

Data driven testing

Testing the feature with multiple set of data is called as **data driven testing**. Or

Data-driven testing (DDT) is taking a test, parameterizing it and then running that test with varying data.

Note:

- This allows you to run the same test case with many varying inputs, therefore increasing coverage from a single test. In addition to increasing test coverage, data driven testing allows the ability to build both positive and negative test cases into a single test.
- Data-driven testing allows you to test the form with a different set of input values to be sure that the application works as expected.
- It is convenient to keep data for automated tests in special storages that support sequential access to a set of data, for example, Excel sheets, database tables, arrays, and so on.
- Often data is stored either in a text file and are separated by commas or in Excel files and are presented as a table. If you need to add more data, you simply modify the file either in any text editor or in Microsoft Excel (in case of hard-coded values, you should modify both data and code).

Selenium data driven testing using webdriver

It is the responsibility of test developer to create and script data driven infrastructure so that the environment is capable of supplying different set of test data repeatedly and workflows that need to be tested against a particular application.

Practical scenario:

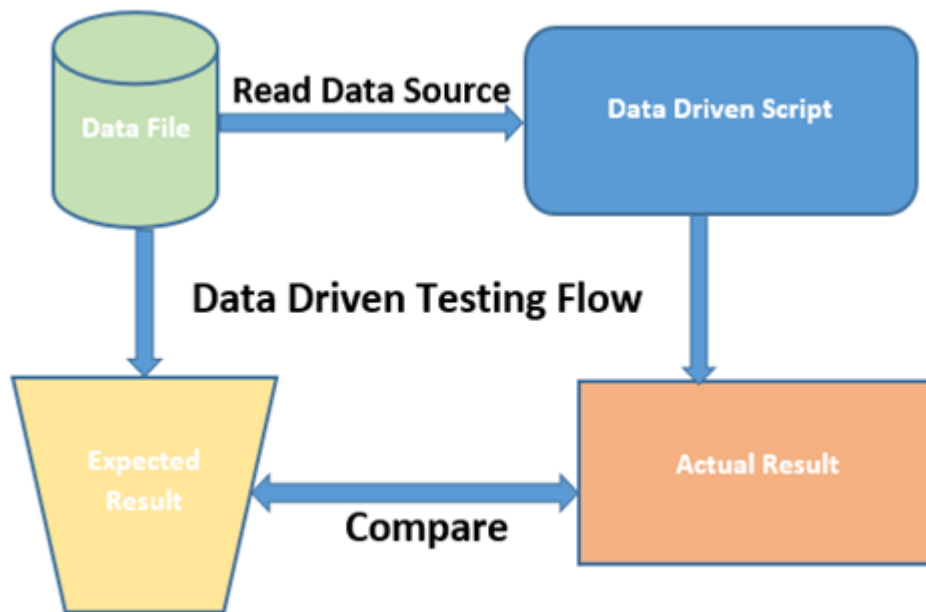
- Let's suppose we have a scenario to test the login functionality using various set of test data. So as per automation testing standards and data driven approach used in automation testing, automation engineer will write a generic script to test the application with varying data values.
- Automation engineer will create test scripts against this scenario:
 - Using valid credentials, test sign in functionality
 - Using username and blank password to test sign in functionality
 - Using blank username and password to test sign in functionality
 - And finally using no username and no password means send both fields blank to check the behaviour of the sign in functionality.

Selenium cannot read the data from external resources. In order to read and write the data into external resources, selenium uses the POI modules to read the data from excel and properties class to read the data from properties file.

Data-driven test includes the following operations performed in a loop:

- Retrieving input data from storage.
- Entering data in an application form.
- Verifying the results.
- Continuing with the next set of input data.

Data Driven Testing can be understood by the following diagram:



Reading the data from Property Files

What is a property file?

- **.properties** files are mainly used in Java programs to maintain **project configuration data, database config** or **project settings** etc.
- Each parameter in properties file are stored as a pair of strings, in **key-value** pair format, where each key is on one line.
- You can easily read properties from some file using object of type **Properties**. This is a utility provided by Java itself.

java.util.Properties;

- In Selenium .properties files are mainly used to store GUI locators / elements, and also global fields like database configuration details.

