Project PM: OO Programming Project, TDA367/DIT212, LP4 2016

General

The project group's first task is to decide on one of the project proposals on course page. This isn't critically related to the grade, almost any application can be complexified by adding more (and sophisticated) requirements.

Second task is to name the project.

The third task is to setup the project site. The project must use the Git versioning system (GitHub or other). Use the selected name!

Final task is to send a mail to course responsible (hajo@chalmers.se) with the following content; Group name, selected project (very short description),
URI to Git repository and lastly, info about all members. The member info format should be (if email doesn't make it obvious if you're a Chalmers or a GU, add Chl or GU. Phone optional);

```
email [Chl/GU], lastname, firstname, pnumber [phone]
email [Chl/GU], lastname, firstname, pnumber
email [Chl/GU], lastname, firstname, pnumber
email [Chl/GU], lastname, firstname, pnumber
```

In response you will get a confirmation mail with an assigned group number. The group number must be used in all communication with assistants, course responsible, etc. (also the group meetings, the seminar and the demo schedule will refer group numbers).

Requirements

This is what you are supposed to produce. All required documents and code should be downloadable from the Git repository (you don't handle in anything, except a self evaluation and a list of use cases, see presentation below). It's assumed that the branch master is the final delivery.

The application

Your group is supposed to develop a standalone desktop or mobile Java application with a graphical user interface (i.e. not a web application). The application must be based on an object oriented model and use some kind of MVC design. The project must be possible to run on Win/Mac/Linux(incl. Android).

You may extend the application (libraries, 2D/3D frameworks, client/server, database, etc.) but note;

- It's much harder to get a clean design when combining different technologies.
- It's not necessary to extend to get the highest grade.
- See project grading below!

The documentation

All documentations should be in pdf format. Standard templates should be used (see course site).

The RAD and SDD

1. A section in RAD or SDD should normally be short and concise probably require at most 3-5 sentences. If something isn't applicable just add a NA (not applicable) under the actual section (the sections are there to guide you, they are not mandatory).
2. UML-diagrams should be on class/package/module-level. We are normally not interested in variables and methods except possibly for methods in interfaces.
    a. There should be a class diagram of the domain model
3. NO auto generated UML (other sensible UML welcome).
4. Javadoc is NOT necessary. Comment if needed but note that we prefer self documentary code.
5. The use cases
    a. There should be at least five well documented use case texts.
    b. There should be an use case diagram (overview over all use cases).
    c. There should be two full sequence-diagram describing the dry-runs of two use case..

The group meeting agendas

Should be in Git repository, make sure they have a date.

The presentation

During the presentation the group should run a demo of the application and do a technical "walkthrough" (incl. slides). The demo should demonstrate all implemented use cases. During the walkthrough: try to explain the construction of your application in a pedagogical manner (especially cool features).

Right before the presentation the group should handle in a list of use cases (for us to tick off) and for each member a self evaluation (see course page)
The presentation will be about 20 min. The presentation is part of the grading

Project grading

What we appreciate is a well designed, smart application with as much functionality as possible. The look is second as is overly technical solutions.

It's very hard to formulate exact criterion for the projects but if a project consists of less than about 2.5k SLOC (source lines of code) including sensible amount of comments the project should have other qualities that compensate for the smallness of the project (that is, about 800 SLOC/member). If project or your contribution is small, document somewhere the reasons. We will use gitinspector to check size and contributions (link on course page).

Some points we will consider

BASIC: Size of application (classes), Total number of code lines (incl. other code (XML. ...), Code style (naming,...), Development environment organization (packages etc.)

DESIGN: Quality of domain model (compare use cases, class diagram and code), Partitioning, Possible to find cohesive parts of appl. (high cohesion/low coupling), Layering: Possible to identify abstraction layers, Clean MVC, Clean subsystems, Comments from tools: Findbugs, STAN, Exception handling, Motivated use of DP, no circular dependencies, etc.

IMPLEMENTATION: General technical level for implementation, no redundancy, Abstraction: Use of interfaces abstract classes, generics, ...model (or other clean solution)

GUI: Simple/advanced, modular, GUI features (Animations/2d/3d libraries),

TEST: Unit tests (how many, quality), Test code separated from appl. code, Code coverage (Jacoco)

FUNCTIONALITY: Number of working use cases (compare RAD, use cases and handled in list from presentation), Average complexity of use cases

ADVANCED ISSUES: Special features (distributed, i18n, other...), Use of external libraries, Use of frameworks

DOCUMENTATION: RAD (Would customer understand) SDD (Possible to quickly get an understanding of the construction of the application), Use cases, Agendas (possible to trace the process?), Class comments for all classes (purpose of class, class used by?)

Examples

Below are some typical examples (non exhaustive).

U (U)
Not possible to run. Project to small, to few use cases, design to strange/bad (hard to understand). Documentation missing or poor. Not possible to trace any process from agendas. No or bad presentation.

Grade 3 (G)
A project fulfilling the base requirements for all grades. Which means: Functionality; 8-10 use cases implemented and working. Simple but functional GUI. In-house MVC. Clean implementation of at least one subsystems. Application uses interfaces to reduce dependencies. There are some usable tests. Documentation is short but correct (and in sync. with the application). It's possible to trace requirements and follow the process. The presentation is ok.

Grade 4 (G)
(NOTE: This is a list of possibilities, not the union of ...) A somewhat larger application with a more sophisticated design, possibly use of (needed) design patterns. Solid code and packaging, everything is easy to locate. Clean subsystems.More functionality (use cases) implemented, possibly attention to non-functional requirements. Good control over dependencies, clean interfaces. Possibly use of external libraries. A more advanced GUI. External configuration and data. Test suites cover a lot of application code. Documentation is short and correct and obviously useful for others. The presentation gives a good view of the strength and weakness of the application.

Grade 5 (VG)
Like grade 4 but even more and with higher technical level and obviously smart features. Possibly some kind of modular design with plug-ins, or other advanced design.