Due dates: The submission of the project will consist of 3 deliverables and 3 short presentations:

1/ Requirements document; Increment 1: February 19th.

2/ Design document; Increment 2: March 12th.

Note: For Increment 1 and 2 each team is required to give a 10mn presentation at the end of the lecture.

3/ Test plan and test results: Increment 3: April 9th: System deliverable.

Note: For Increment 3 each team is required to give a 15 mn presentation at the end of the lecture.

Submission: Submit formats exactly as requested. Documents must be as pdf files and be readable. Source code must be compressed using tar/ zip as indicated; otherwise they cannot be read. Electronic Submission through the Moodle course Homepage by midnight of the due date for each deliverable.

Description

In this project, you will implement a **Personal Budget Manager Application**

Develop a budget Manager Application to help individuals control their expenses. It is distinguished between two types of expenses: *purchase* expenses and *Bill* expenses. The former have a description being linked to retailor(s) (name, location, amount, date, etc.) and status ((*paid* (e.g. cash, debit), *due date* (credit card)). The latter have a description (name of the service company), status (paid, not paid) and a repetition interval (e.g. weekly, monthly, yearly… etc.).

Basic Functionality for Increment 1

The Personal Budget Manager Application should have the following functionalities:

- View all expenses of the spending list (in arbitrary order).
- Add a new expense.
- Mark an expense as paid (or not paid).
- Remove an expense.

The Personal Budget Manager application shall be equipped with a Java Swing User Interface. The expense table should refresh its display automatically when the expenses in the expense list change.

Planned future extensions of the Personal Budget Manager are:

- Addition of new expense types.
- Support for multiple user interfaces (e.g., Web UI, Mobile UI).
- Multiple presentations of the expense list (e.g., using trees or lists).

Basic Functionality for Increment 2

This iteration of the Personal Budget Manager extends the previous iteration by adding the following features:

The expense types (*purchase*, *bill*) instead of being atomic, they can be composite of sub-items expenses (e.g.: *VidéoTron*: Homephone/Internet/Television; *Food&Dinning*: Coffee/Groceries/Restaurants…etc.) which in turn may be complex expenses. Sub-items expenses are dependent on their super-expenses. If a super-item expense is removed from the expense list all its sub-items expense are also removed. Similarly, if the payment status of a super-item is set to complete or incomplete, the completion status of all sub-items will change accordingly.

In this increment implement the following

- View expenses with their hierarchy in case of composite expenses.
- Delete expense.
- Set expense to paid/unpaid.

- Add expense: *purchase* or *bill* as single/periodic/composite.
- Hiding / unhiding paid expenses: where the user has the option to change the current view and to hide/unhide paid expenses (at the root level).

Additional Functionality for Increment 3

In the third Iteration of the Personal Budget Manager Application:

- Create a database schema for storing the various expenses by choosing an appropriate mapping strategy knowing that any expense is either direct payment (cash/debit: with items receipts) or credit (items listed on credit cards statements): document your database schema as a UML class diagram.

- Implement a class that allows for opening and closing a database connection.

- Extend the functionality of the Personal Budget Manager such that at *start-up* all expenses can be loaded from the database. Additionally, at any time, the user shall have the opportunity to commit (save) the current expense in the repository to the database.

Deliverables Submission:

1. Make sure that you have signed the expectation of originality form available on the Moodle course Web page.
2. In addition, write the following statements on your submissions
    - ***Team name:*** *"We certify that this submission is the original work of members of the group and meets the Faculty's Expectations of Originality"*, with the signatures and I.D. #s of all the team members and the date.
3. Hand in each deliverable electronically:
    - Create one zip file, containing all files for each increment.
    - Upload your zip file through your Moodle course Homepage as *TeamName_Increment1*, *TeamName_Increment2*, and *TeamName_Increment3*.

Diary

During the project, each student must keep a **diary** detailing every activity undertaken in relation to the project. Supply the date and time for each activity, together with its duration, a list of the participants, and a brief description of the activity. Provide a summary of the total time spent on the project for each week of the semester.

Keep the diary up to date as the semester proceeds. Keep it as a Latex document.

For each increment from 1 to 3, each student must individually submit their diary as a pdf file. Submit it as "StudentID_Diary1", " StudentID_Diary 2", and "StudentID_ Diary3" respectively.

COMP 5541 Software Engineering Project: Guidelines[1]

This page describes a project for the *COMP5541: Tools and techniques for Software Engineering course*. It focuses on learning the software process and the inter-relatedness of the activities in a project, rather than placing emphasis on the product itself.