Advantages of Property file in Selenium

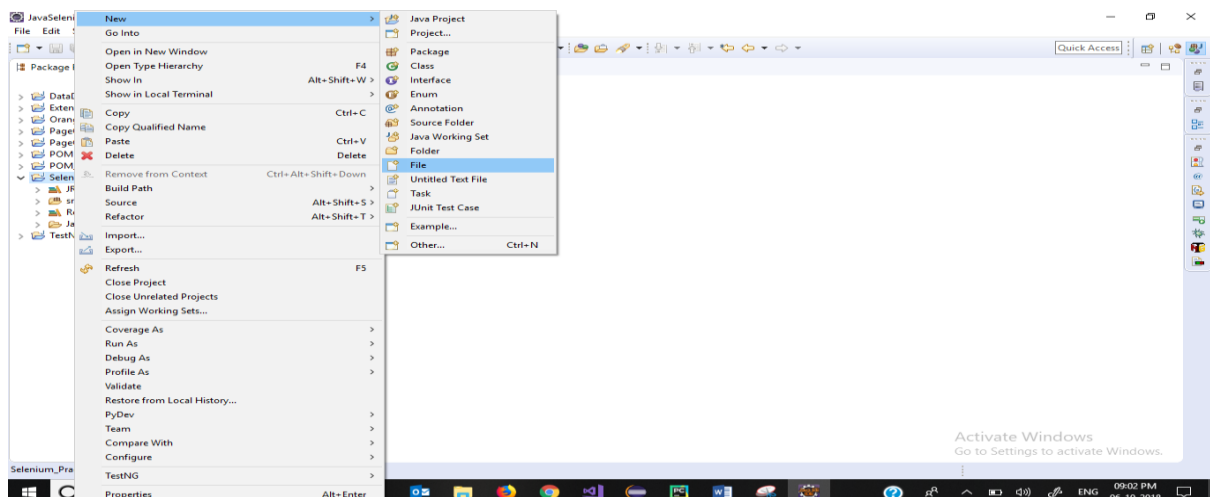
- If any information is changed from the properties file, you don't need to recompile the java class.
- In other words, the advantage of using properties file is we can configure things which are prone to change over a period of time without need of changing anything in code.

For E.g. We keep application URL in property file, so in case you want to run test from on other test environment, just change the URL in property file and that's it. You do not require to build the whole project again.

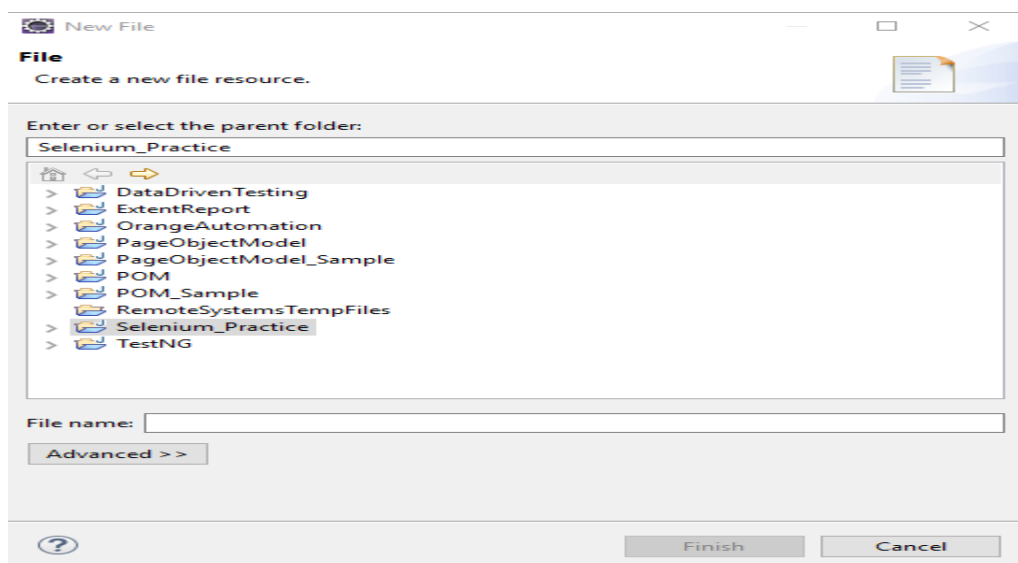
How to write/read from property files

Step 1: Create a property file.

