SALES & COMMISSIONS MANAGEMENT

REENGINEERING THE LEGACY CODE

	Date 9/18/2023

GOAL OF THE PROJECT

The goal of this project is to reengineer a Java application. The general purpose of the application is to manage the sales made and the commissions attributed to the sales representatives of a small clothing company. Generally the application accepts as input txt or xml files that contain receipts information about the sales made by a sales representative. The user of the application can add more receipts manually using the graphical user interface of the application. Based on the receipts information the application calculates the commission to be paid to the sales representative by the clothing company. The application generates respective reports in txt or xml format.

PREPARATION

1. Skim the documentation: The legacy application has been developed based on a more detailed requirements specification that is available along with the application source code (Sales-Commissions-Requirements.pdf). In a first step, study the documentation to get more information concerning the application's requirements.

2. Do a mock installation: The application source code is provided as an eclipse project (sales-commissions-application folder). Setup a running version of the project and try to use its basic functionalities.

3. Build confidence:

    a. Read all the source code once and try to understand the legacy architecture, the role/responsibilities of each class, and so on.

    b. Specify the use cases as they are implemented in the legacy application.

    c. Capture the legacy architecture in terms of a UML package diagram.

    d. Specify the detailed design in terms of UML class diagrams.

4. **Implement tests for the use cases of the application.**

REFACTORING TASKS

Refactor the application so as to fix the following problems:

data package:

1. **Coat, Shirt, Skirt, Trouser classes:** the problem here is **Lazy Classes** that do not do much to deserve to exist. The lazy classes can be removed.

2. **Agent class:** this class has a couple of problems.

   a. Firstly, the name of the class does not reflect the role of the class in the application domain. A better name should be given.

   b. Another significant problem of the class is **Duplicate Code**. Several methods differ only in constants and variables. The duplicate code should be removed.

   c. The class is using the Java Vector data structure which is deprecated. Replace the data structure with a new one and make the respective changes in the algorithms that use the data structure.

3. **FileAppender, FileAppenderTXT, FileAppenderXML classes:** These classes also have a couple of problems.

   a. The classes update the contents of files that contain receipt information. Based on this functionality the names of the classes are not clear. Better names should be given.

   b. Since the classes perform IO operations it seems that they are placed in the wrong package. The classes should be moved to the right package.

4. **FileAppender class:** Another problem of this class is primitive obsession. The class has too many fields of a primitive type that can be replaced by a Receipt object that encapsulates the primitive data. Appropriate changes can be made to the class to reduce the number of class methods.

5. **FileAppenderTXT, FileAppenderXML classes:** observe that the implementations of the appendFile() methods follow the same algorithmic steps to update a file of receipts. The common algorithm can be extracted in a template method[1] that will be placed in the FileAppender class.

input package:

[1] https://refactoring.guru/form-template-method

1. **TXTInput class:** The algorithm that is used in this class to parse txt input files is more complex than necessary given that the order of the elements within the txt file is pre-determined and fixed. Replace the algorithm with a better and simpler one.

2. **Input, TXTInput and XMLInput classes:** after the refactoring of TXTInput to a simpler algorithm, chances are that the parsing algorithms of TXTInput and XMLInput would follow the same steps. The common algorithm can be extracted in a template method[2] that will be placed in the Input class.

output package:

1. **Report, TxtReport, XMLReport:** these classes have a couple of problems.

    a. Firstly, the names of the classes are not adequate. Observe that the classes generate reports they are not the reports themselves. Better names should be given.

    b. Observe that the implementations of the saveFile() methods follow the same algorithmic steps to save a report file of receipts. The common algorithm can be extracted in a template method that will be placed in the base class class.

gui package:

1. Refactor the gui to allow the user select a particular file for saving a report, instead of using a predefined file name.

2. **InputWindow, SelectionWindow, ResultWindow class:** the names of the classes can be improved to better reflect their responsibilities.

3. **SelectionWindowClass:** this class has several problems.

    a. The addReceiptButtonPressed() has a very complex conditional statement that can be improved.

    b. In the appendFile() method we observe chains of method calls that can be removed. The size of the method can also be significantly reduced based on the data package refactoring 4.

[2] https://refactoring.guru/form-template-method

c. The code of the okButtonPressed() method seems more complicated than necessary. Probably the multiple if-else statements can be simplified.

EXTENSION TASK

Extend the application by adding the following functionality and respective test(s):

1. Provide input to the application via an html file. The different elements of the file should be structured with respect to appropriate html tags like headings, lists and so on.

2. Store a report for a sales representative in an html file. The different elements of the file should be structured with respect to appropriate html tags like headings, lists and so on.

PREPARE DELIVERABLES

A **report** based on the given template (Project-Deliverable-Template.doc). For the delivery of the report include a pdf of the report in the project folder. Turn in the **refactored project** and the other deliverables using **turnin deliverables@mye004 <your-project>.zip**, where your-project is a zip file of your Eclipse refactored project.