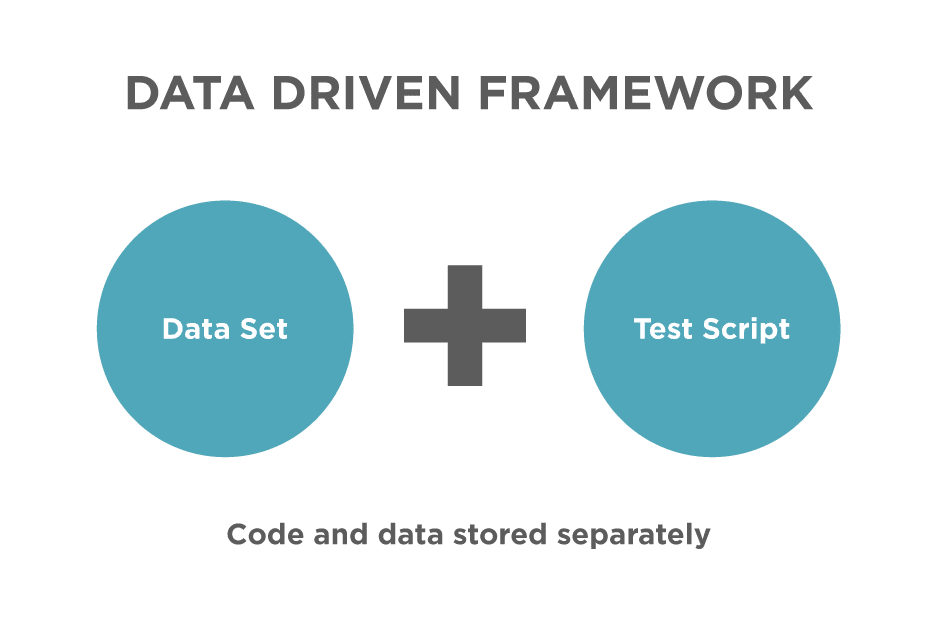
**Task#3**

### ****What is Data Driven Framework****

Data Driven framework is focused on separating the test scripts logic and the test data from each other. Allows us to create test automation scripts by passing different sets of test data. The test data set is kept in the external files or resources such as MS Excel Sheets, MS Access Tables, SQL Database, XML files etc., The test scripts connect to the external resources to get the test data. By using this framework we could easily make the test scripts work properly for different sets of test data. This framework significantly reduces the number of test scripts compared to a modular based framework.