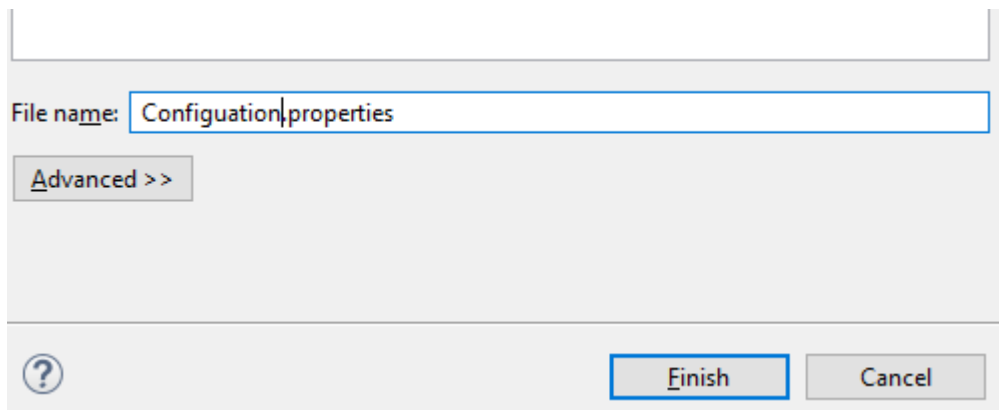
Right click on the project -> new -> File



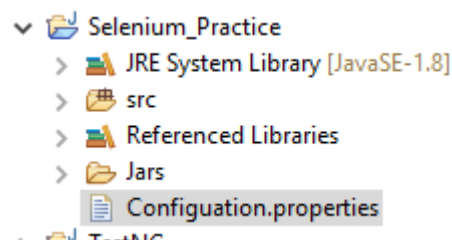
Step 2: After selecting the “File” option. We will get the below window.



Step 3: The only thing we need to do is to give the file name and give the extension as **.properties**. In our case we name it as **Configuration.properties**.



Step 4: Click Finish.



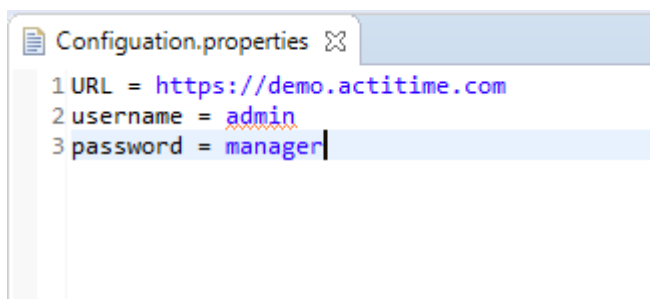
Note: Configuration.properties file added to your project.

Step 5: Store data onto properties file.

Note:

- Data is stored in properties file in the form of key-value pairs, with the key being unique across the file.
- Remember white space between the property name and property value is always ignored.
- Don't declare the key and value within double ("") quotation in properties file.

Example:



How to write the data to properties file.

Step 1: Create the object File output stream and pass the path of the property file.

```
output = new FileOutputStream("./TestData/config.properties");  
// set the properties value
```

Pramod K S

Step 2: Create the object of property file.

```
Properties prop = new Properties();
```

Step 3: Call the method “**setProperty**” method of properties class. This method takes two arguments.

Argument 1: Key.

Argument 2: Value.

