GC01 Introductory Programming (40% of GC01 mark)

Major Coursework - **Due Sunday 24th November 2013, 5pm, Moodle.**

Teams have been formed of 3 or 4 participants and clients have been allocated.

You will be creating a basic Java or HTML5/JS application framework in this coursework based on a specification of work assigned to you by a real world client. As described in the lectures and in the slides given, there are two major components

(1) An object model consisting of code files, either in Java or in HTML5/JS
(2) A Requirements and Teamwork document in Word/PDF format

The weighting for (1) and (2) are 20% each out of the available 40% for GC01.

Note for passing marks – the system designed will be detailed and thoroughly model each process and use case, **with code examples** for each major requirement, though the overall code does not need to execute. The report will have discussed requirements clearly, identified work packages for the team and demonstrated what is to be built. It would also describe the team work plan effectively as to who is doing what and by when.

Note for high distinction marks - If you are aiming to successfully reach a very high distinction level on this GC01 project, the system will be a thorough investigation of the problem, describing in detail every process with completed data structures (arrays/collections), algorithms and a full documentation of the requirements and work plan. Please follow the following guidance on these two components.

Requirements for (1) Object Model in Java Code or HTML5/JS Code

The software produced must have the following;

(1) All objects (or groups of variables and methods with a unique name) are well defined; in addition, arrays and object collections are also well described.
For example, a login object has a username, password and Boolean method for isPasswordAccepted. A scoreboard object is an array (fixed at 10) of the top 10 highest score objects, sorted by highest score, and contains name, score, date and time. A file controller object calls read and write methods as requested by a gui object, from a file reader/writer object (showing several objects belonging together).
(2) All data types are to be identified – this should be detailed in each object model class. A table should be included in the Requirements document of all of this as the "data dictionary" (see below).
(3) All inputs and outputs are described in code – including mouse actions, reading and writing files, reading and writing to websites, reading and writing from local databases.
(4) All processes and algorithms described in as much code detail as possible e.g. any calculations, sorting or searching or transmitting of data.

(1) Implementation Suggestions

The project should comprise of several well organised packages of classes (as in Java packages e.g. com.GC01.AppName.*) which you will categorise and define in your development process. A core set of model object classes should be *packaged* together.

All data should come from files, either from text files or from HTML documents that you create. You should spend some time designing the format and information required based on your client requirements.

A higher level *package* of *library* classes with accessors (get) and mutators (set) should be available as *controllers* to the users' view-level classes (read up on "MVC – Model View Controller" in Java). All related classes should be hierarchically organised, for example, the GUI set of classes should belong in its own package layer, e.g. com.GC01.AppName.GUI.

Implementation criteria for (1)

Reusable programming structures (prototype)

1. Accurately modelling the core learning architecture of the solution as a series of objects **or** well-contained units of code, including all data primitives and appropriate getting and setting methods.
2. Extensive use of arrays or collections (arraylist!) for checking and iterating through data. Also verification that a sequence is correct (correct logic in the iteration).
3. Demonstrating appropriate use of Strings manipulation, appropriate use of "extends" and "interfaces" with inheritance, abstract classes, data containers, selection (if/else) and iteration (for/while) processes.
4. Demonstrating Java or JavaScript API calls of library methods, e.g. file handling input and output and ideas for graphics e.g. any charting or drawing of images.
5. Package structure of your Java object model.
6. Don't forget to describe a welcome screen, error message handling, and a help page object!

If you are doing a Java Object Model then you do not need any GUI (Swing/AWT) code to pass, but advanced marks will be awarded for any team that does include a basic GUI as a model prototype (even if it is not connected directly to the objects behind) that you can demonstrate to your client. You can of course use WindowBuilder in Eclipse to drag and drop the GUI components into place. A JTabbedPane on a JFrame is a relatively similar GUI concept to browsing through several app panels. If you are doing a HTML5/JS Object Model then you will need to show some examples of how the HTML5 browser layer connects to the code in JavaScript behind (e.g. several example webpages should be completed at least). You are allowed and encouraged to make use of modern HTML5/JS/CSS editors such as Aptana for Eclipse, TopStyle 5, Visual Studio 2012 etc.