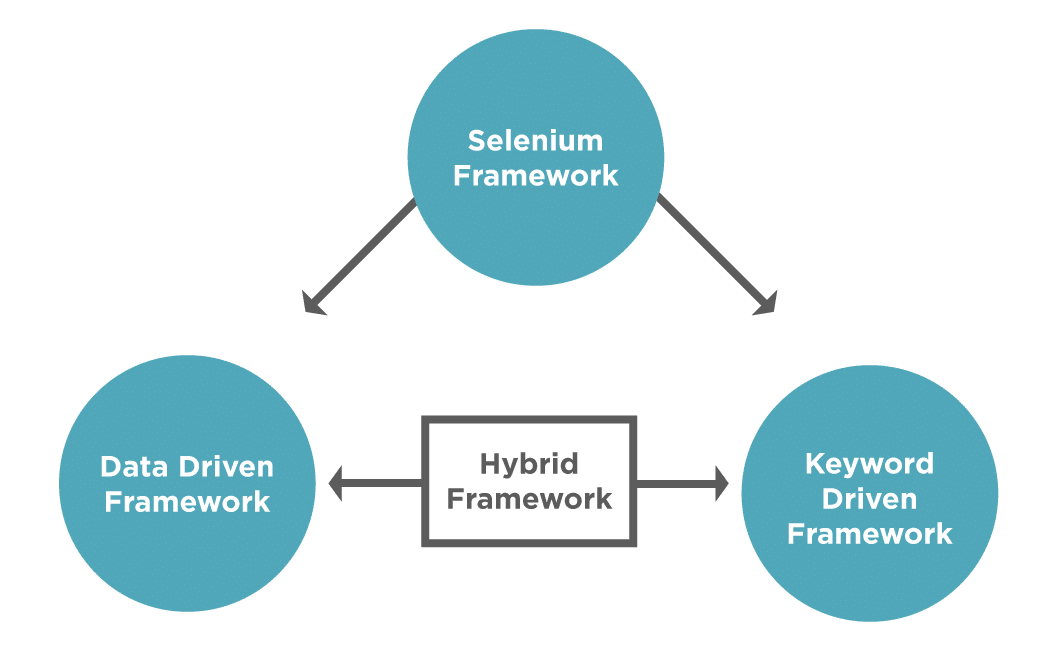


### ****Why Data Driven Framework****

Usually, we place all our test data in excel sheets which we use in our test runs. Assume, we need to run a test script (Say, login test) with multiple test data. If we run the same test with multiple test data sets manually is time-consuming, and error-prone. In simple words, we adopt Data Driven Framework when we have to execute the same script with multiple sets of test data.

### ****Advantages of using Data Driven Test Framework****

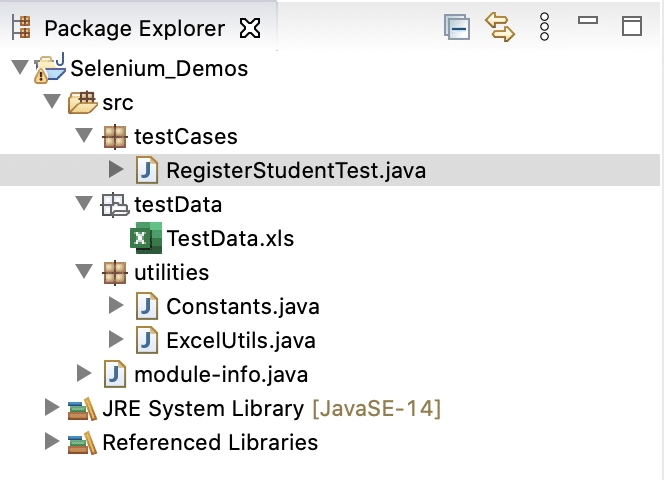
* Re-usability of code
* Improves test coverage
* Faster Execution
* Less maintenance
* Permits better error handling



A basic thumb rule for the data driven testing framework would be to segregate the test data from the test scripts. Also, the actions to read/write the data from files should segregate and be available as utilities.

Follow the steps as mentioned below to create a basic Data Driven framework, which will be used to automate the **["Student Registration Form".](https://demoqa.com/automation-practice-form)**

* Create three **[New Packages](https://www.toolsqa.com/selenium-webdriver/configure-selenium-webdriver-with-eclipse/)** in your Project for testCases, testData, and utilities.
* Under the ****testData**** package, put your Excel Sheet that has test data. Using this, we separate the test data from the ****testCases.****
* Under the ****utilities,**** **[create a New Class](https://www.toolsqa.com/selenium-webdriver/configure-selenium-webdriver-with-eclipse/" \l "package)** and name it ****"ExcelUtils"****. It will contain all functions related to Excel used for reading and writing.
* Under the ****utilities**** package, create another class ****"Constants"****. It will contain the constant values across the framework like testdata file path, URL of the application, etc.
* Under the ****testCases**** package, we will create the test files that contain the Selenium code for interacting with web elements. (For Example, RegisterStudentTest.java)



After performing the above steps, the folder structure will look like:

Let's understand the details of each of these classes:

****ExcelUtils Class -**** It is a utility class that will contain all the methods related to Excel Sheet read and write operations along with initializing the Workbook. You can then reuse the methods in different test cases, by creating an object of Excel Utils Class. The code for this class will be as below -

package utilities;

import org.apache.poi.hssf.usermodel.HSSFCell;import org.apache.poi.hssf.usermodel.HSSFRow;import org.apache.poi.hssf.usermodel.HSSFSheet;import org.apache.poi.hssf.usermodel.HSSFWorkbook;

import java.io.File;import java.io.FileInputStream;import java.io.FileOutputStream;import java.io.IOException;

public class ExcelUtils {

private static HSSFWorkbook workbook;

private static HSSFSheet sheet;

private static HSSFRow row;

private static HSSFCell cell;

public void setExcelFile(String excelFilePath,String sheetName) throws IOException {