Note: Key should be unique, value should be the value you want to write to properties file.

```
// set the properties value
prop.setProperty("database", "localhost");
prop.setProperty("dbuser", "pramod");
prop.setProperty("dbpassword", "password");
```

Step 4: After writing the data, we need to save the file. In order to save, call “**store**” method. This method will save the properties file.

Step 5: Once after saving, we need to close the output stream using the “**close**” method of file output stream class.

Below is a program to write the data to properties file.

```
package com.properties;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.util.Properties;

public class SetProperties {
    public static void main(String[] args) {
        System.out.println("Program Starts.....");
        Properties prop = new Properties();
        OutputStream output = null;

        try {
            output = new FileOutputStream("../TestData/config.properties");
            // set the properties value
            prop.setProperty("database", "localhost");
            prop.setProperty("dbuser", "pramod");
            prop.setProperty("dbpassword", "password");
            // save properties to project root folder
            prop.store(output, "Storing the data to config file");

        } catch (IOException io) {
            io.printStackTrace();
        } finally {
            if (output != null) {
                try {
                    output.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            System.out.println("Program Ends.....");
        }
    }
}
```

How to read the data from property file.

- In order to read the data from the properties file. Please follow the below step

Step 1: Create an object of properties class.

```
Properties prop = new Properties();
```

Step 2: Create an object of file input stream and pass the path of the file.

```
input = new FileInputStream("./TestData/config.properties");  
// Read the file
```

Step 3: So now, in order to read, after creating the object above classes, load the file using the “load” method of properties class by passing the input stream class object as an argument.

```
// load a properties file  
prop.load(input);
```

Step 4: Call “getProperty” method and pass the key name to get the values from properties file.

```
// get the property value and print it out  
System.out.println(prop.getProperty("database"));  
System.out.println(prop.getProperty("dbuser"));  
System.out.println(prop.getProperty("dbpassword"));  
} // End of the program
```

Step 5: Close the input stream using “close” method of file input stream.

Below the program to read the date from properties file.

```

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;

public class ReadProperties {
    public static void main(String[] args) {
        Properties prop = new Properties();
        InputStream input = null;

        try {
            input = new FileInputStream("./TestData/config.properties");
            // load a properties file
            prop.load(input);
            // get the property value and print it out
            System.out.println(prop.getProperty("database"));
            System.out.println(prop.getProperty("dbuser"));
            System.out.println(prop.getProperty("dbpassword"));
        } catch (IOException ex) {
            ex.printStackTrace();
        } finally {
            if (input != null) {
                try {
                    input.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}

```

Excel Automation

- Excel files (spreadsheets) are widely used by people all over the world for various tasks related to organization, analysis, and storage of tabular data.
- Since excel files are so common, we automation engineers often encounter use-cases when we need to read data from an excel file or generate a report in excel format.
- Here, I'll show you how to read excel files in Java using a very simple yet powerful open source library called **Apache POI**.
- And also, you'll learn how to create and write to an excel file using **Apache POI**.

Let's get started!

Dependencies

First of all, we need to add the required dependencies for including Apache POI in our project. If you use maven, you need to add the following dependencies to your `pom.xml` file

```
<!-- https://mvnrepository.com/artifact/org.apache.poi/poi -->
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi</artifactId>
  <version>3.17</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.poi/poi-ooxml -->
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi-ooxml</artifactId>
  <version>3.17</version>
</dependency>
```

Or Else navigate to the following web site

<https://poi.apache.org/download.html#POI-4.0.0>

Scroll down to “Binary Distribution” and click on the “poi-bin-4.0.1-20181203.zip” file.

repository under Group ID "org.apache.poi" and Version "4.0.1".

