MINNESOTA INCOME TAX CALCULATION PROJECT

PHASE 1 - REVERSE ENGINEERING AND QUALITY ASSESSMENT

	Date 10/2/2018

GOAL OF THE PROJECT

The goal of this project is to reengineer a Java application. At a glance, the application serves for **the income tax calculation of the Minnesota state citizens**. The tax calculation accounts for the marital status of a given citizen, his income, and the amount of money that he has spend, as witnessed by a set of receipts declared along with the income. The legacy application takes as **input txt** or **xml** files that contain the necessary data for each citizen. The tax calculation is based on a complex algorithm provided by the Minnesota state. The application further produces graphical representations of the data in terms of **bar and pie charts**. Finally. the application produces respective **output** reports in **txt** or **xml**.

PHASE 1 TASK LIST

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 (MinessotaIncomeTaxCalculation-Requirements.pdf). In a first step, study the documentation to get more information concerning the application's architecture and use cases.

2. **[Do a mock installation]** The application source code is provide as an eclipse project (IncomeTaxCalculatorProject folder). Few test input files are also available in the project's main folder (e.g. 123456789_INFO.txt, 130456094.xml, ...). Setup a running version of the project and test it based on the given input files.

   ATTENTION the main folder of the project includes 3 jar files (forms-1.3.0, jcommons-1.0.23, jfreechart-1.0.19) that should be included in the project's libraries.

3. **[Build confidence]** Read all the source code once and try to understand the legacy architecture, the role/responsibilities of each class, and so on.

4. **[Write tests to understand and enable evolution]** Prepare test cases that will allow you to understand and test the inner workings of the classes. Use JUnit for the tests. More specifically, develop JUnit tests that automatically test the back end of the application (i.e. the incometaxcalculator.data.io and the incometaxcalculator.data.management classes). Test the following functionalities:

   - **Tax calculation algorithms:**

- Tax calculation algorithm (calculateBasicTax() of SingleTaxpayer).
- Tax calculation algorithm (calculateBasicTax() of HeadOfHouseholdTaxpayer).
- Tax calculation algorithm (calculateBasicTax() of MarriedFilingJointlyTaxpayer).
- Tax calculation algorithm (calculateBasicTax() of MarriedFilingSeparatelyTaxpayer).
- **ATTENTION** to cover all cases of income layers we need multiple tests for each algorithm.

- **Input facilities**
  - Load taxpayer personal information from txt file (readFile() of TXTFileReader class).
  - Load taxpayer personal information from xml file (readFile() of XMLFileReader class).
  - **ATTENTION** the readFile() methods of the *FileReader classes do not return any values; instead, the methods change the contents of two static HashMap fields, stored in the TaxpayerManager class. Hence, the assertions of the tests should be written based on the methods of the TaxpayerManager class.

- **Output facilities**
  - Save taxpayer personal information to a txt file (generateFile() of TXTInfoWriter class).
  - Save taxpayer personal information to a xml file (generateFile() of XMLInfoWriter class).
  - Save taxpayer payment information to a txt file (generateFile() of TXTLogWriter class).
  - Save taxpayer payment information to a xml file (generateFile() of XMLLogWriter class).
  - **ATTENTION** the generateFile () methods of the *Writer classes do not return any values; instead, the methods produce output files. Hence, the assertions of the tests should be written based on the contents of the output files that can be read back by the test code.

	Date 10/2/2018

5. **[Capture the design]** Specify the application architecture in terms of a UML package diagram. Specify the detailed design in terms of UML class diagrams. Prepare CRC cards that describe the responsibilities and collaborations of each class.

6. **[Identify problems]** Assess the quality of the code.

    a. Check the quality of the code using the CheckStyle plugin for eclipse (http://eclipse-cs.sourceforge.net/#!/). Install the plugin. Start from the standard SUN Checks that correspond to the standard Java coding conventions. Keep all the rules sections unchanged, except for the **"Size Violation" rules section** that you should change as follows:

        1. Max file size = 200 lines.
        2. Max method length = 10 lines.
        3. Max line length = 80 chars.
        4. Max number of parameters = 2.

    b. Look for more general problems

        i. Detect possibilities of problematic classes with many responsibilities.

        ii. Detect possibilities of problematic classes with very few responsibilities.

        iii. Detect possibilities of similar classes/methods with duplicated code.

7. **[Prepare report]** prepare a detailed report based on the given template (Project-Deliverable-Phase1.doc).