DASS Assignment 4

Code Review and Refactoring

Deadline: 18/04/2021

In the previous assignments, you have built software systems from scratch making use of different technologies and frameworks as well as implementing good coding standards to write extensible code. We will now look at another valuable skill along with code writing - code reading and analysis.

For this assignment, you will analyze and refactor the design on an existing software system. Your team will read and study the given codebase, reverse engineer the design, and propose refactoring to improve the code structure for future maintenance and evolution.

This is a team assignment and teams will be the same as your project teams. Please find your code here - https://drive.google.com/drive/folders/1OLPytD1JUy9PXOgqc3VPnrzpoBKTtuaP. The code base assigned to your team is <TeamNumber>.rar and team numbers are the same as those assigned at the beginning of the course.

Part 1: Code Review

For this part of the assignment, you will analyze the code to understand the existing design including its strengths and weaknesses. The design can be presented using UML diagrams. You will also find and report all bugs and code smells in the code.

(Read: https://blog.codinghorror.com/code-smells/, https://en.wikipedia.org/wiki/Code_smell)

Part 2: Refactoring

Now that you have analyzed the existing codebase and its design, you will suggest how you would improve the design. You will also suggest how you will refactor the code by using the various good practices taught in class. Make use of OOPs concepts, Design Patterns, elimination of code smells etc for this purpose.

Note: You do not need to refactor the code; only suggest improvements

Submission format

Every team has to submit a document TeamNumber.pdf containing the following information

1. Title information, including team number, team members, roll numbers etc. Mention clearly the contribution made by each member of the team.
2. A short overview section describing the software system you will study and the features. (**5 marks**)
3. A narrative analyzing the original design (note: the "original" design is what is in the code you reverse-engineered), its weaknesses and strengths etc. (**5 marks**)
4. UML class diagrams showing the main classes and interfaces in the design, along with inheritance (generalization), association, aggregation, and composition relationship for the **original design**. Include cardinality and role indicators as you deem appropriate to make the diagram clear. You may decide on the appropriate level of abstraction with respect to state or method information. You may need several class diagrams at different levels of abstraction and for different subsystems to completely document your design in a way that the reader can physically see and intelligently understand. (**15 marks**)
5. A table summarizing the responsibilities of the major classes. (**10 marks**)
6. A table containing all code smells found along with their short description (eg: which file contains it, what is the exact problem etc). **(15 marks)**

Code smell	**Short description**
#category	Example description

   Include other columns as needed
7. A table containing bugs along with their short description. (**BONUS 15 marks**)

Bug number	**Short description**
#category	Example description

   Include other columns as needed
8. A narrative highlighting the changes you would make to the existing changes and how it is better than the existing design. Your refactored design should contain at least two

instances of Design Patterns you would incorporate. Make use of good design principles taught in class. (**BONUS 20 marks**)

9. UML class diagrams showing the main classes and interfaces in the design, along with inheritance (generalization), association, aggregation, and composition relationships for your **proposed design**. Include cardinality and role indicators as you deem appropriate to make the diagram clear. You may decide on the appropriate level of abstraction with respect to state or method information. You may need several class diagrams at different levels of abstraction and for different subsystems to completely document your design in a way that the reader can physically see and intelligently understand. (**BONUS 15 marks**)