Sercan Şensülün 242001007
Emre Karakış 232001023

Tuğkan Tuğlular
CENG 554

Homework 1 Report

What we try to do ?

Project covers that when user/tester enters *Rules* on *Decision Table* with respect to *T* as **True** , *F* as **False** and *X* as **active action** on Excel file, it converts these rules as List of Abstraction Test Cases and put them into Abstract Test Cases tab on ~~same Excel file~~.We did change it as there exist two folders in project called as Input and Output. After execution, user could find results into excel file which locates under Output folder.

What did we in terms of Object Oriented ?

We try to explain our UML design as class by class with their explanations, you could find code comments on implementation of project too.

ExcelFile.java
It actually supplies as Excel File as real world example. It has physical *path* as **String** and *List* of **Tab** as given template. Thanks to this class, every Tab knows that which ExcelFile object they belongs and also their physical path. Thanks void setTab() method, we could create any Tab what we want and set them into **Tabs** which is represented as List of Tab objects.

<abstract> Tab.java
It is a super class for every type of *Tabs* which exist or going to be added on Project. It has *hostExcelFile* as **ExcelFile**, *sheet* as **Sheet** (class of apache.poi), *eTabType* as **ETabType** and *name* as **String**. Tab knows which file it belongs with *hostExcelFile* property and *sheet* equals Tab structure on real world excel file and thanks to it we could read records as row

by row. In project, there exist more than one type of Tab and we could understand easily that what is the type of given instance of Tab thanks to eTabType property.It has two abstract methods which are Boolean read() and Boolean write() should be override from child class. Implementation of child Tabs could be different from each other thats why we did prepare these abstract methods.

When we analyse given excel file template, we recognise that there are mainly two kinds of Tab. One of them represents object as row by row and the other one represents as column by column, and we decided to create two kinds of class as abstract which are **RowTab** and **ColumnTab**.

<abstract>RowTab<T>.java
It represents its object as row by row. It holds its objects into *rows* as **List<T>**.

<abstract>ColumnTab<T>.java
It represents its object as column by column. It holds its objects into *columns* as **List<T>**.

RequirementRowTab.java
It holds *rows* as list of **Requirement**, *type* of Tab as **ETabType.Requirements** and its *hostExcelFile* as **ExcelFile.** Thanks to super class as abstraction, we could implement / override Boolean read() and Boolean write(). In project domain, we assume that user enter inputs on Requirement Tab that why Boolean write() method did not implemented. On the other hand, into Boolean read() method we read real values of Requirements thanks to apache.poi library and fill them into *rows.*

InputRowTab.java - OutputRowTab.java - ConditionRowTav.java - ActionRowTab.java
Logic is the same as explanation of **RequirementRowTab.** Into Boolean read(), actually the main logic is the same like Boolean read() of **RequirementRowTab** as reads from real world and fill them into rows, however source of file is different.

DecisionTableColumnTab.java
Project has only one **ColumnTab's** child class up to now.Its Boolean read() method is little bit complicated from the others, because ,as logic, it should be aware of list of **Condition** and **Action.**Thats why it pulls list of **Tab** from hostExcelFile and reads from instances of **ConditionRowTab** and **ActionRowTab.**

AbstractTestCaseRowTab.java
We override both Boolean read() and Boolean write() methods. Logic of Boolean read() is same as the others, this class should be aware of list of **Rule**. The critical part of class is of

course Boolean write() method because when it is aware of list of **Rule**, it should create list of **AbstractTestCase** and write them perfectly onto Abstract Test Case excel tab.

Requirement.java
It represents requirement on project with description property as **String**.

<abstract> Parameter <T>.java
When we analyse **Input** and **Output** as domain, we did notice that these objects are actually have same properties for this project that why we did create it. It has two different properties which are name as **String** and parameter as **T**, because parameter could be Integer or String etc… for Input and Output.

InputVariable<T> .java
For now it has only constructor which connects to its super class.

OutputVariable<T> .java
For now it has only constructor which connects to its super class.

Condition.java
It has two properties which are isOccurred as **Boolean** and value as **String.** isOccurred indicates that given condition is T or F, and value is actually a tag to make sense full for everybody such as a>b etc.

Action<T>.java
It has two different properties that are actionType as **EActionType** and parameter as **T**. Thanks to enum property we understand that what is the instance of action do or do not.While doing its action, to send any parameter we could use parameter property.

Rule.java
Thanks to conditions as **List<Condition>**, we could notice that which conditions are T or F for an instance of Rule, and we could understand which **Action<String>** should be performed from actions as **List<Action<String>>.** For given triangle example, there is only one **Action** for each **Rule** but it could have more than one that's why it's List type.

DecisionTable.java
It has only one property rules as **List<Rule>.**

AbstractTestCase.java
It has two properties which are conditions as **List<Condition>** and action as **Action<?>.**