//Create an object of File class to open xls file

File file = new File(excelFilePath);

//Create an object of FileInputStream class to read excel file

FileInputStream inputStream = new FileInputStream(file);

//creating workbook instance that refers to .xls file

workbook=new HSSFWorkbook(inputStream);

//creating a Sheet object

sheet=workbook.getSheet(sheetName);

}

public String getCellData(int rowNumber,int cellNumber){

//getting the cell value from rowNumber and cell Number

cell =sheet.getRow(rowNumber).getCell(cellNumber);

//returning the cell value as string

return cell.getStringCellValue();

}

public int getRowCountInSheet(){

int rowcount = sheet.getLastRowNum()-sheet.getFirstRowNum();

return rowcount;

}

}}

public void setCellValue(int rowNum,int cellNum,String cellValue,String excelFilePath) throws IOException {

//creating a new cell in row and setting value to it

sheet.getRow(rowNum).createCell(cellNum).setCellValue(cellValue);

FileOutputStream outputStream = new FileOutputStream(excelFilePath);

workbook.write(outputStream);

}

}

The above code contains different methods like ****setExcelFile**** to initialize the Excel Workbook, ****getCellValue**** to retrieve the value present in a particular cell in the file, ****setCellValue**** to set some value into a newly created cell. In a similar way, you can create different methods related to excel operations in this class.

****Constants Class-**** It is used to put constant values in a file so that the same can be reused across test cases. One more advantage of placing values in separate files is that since these values are common across various tests if there is any change in any of the values, you will just have to update in one place. For example, if the file path is changed, then instead of updating all the test cases with the new value, you can just update it here in one file. The structure and values present in this class are as below -

package utilities;

public class Constants {

public static final String URL = "https://demoqa.com/automation-practice-form";

public static final String Path\_TestData = "E:\\Projects\\src\\testData\\";

public static final String File\_TestData = "TestData.xls";

}

****RegisterStudentTest-**** It is the test script for the student registration form, which we used to enter the first name, last name, mobile, email, gender, etc for a particular student. Since we have now separated the excel related methods in a separate file, the code of our test case also changes.

We will create an object of ****ExcelUtils**** class in this test file and also use ****Constants**** to refer to the path of the file.

The updated code now looks like -

package testCases;

import org.openqa.selenium.By;import org.openqa.selenium.JavascriptExecutor;import org.openqa.selenium.WebDriver;import org.openqa.selenium.WebElement;import org.openqa.selenium.chrome.ChromeDriver;import utilities.Constants;import utilities.ExcelUtils;import java.io.IOException;import java.util.concurrent.TimeUnit;

public class RegisterStudentTest {

//creating object of ExcelUtils class

static ExcelUtils excelUtils = new ExcelUtils();

//using the Constants class values for excel file path

static String excelFilePath =Constants.Path\_TestData+Constants.File\_TestData;

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

//set the Chrome Driver path

System.setProperty("webdriver.chrome.driver","E:\\Projects\\chromedriver.exe");

//Creating an object of ChromeDriver

WebDriver driver = new ChromeDriver();

//launching the specified URL

driver.get("https://demoqa.com/automation-practice-form");

//Identify the WebElements for the student registration form

WebElement firstName=driver.findElement(By.id("firstName"));

WebElement lastName=driver.findElement(By.id("lastName"));

WebElement email=driver.findElement(By.id("userEmail"));

WebElement genderMale= driver.findElement(By.id("gender-radio-1"));

WebElement mobile=driver.findElement(By.id("userNumber"));

WebElement address=driver.findElement(By.id("currentAddress"));

WebElement submitBtn=driver.findElement(By.id("submit"));

//calling the ExcelUtils class method to initialise the workbook and sheet

excelUtils.setExcelFile(excelFilePath,"STUDENT\_DATA");

//iterate over all the row to print the data present in each cell.

for(int i=1;i<=excelUtils.getRowCountInSheet();i++)

{

//Enter the values read from Excel in firstname,lastname,mobile,email,address

firstName.sendKeys(excelUtils.getCellData(i,0));

lastName.sendKeys(excelUtils.getCellData(i,1));

email.sendKeys(excelUtils.getCellData(i,2));

mobile.sendKeys(excelUtils.getCellData(i,3));

address.sendKeys(excelUtils.getCellData(i,4));