Binary Distribution

- [poi-bin-4.0.1-20181203.tar.gz](#) (26.85 MB, [signature \(.asc\)](#), checksum: [SHA-256](#), [SHA-512](#))
- [poi-bin-4.0.1-20181203.zip](#) (36.28 MB, [signature \(.asc\)](#), checksum: [SHA-256](#), [SHA-512](#))

Source Distribution

- [poi-src-4.0.1-20181203.tar.gz](#) (93.03 MB, [signature \(.asc\)](#), checksum: [SHA-256](#), [SHA-512](#))
- [poi-src-4.0.1-20181203.zip](#) (97.23 MB, [signature \(.asc\)](#), checksum: [SHA-256](#), [SHA-512](#))

Nightly Builds

It will navigate to the following page



We suggest the following mirror site for your download:

<http://mirrors.estointernet.in/apache/poi/release/bin/poi-bin-4.0.1-20181203.zip>

Other mirror sites are suggested below.

It is essential that you [verify the integrity](#) of the downloaded file using the PGP signature (.asc file) or a hash (.md5 or .sha* file).

Please only use the backup mirrors to download KEYS, PGP and MD5 sigs/hashes or if no other mirrors are working.

HTTP

<http://mirrors.estointernet.in/apache/poi/release/bin/poi-bin-4.0.1-20181203.zip>

<http://mirrors.wuchna.com/apachemirror/poi/release/bin/poi-bin-4.0.1-20181203.zip>












Click on the highlighted “zip” file and download the zip.

Unzip the downloaded zip file.



And copy all the jar file and add to build path.

Name	Date modified	Type	Size
docs	26-11-2018 10:53 ...	File folder	
lib	26-11-2018 10:53 ...	File folder	
ooxml-lib	26-11-2018 10:53 ...	File folder	
LICENSE	26-11-2018 10:27 ...	File	30 KB
NOTICE	26-11-2018 10:27 ...	File	2 KB
poi-4.0.1	26-11-2018 10:53 ...	Executable Jar File	2,655 KB
poi-examples-4.0.1	26-11-2018 10:53 ...	Executable Jar File	415 KB
poi-excelant-4.0.1	26-11-2018 10:53 ...	Executable Jar File	31 KB
poi-ooxml-4.0.1	26-11-2018 10:53 ...	Executable Jar File	1,725 KB
poi-ooxml-schemas-4.0.1	26-11-2018 10:53 ...	Executable Jar File	7,589 KB
poi-scratchpad-4.0.1	26-11-2018 10:53 ...	Executable Jar File	1,352 KB

PC > Downloads > poi-bin-4.0.1-20181203 > poi-4.0.1 > lib

Name	Date modified	Type	Size
 activation-1.1.1	26-11-2018 10:42 ...	Executable Jar File	68 KB
 commons-codec-1.11	26-11-2018 10:42 ...	Executable Jar File	328 KB
 commons-collections4-4.2	26-11-2018 10:42 ...	Executable Jar File	736 KB
 commons-compress-1.18	26-11-2018 10:43 ...	Executable Jar File	578 KB
 commons-logging-1.2	26-11-2018 10:42 ...	Executable Jar File	61 KB
 commons-math3-3.6.1	26-11-2018 10:42 ...	Executable Jar File	2,162 KB
 jaxb-api-2.3.0	26-11-2018 10:42 ...	Executable Jar File	123 KB
 jaxb-core-2.3.0.1	26-11-2018 10:42 ...	Executable Jar File	249 KB
 jaxb-impl-2.3.0.1	26-11-2018 10:42 ...	Executable Jar File	939 KB
 junit-4.12	26-11-2018 10:42 ...	Executable Jar File	308 KB
 log4j-1.2.17	26-11-2018 10:42 ...	Executable Jar File	479 KB

This PC > Downloads > poi-bin-4.0.1-20181203 > poi-4.0.1 > ooxml-lib

Name	Date modified	Type	Size
 curvesapi-1.05	26-11-2018 10:43 ...	Executable Jar File	110 KB
 xmlbeans-3.0.2	26-11-2018 10:43 ...	Executable Jar File	2,513 KB

Add all the above “Jars” to “Build Path”

Note: In order to read the test data from excel. First create a excel file with the test data.

