CPSC4360, CPSC5360 - Software Engineering
Spring Semester, 2015
Project Specifications and Instructions

Submission date: the last three weeks of the semester

Project Description (The 'Automatic Document Generation'):

This project (**The 'Automatic Document Generation'**) requires you to adopt a use-case driven object-oriented approach to develop a system to support the **automatic generation of a document based on data saved on a database or other input from the environment for the benefit of the customers.**

Motivation:

We live in a society guided by computers. For example, we receive many documents automatically generated by the computers (such as the monthly electrical bill, phone bill, water bill, bank statements, and yearly social security benefits letters and more). These documents come to our mail as letters or by email, or require us to login into their website to check their status. The purpose of an Automatic Document Generation is to help those institutions to automatically and accurately generate those documents. The second purpose is to facilitate the customer to obtain those documents by log in into a website and process a payment, if needed.

Difficulties:

An Automatic Document Generation is a quite complex task. Some of the data do not change often or periodically (such as the address, name, identification or account number, and more), but some of the data are changed periodically (such as the amount of kilowatts consumed per 30 days, the amount of water consumed per 30 days, the number of minutes or Gigabytes consumed per 30 days, the contribution to the personal social security account, and more). The software should consider both categories of data.

Structure:

The software should include two databases:

1. One database contains the unchangeable data, such as address, name, identification or account number. This database can change its data as the request of the customer (changing address, etc) or by the institution in charge of the statements (bank account number, identification number, etc);
2. The second database contains the changeable data for a period of time, such as the amount of minutes spent on the phone, the amount of kilowatts spent by your dwelling, etc. These data are read by the institution (either electronically, or by sending a person to update the data in the database).

The 'Automatic Document Generation' Computer Support:

All services enumerated in the above Structure should be accompanied by the computer. For example, the databases should be maintained as files (text, DB2, EXCEL, or more). The software should also include a way to inform the customer that a previous payment was not received as well as to acknowledge that a payment was received. You need to identify which data are unchangeable or modifiable, as well as to manage the payment process and other transaction details using a computer.

The 'Automatic Document Generation' Requirements:

1. To develop a system that will hold information about data and their services.

- **1.1.** To register/login online (or in person) as a customer looking to check the status of a bill;
- **1.2.** To process a payment relative to a bill.

1.3. To inform the customer about the hurricane/disaster preparedness related to payment of the bill.

1.4. To inform the customer about the benefits of their institution (energy saving tips, social security benefits, etc).

1.5. To inform the customer about the legislative activities related to their institution.

2. To provide the history of bills.

2.1. If a customer wants to compare the current data with the last year data, the Automatic Document Generation should provide this information to the customer and help him/her decide the best thing to do.

3. To provide an estimate of how much a customer will most likely consume in the future or how much would be the benefits or awareness if some data are changed.

4. The system must be able of future expansion to incorporate information about existing Automatic Document Generation and how to expand them in the future.

Your team will work together to perform Object-Oriented Analysis and design and implementation of the proposed System. If you find that certain important information about the requirements is missing or ambiguous, you may explore on your own and suggest additional requirements or refine given requirements. While doing so, make reasonable assumptions. Your assumptions must not conflict with each other or with the given requirements. The problem and solution space needs to be decomposed among the classes. The system is to be modeled in UML and implemented in Java. It is not however compulsory to program in Java, hence you can actually choose any other object-oriented programming language, such as C++, Python, etc.

Instructions

You are required to use Java (or another object-oriented programming language) to implement classes according to specific software requirements, especially the persistency requirement. You should be able to demonstrate the functionality of the system through a menu-driven interface. It can be a text-based menu, GUI or web-based interface. Type of interface does not have any impact on project evaluation.

You should make all functions very easy to use (user friendly). For example, you should not expect the user to enter all parameters required in one long string. You can use GUI, but you may have to bear the risk of having a system that does not work correctly with the test driver (see following paragraph).

Testing

You are required to perform **black box testing** on the functionality of classes and the interaction between them. To carry this out you may either implement a test driver or use JUnit to execute automated testing. Ideally the menu-driven interface and the test driver can be combined as one program. If you have difficulty in combining them, it is acceptable to separate them into 2 programs.

Data Persistency

You are allowed to use Java file system to retrieve and populate the data by using the **object serialization** to meet the persistency requirement.

Deliverables:

I. Project Report:

Your report should include the following:

(a) A complete **use-case diagram** of the proposed system illustrating functionality to be developed by your team. A **domain model** is also requested for later refinement into the class diagram.

(b) A **class diagram** to illustrate design for the proposed system. The diagram should include classes, associations, multiplicities, attributes, and methods. You are required to use UML notation for drawing the class diagram. Please choose relevant and meaningful names for classes, attributes, associations, and

methods. Level of details for class diagram e.g. how detailed method signature should be, is left to the team members' decision.
(c) Any assumptions, interpretation, and limitations that you have included while developing the models. Please be concise. Write all these as a separate section at the beginning of your report or after the introductory note, if you have any.
(d) An **interaction diagram** of the most complex use case scenario chosen by your team; a **state chart** for any object from your design; and an **activity diagram** for the system you propose. If you find that drawing one activity diagram is difficult or not possible, you are allowed to provide multiple activity diagrams. However, a brief reasoning for the same should be included in the report.
(e) **Test cases** used for automated testing.
Important: The total page limit for report is 15 pages, single spaced letter report (include all diagrams, appendix and references if any). The report will be assessed for abstraction, completeness, clarity, correctness, structure and innovative design /discussion /presentation.

II Project Demonstration

Each team is required to demonstrate:
- functionality of its system through a menu-driven interface;
- testing of its system through an automated test driver.
Each team is to submit its project source code along with report. Put the source code in a CD. The content of the CD should at least include the following:
a) Source code of the system. Put them in a subdirectory called **"source"**.
b) Test driver and test cases that team used for testing your system. Put them in a subdirectory called **"testing"**.
c) A "`Readme.txt`" file containing instructions on how to compile and run your system. Indicate important configuration information, e.g., the class-paths (if any) that need to be set. Any special requirements, e.g., a web server must be installed, or MySQL must be installed etc., should be indicated as well.
Note: As we will be using the source code on your CD for demonstration, please ensure that all the required files in your CD are not corrupted. You will not be allowed to use a fresh copy of your source code during the demonstration. Remember to label your CD with your team members' names (at least).
There is no need to include additional third party software in your CD.
What to bring for the demonstration: Each team is to bring one laptop for doing the demo. We will run your system on this laptop. Ensure any third party software necessary for running your system is installed.

Important points to note:

Each team has about 10-15 minutes to do the demonstration during the weeks starting April 22 until May 7, 2015. You will be providing your own test data. Size and variety of data should be adequate to demonstrate the functionalities of your system. It is preferable that your system has a 'bulk loading' function to load all the test data in a single operation (say from text file(s) containing the test data), rather than adding all the test data one-by-one into your system (which can be painfully slow).