functions in different tests
4. Generates test data automatically. This is helpful when large volumes of random test data are necessary
5. Results in the creation of extensive code that is flexible and easy to maintain
6. Lets developers and testers separate the logic of their test cases/scripts from the test data
7. Allows execution of test cases several times which helps to reduce test cases and scripts
8. It does not let changes in test scripts affect the test data.

By incorporating data-driven testing using Selenium, testers can refine their test cases for more efficient execution. This shortens timelines, makes their lives easier and results in more thoroughly tested and better quality software.