Example:

A	B	C	D
UserName	Password	Expected_result	Result
admin	manager	actiTIME - Enter Time-Track	
admin	mnagher	actiTIME - Login	
user	manager	actiTIME - Login	
admin	manager	actiTIME - Login	
user2	user23	actiTIME - Login	

Apache POI excel library revolves around following four key interfaces

1. **Workbook:** A workbook is the high-level representation of a Spreadsheet.
2. **Sheet:** high level representation of an Excel worksheet.
3. **Row:** As the name suggests, it represents a row in the spreadsheet.

Pramod K S

4. **Cell**: high level representation of a cell in a row.

Step 1: Create a file input stream and pass the path of the excel file as an argument.

```
String FilePath = "./TestData/Data.xlsx";  
FileInputStream fis = new FileInputStream(FilePath);
```

Step 2: Call **create** method from **"workbookfactory"** class, and pass the object of file input stream as an argument. Which return object of workbook.

```
Workbook wb = WorkbookFactory.create(fis);
```

Step 3: Call **"GetSheet"** method from workbook by passing the **"Sheet Name"** as an argument. Which returns the Sheet.

```
Sheet s = wb.getSheet("Sheet1");
```

Step 4: Call **"GetRow"** method from sheet, and pass the row number, which will return the row.

```
Row r = s.getRow(0);
```

Step 5: Call **"getCell"** method from Row, and pass the cell number, which will return the cell.

```
Cell c = r.getCell(0);
```

Step 6: Now, in order to get the value from the sheet, you can use the following methods.

- **getStringCellValue ()** -> which returns the string value.
- **getNumericCellValue ()** -> which returns the double value.
- **getBooleanCellValue ()** -> which returns the Boolean value.

```
String value = c.getStringCellValue();  
System.out.println(value);
```

Write a program to read data from 1st cell of excel sheet.

```

import java.io.FileInputStream;
import java.io.IOException;
import org.apache.poi.openxml4j.exceptions.InvalidFormatException;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.ss.usermodel.WorkbookFactory;

public class ReadCellValue {
    public static void main(String[] args) throws IOException, InvalidFormatException {
        System.out.println("Program Starts");
        String FilePath = "./TestData/Data.xlsx";
        FileInputStream fis = new FileInputStream(FilePath);

        Workbook wb = WorkbookFactory.create(fis);

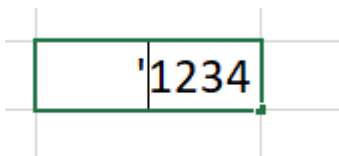
        Sheet s = wb.getSheet("Sheet1");
        Row r = s.getRow(0);
        Cell c = r.getCell(0);
        String value = c.getStringCellValue();
        System.out.println(value);

        System.out.println("Program Ends");
    }
}

```

Note: File input stream will throw “**IOException** (Checked exception)”.

- Create method of workbook factory will throw “**Invalid Format Exception (Checked Exception)**”
- If sheet name, row number, cell number is not correct or if the cell is blank then it will through “**Null Pointer Exception (Unchecked Exception)**”.
- Both row number and cell number starts from zero.
- “**getStringCellValue**” we can read only the string present in the cell, if the cell contains number we should use “**getNumericCellValue**”.
- Instead of using different methods for different types of data we can use single quote (') at the beginning in the excel cell so that we can always read the value using “**getStringCellValue**” method.



After adding the (') quotes, excel cell will look like below,



Write a program to print the number of cells present in each row of the excel sheet?

```
import java.io.FileInputStream;
import java.io.IOException;
import org.apache.poi.openxml4j.exceptions.InvalidFormatException;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.ss.usermodel.WorkbookFactory;

public class NumberOfCellPresentInEachRow {
    public static void main(String[] args) throws IOException, InvalidFormatException {
        String FilePath = "./TestData/Data.xlsx";
        FileInputStream fis = new FileInputStream(FilePath);

        Workbook wb = WorkbookFactory.create(fis);
        Sheet s = wb.getSheet("Sheet1");
        System.out.println("Total number of rows present is: " + s.getLastRowNum());
        int rowCount = s.getLastRowNum();
        for(int i=0; i<=rowCount;i++) {
            Row r = s.getRow(i);
            int cellCount = r.getLastCellNum();
            System.out.println("Total number of cells present in "+i+"th row is: "+cellCount);
        }
    }
}
```