//Click on the gender radio button using javascript

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript("arguments[0].click();", genderMale);

//Click on submit button

submitBtn.click();

//Verify the confirmation message

WebElement confirmationMessage = driver.findElement(By.xpath("//div[text()='Thanks for submitting the form']"));

//check if confirmation message is displayed

if (confirmationMessage.isDisplayed()) {

// if the message is displayed , write PASS in the excel sheet using the method of ExcelUtils

excelUtils.setCellValue(i,6,"PASS",excelFilePath);

} else {

//if the message is not displayed , write FAIL in the excel sheet using the method of ExcelUtils

excelUtils.setCellValue(i,6,"FAIL",excelFilePath);

}

//close the confirmation popup

WebElement closebtn=driver.findElement(By.id("closeLargeModal"));

closebtn.click();

//wait for page to come back to registration page after close button is clicked

driver.manage().timeouts().implicitlyWait(2000,TimeUnit.SECONDS);

}

//closing the driver

driver.quit();

}

}

The above class will perform the actions on the student registration page. However, if you notice, methods of the ****ExcelUtils**** handle all the excel related code.

So, this is one of the ways you can use the data driven framework in Selenium. Additionally, you can utilize the advantage of running the same test across multiple sets of data.

**What is Hybrid Framework?**

Hybrid framework is a technique wherein we can make the best use of both Data Driven & Keyword Driven Selenium framework (s). Using the examples shown above in this blog, we can build a Hybrid framework by storing the methods to execute in an excel file (keyword driven approach) and passing these method names to the Java Reflection Class (data driven approach) instead of creating an ****If/Else**** loop in the “DriverScript” class.

Take a look at the modified “DriverScript” class in the below code snippet. Here, instead of using multiple If/ Else loops, data driven approach is used to read the method names from the excel file.

|  |  |
| --- | --- |
|  | **package** HybridFramework;  **import** java.lang.reflect.Method;  **public** **class** DriverScriptJava  {   //This is a class object, declared as 'public static'   //So that it can be used outside the scope of main[] method  **public** **static** Actions actionKeywords;  **public** **static** String sActions;   //This is reflection class object, declared as 'public static'   //So that it can be used outside the scope of main[] method  **public** **static** Method method[];  **public** **static** **void** main(String[] args) **throws** Exception   {   //Declaring the path of the Excel file with the name of the Excel file   String sPath = "C:UsersVardhanworkspaceSelenium Frameworks DemodataEngine.xlsx";   //Here we are passing the Excel path and SheetName to connect with the Excel file   //This method was created previously   ReadExcelData.setExcelFile(sPath, "Sheet1");   //Hard coded values are used for Excel row & columns for now   //Later on, we will use these hard coded value much more efficiently   //This is the loop for reading the values of the column (Action Keyword) row by row   //It means this loop will execute all the steps mentioned for the test case in Test Steps sheet  **for** (**int** iRow=1;iRow<=7;iRow++)   {   sActions = ReadExcelData.getCellData(iRow, 1);   //A new separate method is created with the name 'execute\_Actions'   //You will find this method below of the this test   //So this statement is doing nothing but calling that piece of code to execute   execute\_Actions();   }   }  //This method contains the code to perform some action  //As it is completely different set of logic, which revolves around the action only, it makes sense to keep it separate from the main driver script  //This is to execute test step (Action)  **private** **static** **void** execute\_Actions() **throws** Exception   {   //Here we are instantiating a new object of class 'Actions'   actionKeywords = **new** Actions();   //This will load all the methods of the class 'Actions' in it.   //It will be like array of method, use the break point here and do the watch   method = actionKeywords.getClass().getMethods();   //This is a loop which will run for the number of actions in the Action Keyword class   //method variable contain all the method and method.length returns the total number of methods  **for**(**int** i = 0;i<method.length;i++)   {    //This is now comparing the method name with the ActionKeyword value received from the excel  **if**(method[i].getName().equals(sActions))   { //In case of match found, it will execute the matched method    method[i].invoke(actionKeywords);     //Once any method is executed, this break statement will take the flow outside of for loop  **break**;   }   }   }  } |