Requirements for (2) Requirements and Teamwork Document

A word or pdf document that describes:

1. A description of the team and its primary and **several** secondary roles allocated to each member as responsibilities:
   Allocate all roles from this list { **Primary team lead, secondary team lead, client main contact, background researcher, technology researcher, systems designer and analyst, data designer, UI designer, documentation lead, primary developer, secondary developer, software tester, device and deployment tester**}
2. A single paragraph of description of the app problem
3. A single paragraph on background knowledge of the problem, with any appropriate referenced citations
4. the client interview questions and answers, including technology and platform requests.
5. A task list of all packages of the app that need to be created, with who is working on what parts of the system and documentation.
6. A schedule (gantt chart or spreadsheet by time) of the work packages for this coursework and for GC02's app design.
7. A data dictionary of every variable in the system, tabulated – some will have multiple places where they belong indicating a transfer of data from one part of the system to another.
8. A client-reviewed series of GUI paper prototype sketches of every screen of the solution as it should be as an app – this can be drawn on paper and submitted as JPG images in the Word document but it should be annotated with object types, on the GUIs. E.g. a form would indicate what is a textfield, radio button, pull down menu, button etc. If you manage to create some basic GUI prototypes either in Java WindowBuilder or HTML5/JS then please also take screenshots and include them in your report.
9. A tabulated list of use cases that describe the key activities e.g. User logs in, has a menu on a home screen, searches a record, adds it if it doesn't exist, validates the fields, saves it, verifies it is saved and returns to the home screen.
10. Any client feedback (emails or written notes or photographs) or comments on the first prototype and requirements documentation, ready to build the app for the solution in GC02.
11. Any commentary on the team performing together.
12. (Hard) a software engineering investigation of the project:
    a. this includes a basic uml notation of how the objects will connect diagrammatically,
    b. state chart diagrams,
    c. inheritance modelling,
    d. demonstrating what conditions are needed for data coming to and from files, any transactions (that should roll back if not completed/errors are made) and any sessions (for temporary existential data).

The more detailed this document solution, the higher the mark. Bonus marks will be awarded in GC01 and in GC02 by discretion where evidence is shown of further research, incorporation of 3rd party libraries and good package construction is shown to create new APIs for future projects to reference this one.

Grading and submission files

. You are required to submit a single ZIP file on moodle containing:

1. The source code files of your Object Model in either Java or HTML5/JS. Include the Eclipse project workspace or Visual Studio project files.
2. An instruction guide (a text file) on what is completed and if it is executable, how to run it.
3. Your Requirements and Teamwork report in Word or PDF format.This will be passed through Turn-It-In for plagiarism detection.
4. (Optional) tests folder for any other demos (compiled and source code) that stand alone in demonstrating features of your system.

Notes

- If you include quoted text in your project, then 1) quote sparingly, 2) make sure the pertinent text is highlighted with quotation marks, and 3) provide a citation to the source of the quote (including full bibliographic detail). **Ensure any cited works are free and open for reuse.**

Tips

1. Keep in mind this is the programming model to your solution, and not necessarily a fully executing program. The purpose of this coursework is for you to practice the conversion of real-world problem domains into well-defined work packages and object oriented programmatic structures that can be described to others, before setting it in stone in a particular language or platform.
   It is not intended to be a purely programming exercise and it must giving clear information back to your team or client in its report. It intends to cover all of the "controller" and "model" with respect to the "Model-view-controller" design pattern, which is important for you all to investigate.
2. Respect your team, their roles, and **enforce regular meetings**. If you are struggling, list the things you are struggling with, meet up as a team and go through each item together, noting them in minutes (a meeting diary), suggesting where to find solutions and next steps. If you are still struggling, please contact the teaching assistants and the research associates, but not without first suggesting some solutions and next steps to your problems.
3. Revise eclipse strategies for creating packages, compiling, debugging methods, adding breakpoints, how to load existing Eclipse projects and how to import other libraries.
4. Reminder: Plagiarism of any kind, on the applications and on your content prepared will not be tolerated! All of the software developed must be your own works. Any Java code examples used from online sources and tutorials must be CITED with the original reference location.
5. You will learn along the way how best to test it with small test apps. You may keep these in a separate folder called tests if you wish to submit it with your main Zip file.
6. Please work in teams effectively, meet up often, share your findings and request your team double check each other's work. **The best of you will set up a GitHub (or Dropbox) and check in daily with both code and documentation, and include weekly minutes of each of your meetings in your report.**