Write a code to print all the values from the excel?

```
import java.io.FileInputStream;
import java.io.IOException;
import org.apache.poi.openxml4j.exceptions.InvalidFormatException;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.ss.usermodel.WorkbookFactory;

public class PrintAllValuesInExcel {
    public static void main(String[] args) throws IOException, InvalidFormatException {
        String FilePath = "./TestData/Data.xlsx";
        FileInputStream fis = new FileInputStream(FilePath);
        Workbook wb = WorkbookFactory.create(fis);
        Sheet s1 = wb.getSheet("Sheet1");

        int rc = s1.getLastRowNum();
        for (int i = 0; i <= rc; i++) {
            int cc = s1.getRow(i).getLastCellNum();

            for (int j = 0; j < cc; j++) {
                String value = s1.getRow(i).getCell(j).getStringCellValue();
                System.out.print(value + " ");
            }

            System.out.println();
        }
    }
}
```

Writing the date in a new cell?

- In order to write the data into new cell first we should create the cell then we should store the value using “**setCellValue**”, if the specified row doesn’t exist then we should create it using the “**createRow**” method. While writing the data into excel file we always use “**createCell**” method.

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import org.apache.poi.openxml4j.exceptions.InvalidFormatException;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.ss.usermodel.WorkbookFactory;

public class WritingTheData {

    public static void main(String[] args) throws InvalidFormatException, IOException {
        String FilePath = "./TestData/Data.xlsx";
        FileInputStream fis = new FileInputStream(FilePath);

        Workbook wb = WorkbookFactory.create(fis);
        Sheet s1 = wb.createSheet("Sheet2");
        Row r = s1.createRow(0);
        Cell c = r.createCell(0);
        c.setCellValue("Qspiders");

        FileOutputStream fos = new FileOutputStream(FilePath);
        wb.write(fos);
    }
}
```

Write a code to test the login feature of actiTIME?

- Create the following data in excel sheet.

A	B	C	D	
UserName	Password	Expected_result	Result	
admin	manager	actiTIME - Enter Time-Track		
admin	mnagher	actiTIME - Login		
user	manager	actiTIME - Login		
admin	manager	actiTIME - Login		
user2	user23	actiTIME - Login		

```

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.concurrent.TimeUnit;
import org.apache.poi.openxml4j.exceptions.InvalidFormatException;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.ss.usermodel.WorkbookFactory;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class Demo {

    public static String testLogin(String UserName, String PassWrod) throws InterruptedException {
        System.setProperty("webdriver.chrome.driver", "D:/Softwares/Drivers/chromedriver.exe");
        WebDriver driver = new ChromeDriver();
        driver.manage().window().maximize();
        driver.get("https://demo.actitime.com/");
        driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);

        driver.findElement(By.id("username")).sendKeys(UserName);
        driver.findElement(By.name("pwd")).sendKeys(UserName);
        driver.findElement(By.id("loginButton")).click();
        Thread.sleep(10000);
        String title = driver.getTitle();
        driver.close();
        return title;
    }

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

        String Status;
        String FilePath = "./TestData/Data.xlsx";
        FileInputStream fis = new FileInputStream(FilePath);
        Workbook wb = WorkbookFactory.create(fis);
        Sheet s = wb.getSheet("Sheet1");

        int rc = s.getLastRowNum();
        for (int i = 1; i <= rc; i++) {
            String username = s.getRow(i).getCell(0).getStringCellValue();
            String password = s.getRow(i).getCell(1).getStringCellValue();
            String eTitle = s.getRow(i).getCell(2).getStringCellValue();
            String aTitle = testLogin(username, password);
            s.getRow(i).createCell(3).setCellValue(aTitle);
            if(eTitle.equals(aTitle)) {
                Status = "PASS";
            }else {
                Status = "FAIL";
            }
            s.getRow(i).createCell(4).setCellValue(Status);
        }
        FileOutputStream fos = new FileOutputStream(FilePath);
        wb.write(fos);
    }
}

```

Acti

Generic methods to perform the operation on excel.

- Create the class "Excel Automation"

Add the following methods, below is the generic method to get the value from the excel.

