Linnéuniversitetet | Fakulteten för teknik | 1DV607

Objektorienterad analys och design med UML

Workshop 2 – Design

Senast ändrad 2020-10-07 08:24 av ***Tobias Ohlsson***

This describes the task for the second workshop, perform all the steps in order. Be sure to document all assumptions and changes you are making. Also, be sure to specify who participated in the work and be prepared to answer any questions about your model. You must be a course participant and logged in to see the information below.

Please use English in your models and documents. You may get international students as reviewers.

All files should be zipped into an archive and be in one of the following formats. plain text (i.e. different source code files), pdf or common image formats (e.g. jpg, gif, png). **Do not use word documents etc**.

Note this workshop is only conceptually related to workshop 1 and you are not supposed to implement the requirements from workshop 1. The requirements for this workshop are listed below and this is the scope of the design and implementation in this workshop.

Requirements for grade 2

Design and implement a simple member registry with CRUD (Create, Retrieve, Update, Delete) functionality. Implementation (source code + "running" version), class- and interaction-diagrams are to be created and presented. The interaction diagrams should show how a model-view separation is achieved (i.e. start in the UI) and how the different requirements are met. Design and implementation should match.

The focus is **not** to create a usable or fancy user interface but to have a robust and well-documented design that can handle change and follows the GRASP. My recommendation is that you base your work on a console application.

OBS: It is not permitted to use any type of framework, however, class libraries, API:s etc are permitted. Basically, you should design and code your own application.

The following requirements are to be met:

1. Create a new member with a name, personal number, a unique member id should be created and assigned to the new member.
    1. The member id should be something that could be printed on a small membership-card and handled by a human mind, I.e. **no** superlong GUID like stuff
2. Show lists of all members in two different ways:
    1. "Compact List"; name, member id and number of boats
    2. "Verbose List"; name, personal number, member id and boats with boat information
3. Delete a member
4. Change a member's information
5. Look at a specific member's information
6. Register a new boat for a member the boat should have a type (Sailboat, Motorsailer, kayak/Canoe, Other) and a length.
7. Delete a boat
8. Change a boat's information
9. Persistence (the registry should be saved and loaded for example from a text file.)
10. Strict Model-View separation:
    - The model should not depend on the view or user interface in any way (direct or indirect)
    - The model should not have user interface responsibilities
    - The user interface (view) should not implement domain functionality
11. Good quality of code (for example naming, standards, duplication)
12. An object oriented design and implementation. This includes but is not limited to:
    - Objects are connected using associations and not with keys/ids.
    - Classes have high cohesion and are not too large or have too much responsibility.
    - Classes have low coupling and are not too connected to other entities.
    - Avoid the use of static variables and operations as well as global variables.
    - Avoid hidden dependencies.
    - Informations should be encapsulated.
    - Use a natural design, the domain model should inspire the design.
13. Simple error handling. The application should not crash but it does not need to be user friendly.
14. Participate in the peer review process

You should submit the following and all parts should match.

- A well tested runnable version of the application. For example, an .exe or .jar file, link to website etc. If it is not easy to run the application you must include instructions on how to run it.
- The source code of the application, with possible instructions on how to compile it, external dependencies etc.
- A class diagram for the entire application, focus on relationships of the classes and important details (do not add every single attribute or operation)
- Sequence diagrams that cover one input requirement (e.g. create member, register boat or change information) and one output requirement (e.g. list members or look at member info)
- Optional: A “readme.txt” explaining any assumptions, changes or clarifications to requirements made during the modeling.
- Optional: A “readme.txt” listing all the members of the group (first name, last name, user name).

Some tips on tools to create diagrams are yuml.me and websequencediagrams.com that seem to cover the needs we have and are quite agile. Some versions of Visual Studio has a number of tools for diagrams.

Tutorial on compiling your c# project into a self contained exe from vscode by Molly Arhammar.

Requirements for grade 3

- Perform the requirements for grade 2
- Expand the design and implementation to include:
    1. Simple authentication; a user must be logged in to create, change and delete information (members and boats). Not logged in users should be able to see information (list, see details and search (see below).
    2. A simple selection/search of members. That is a subset of the member should be listed, for example, members with a name that starts with “ni”, members who are older than a certain age, born a certain month, has a certain type of boat etc. A **GoF design pattern should be used** to solve the problem and it should be easy to add new types of criteria, that is the design should show how new criteria can be added and what needs to be changed. You do not need to implement all the above examples.

Requirements for grade 4

- Perform the requirements for grade 2
- Perform the requirements for grade 3
- Expand the design and implementation to include:
    1. Data validation and error handling with user-friendly error messages. The solution should avoid code duplication and be as flexible as possible.
    2. Complex selection/search of members. The model should be able to perform arbitrarily complex/nested selections of the type: “members who are born a certain month or has a name that starts with “ni” and are older than a certain age”. That is (month==Jan || (name=”ni*” && age > 18). **A GoF design pattern should be used** it is enough to show flexibility on the model-side, that is you do not need to implement a view (user interface) for this. A few “hard coded” examples on searches are good enough.