The process emphasizes

- test-driven development, to ensure good understanding of requirements, and working software;
- agile development, to encourage iteration, to divide the work into small chunks, and to allow students to improve their skills from one iteration to the next;
- object-oriented software development, in particular software systems as collections of collaborating objects, with a focus on responsibility-driven design, an emphasis of separating interfaces from implementations, and the use of patterns;
- communication, by being team-based, and requiring basic documentation using UML for domain model, use cases, architecture and design, and testing.

There are three iterations. Each iteration produces a working system: the increment.

1. The first iteration constructs a small working system with basic functionality. It focuses on core application rather than GUI design, advanced features, or unnecessary features. It provides an easy-to-understand task while students ensure that they are familiar with all the tools, and students learn to work as a team. The documentation at this stage is a Requirements Document containing a domain model and the major use cases.
2. The second iteration aims to incorporate all the major functionalities that will impact the architecture and design. Generally, the architecture is Model-View-Control, so the second iteration does incorporate a GUI. The documentation at this stage is a Design Document emphasizing an overview of the architecture and the interfaces.
3. The third iteration considers one or more advanced features for the system, mainly to challenge the students, to let them see how robust their design is in light of the new features, and to experience testing beyond unit testing of functionality.
   Both the second and third iterations should construct useful working systems.
   The documentation at this stage is a Test Document containing a test plan, test suite, and test results.
4. **[Optional]**The fourth iteration allows a final polishing step to eliminate any remaining major problems, and to ensure consistency of source code with each of the documents.

Teams and Roles

The project is team-based. Each team of approximately 4 to 5 students.

There are three roles on the project:

- **Tester** - creates test cases, codes unit tests, and runs unit tests to check all is working.
- **Documenter** - should produce the document, but the content for the document is the joint responsibility of the team.
- **Developer** - should produce working source code, but Testers produce and code the unit tests.

[1] http://users.encs.concordia.ca/~gregb/home/se-w2013-project.html

Each subteam assumes a role of either Tester, Documenter, or Developer for the duration of an iteration. During the course of the three iterations, each subteam will play each of the three roles.

Note that the project is a team effort:

- Tester needs to talk to Developer to know class interfaces, and the expected results of each method. Tester needs to know use cases and check that each is testable (by creating test cases). Tester also is involved in review of the design.
- Documenter should produce the document for the iteration where they play the role of Documenter. The content of each document is the joint responsibility of the team: for each of the documents, Requirements Document, Design Document, Test Document.
- Developer in iteration 1 is heavily involved in design of the interface for the core application. Developer in iteration 2 is heavily involved in creating the design. Developer in iteration 2 is heavily involved in designing UI.

Standards

The standard software development environment in the department consists of:

- Java as the programming language, using either Swing or SWT for GUI development;
- Eclipse as the integrated development environment (IDE);
- JUnit for unit testing in Java;
- Rational Rose for UML modeling; and
- Latex for documentation.

Your background should mean that you know these languages and tools, or know similar languages and tools, and can easily learn to proficiently use these languages or tools.

You should be competent in object-oriented software development, and be capable of producing working, well-tested, documented source code for software systems of moderate size.

You should know how to design an interface for a class. You should know how to use classes and design your own classes for software systems of moderate size. You should know standard data structures and standard algorithms.

You should know how to create a desktop application that is commandline driven, as well as one that has a simple GUI.

You should know Linux / Windows.

Increment 1

Documents should be submitted as "Project 1" and Source code as "Program 1".

The use case document (see an example) should contain a use case diagram to provide the context for the overall system, together with a description of the actors and the major use cases. Choose an appropriate level of detail for each use case. The document should also include a (problem) domain model consisting of a UML class diagram and explanatory text.

Submit a .tar.gz version of your source code (including JUnit tests).

Increment 2

Documents should be submitted as "Project 2" and Source code as "Program 2".
Example of a design document.
In section 3 there is one subsection per subsystem. Remove (comment out) the explanatory text in the template.
Submit a .tar.gz version of your source code (including JUnit tests).

Increment 3

Documents should be submitted as "Project 3" and Source code as "Program 3".
Document is a short summary of your test plan and test results: basically tell me how thoroughly you tested and whether there are still outstanding bugs or issues.
Example of a test document

Make sure that each test is individually numbered. Regard inputs with multiple effects/results as multiple tests, and number them individually.
Note that you may have some common situations/scenarios such as sample file contents, sample documents, sample screen positions, etc. used in many tests. Document and number them in a section for reference in the test cases.
The test plan should indicate which tests cases apply to which requirements, and make it abundantly clear (traceable) that each requirement is tested.

Submit a .tar.gz version of your source code (including JUnit tests).