```
import java.io.FileInputStream;
import java.io.FileOutputStream;

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

public class Excel {

    public static String getValue(String excelPath, String sheetName, int rowNum, int cellNum) {
        String value;
        try {
            FileInputStream fis = new FileInputStream(excelPath);
            Workbook wb = WorkbookFactory.create(fis);
            Sheet sheet = wb.getSheet(sheetName);
            value = sheet.getRow(rowNum).getCell(cellNum).getStringCellValue();
        } catch (Exception e) {
            value = "";
        }
        return value;
    }
}
```

Generic method to get the total number of rows in the excel sheet.

```
public static int getTotalRow(String excelPath, String sheetName) {
    int rowCount;
    try {
        FileInputStream fis = new FileInputStream(excelPath);
        Workbook wb = WorkbookFactory.create(fis);
        Sheet sheet = wb.getSheet(sheetName);
        rowCount = sheet.getLastRowNum();
    } catch (Exception e) {
        rowCount = -1;
    }
    return rowCount;
}
```

Generic method to write the data to excel cell value

```
public static void setValueToCell(String excelPath, String sheetName, int rowNum, int cellNum,
    String valueToWrite) {
    try {
        FileInputStream fis = new FileInputStream(excelPath);
        Workbook wb = WorkbookFactory.create(fis);
        Sheet sheet = wb.getSheet(sheetName);
        sheet.getRow(rowNum).getCell(cellNum).setCellValue(valueToWrite);
    } catch (Exception e) {
        System.out.println("Unable to set the value");
    }
}
```

Generic method to save the excel sheet after writing.


```

public static void writeToExcel(String excelPath) {
    try {
        FileInputStream fis = new FileInputStream(excelPath);
        Workbook wb = WorkbookFactory.create(fis);
        FileOutputStream fos = new FileOutputStream(excelPath);
        wb.write(fos);
    } catch (Exception e) {
        System.out.println("Unable to write");
    }
}

```

Note: Add all the method to Excel Automation class.

Write a script to test the login feature of actiTIME using the above generic methods?

```

package com.excelactions;

import java.io.IOException;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class Demo1 {
    public static String testLogin(String UserName, String PassWrod) throws InterruptedException {
        System.setProperty("webdriver.chrome.driver", "D:/Softwares/Drivers/chromedriver.exe");
        WebDriver driver = new ChromeDriver();
        driver.manage().window().maximize();
        driver.get("https://demo.actitime.com/");
        driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);

        driver.findElement(By.id("username")).sendKeys(UserName);
        driver.findElement(By.name("pwd")).sendKeys(UserName);
        driver.findElement(By.id("loginButton")).click();
        Thread.sleep(10000);
        String title = driver.getTitle();
        driver.close();
        return title;
    }

    public static void main(String[] args) throws IOException, InterruptedException {
        String Status;
        String FilePath = "./TestData/Data.xlsx";

        int rc = Excel.getTotalRow(FilePath, "Sheet1");
        for (int i = 1; i <= rc; i++) {
            String username = Excel.getValue(FilePath, "Sheet1", i, 0);
            String password = Excel.getValue(FilePath, "Sheet1", i, 1);
            String eTitle = Excel.getValue(FilePath, "Sheet1", i, 2);
            String aTitle = testLogin(username, password);
            Excel.setValueToCell(FilePath, "Sheet1", i, 3, aTitle);
            if(eTitle.equals(aTitle)) {
                Status = "PASS";
            } else {
                Status = "FAIL";
            }
            Excel.setValueToCell(FilePath, "Sheet1", i, 4, Status);
        }
        Excel.writeToExcel(FilePath);
    }
}

```