**Data Driven Testing** - выполнение какого то теста с разными данными. For example if we test authorization form we could want to test it with:

valid\_login+valid\_password,

invalid\_login+valid\_password,

invalid\_login+invalid\_password…

In other words, to test the same scenario with positive and negative sets of data.

With this approach, the main part is data. And we should not have any data in the automation script. The data we need to maintain in the separate file.

Excel ← these type of file are the main source for storing/receiving data. (also it can be notePad, TextPad…).

Java and Selenium do not have special classes to work with Excel. We need an additional library Apache POI.

**Apache POI** library integrates Java and Excel, to add Apache POI to POMxml file search for dependency Apache POI in Maven central repository. We are looking for 2 dependency:

1. Apache POI Common

2. Apache POI API Based On OPC and OOXML Schemas

Any Excel document has a Hierarchy of Excel:

excel file →workbook→sheets→Rows→Columns/cells.

**For every component in an Excel (see hierarchy) we have an appropriate class in Apache POI library.**

**For Excel files** we use FileInputStream class and FileOutputStream that are provided by Java. By using these classes we can open any files (not only Excel) for reading and writing

For workbook we use **XSSFWorkBook** class

For sheets we use the **XSSFSheet** class.

For rows we use the **XSSFRow** class.

For columns we use the **XSSFCell** class.

The data presents in the cell (column). If we want to write or to read data from the cell we need to navigate starting from excel file than to workbook than sheet than row than cell than we will be able to work with data.

**NOTE!** This API works only with Microsoft Excel not with Excel from other vendors(продовцов).

Ниже пример из IDE, здесь мы читаем из Excel файл расположенный в корневой папке проекта (выводим все столбцы)

**FileInputStream fileInputStream = new FileInputStream("TestData/ExperementData.xlsx");**

**XSSFWorkbook xssfWorkbook = new XSSFWorkbook(fileInputStream);** //создание workBook

**XSSFSheet sheet = xssfWorkbook.getSheet("Лист1");** //указать название sheet оно указано с лева в низу в Excel-e

//xssfWorkbook.getSheetAt(0);//можно передовать индекс sheet, наченается с 0

**//every sheet contains a number of rows. Before reading the data from the sheet we need to know the number of rows //in the sheet**

**System.out.println(sheet.getLastRowNum());** //count from 0 узнаём сколько столбцов в ряде

**//we can not jump from sheet to cell we have to go through the rows**

**System.out.println(sheet.getRow(1).getLastCellNum());** //вернёт индекс последней колонки в первом ряду

**//inner loop represents the Excel document.**

**//Outer loop for rows**

**//Inner for cells**

**for(int rowNumber=0; rowNumber<= sheet.getLastRowNum();rowNumber++){**

**XSSFRow currentRow = sheet.getRow(rowNumber);**

**for(int cells =0; cells<sheet.getRow(1).getLastCellNum(); cells++){**

**XSSFCell cell = currentRow.getCell(cells);**

**//String cellValue = currentRow.getCell(cells).toString();**

**System.out.print(cell.toString()+" ");**

**}**

**System.out.println();**

**}**

**xssfWorkbook.close(); //надо закрывать**

**fileInputStream.close();**

Ниже пример из IDE, здесь мы **вносим изменения** в Excel файл расположенный в корневой папке проекта

Мы должны следовать иерархии, если в предыдущем примере мы что-то считывали из Excel, то для записи всё тоже самое, мы должны следовать иерархии

Так как щас изменяем файл то работаем с классом FileOutputStream

**FileOutputStream fileOutputStream = new FileOutputStream("TestData/writingToExcel.xlsx");**

//create workbook, предпологается что Excel пустой поэтому мы создаём новый workbook

// поэтому не передоём в параметрах ничего

XSSFWorkbook xssfWorkbook = new XSSFWorkbook();

//Создаём новую страницу /sheet

XSSFSheet sheet = xssfWorkbook.createSheet();

//Create **MANUALLY** the rows and 3 cell in every row, and write some rundom info in the every cell

/\*XSSFRow row1 = sheet.createRow(0);

row1.createCell(0).setCellValue("Hello");

row1.createCell(1).setCellValue("Carnival");

row1.createCell(2).setCellValue(555);

//Create one more row

XSSFRow row2 = sheet.createRow(1);

row2.createCell(0).setCellValue("Hellogen");

row2.createCell(1).setCellValue("Birma");

row2.createCell(2).setCellValue(444);

\*/

//Create **WITH LOOPS** the rows and cells

for(int rowsN=0;rowsN<26;rowsN++){

XSSFRow currentRow = sheet.createRow(rowsN);

for (int cellN=0;cellN<14;cellN++){

currentRow.createCell(cellN).setCellValue(cellN);

}

}

//Important step, we need to attach the created document to our Excel file

xssfWorkbook.write(fileOutputStream);

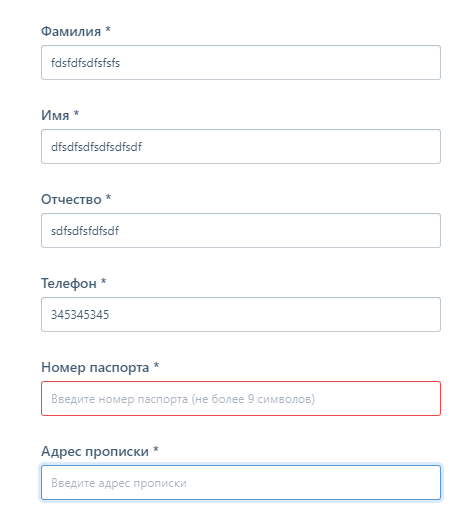
xssfWorkbook.close();

fileOutputStream.close();

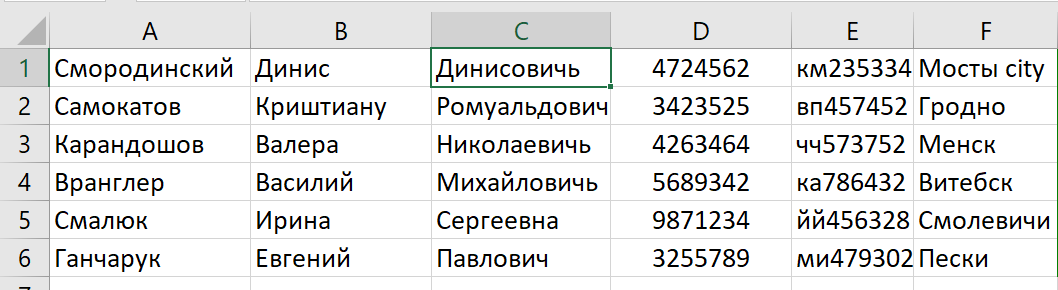
Итак данная разновидность тестов основана на применении разных данных к одному и тому же тесту, но нам нельзя хранить данные прямо в тесте. Данные хранят отдельно, в этом примере рассматривается Excel document.

У Каждого теста есть часть кода которая работает с приложением и есть часть кода которая обращается к данным (к Excel), при этом код обрабатывающий приложение в каждом тесте (если у нас не один тест) будет разный, а та часть кода которая обращается к Excel будет одинакова в каждом тесте. Решением является создания Utility class-a, в котором будут описаны методы обработки данных из Excel, в этом случае не надо писать код каждый раз один и тот же в каждом тесте, надо только использовать методы из Utility class-a от этого тесты станут более понятны и увеличится reusability.

1)-Определится с веб страницей

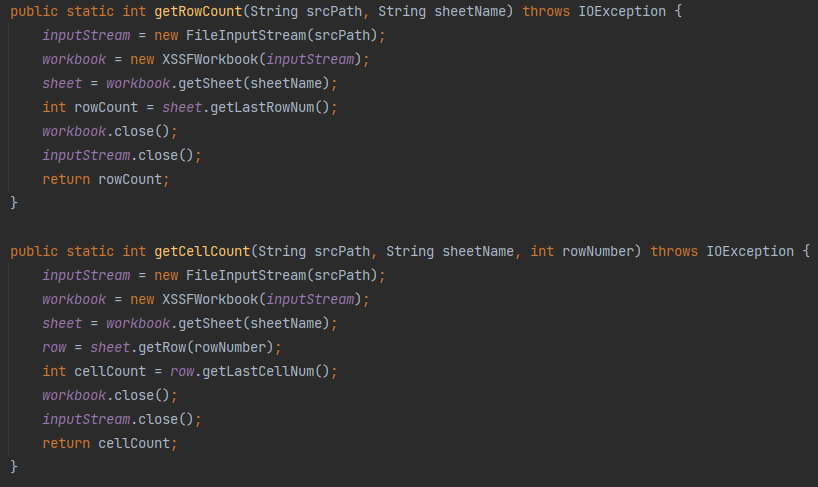


2)-подготовить данные с которыми мы будем работать(подготовить Excel)



3)- Напишим Utility class

C:\Users\user\IdeaProjects\RedRoverForTrainig\src\main\java\workWithExcell\UtilityFileForExcel.java



4) test

C:\Users\user\IdeaProjects\RedRoverForTrainig\src\main\java\workWithExcell\DDTPracticeWithZags.java

