telerik
33, Alexander Malinov Blvd., Sofia, 1729, Bulgaria
Phone: (+359) 2 80-99-862; Fax: (+359) 2 80-99-888

High-Quality Programming Code – Team Projects – May – July 2014

Overview

You are given a **C# software project** designed to implement some of the following **console-based games**:

- Balloon Pops
- Battle Field
- Bulls and Cows
- Game 15
- Hangman
- King Survival
- Labyrinth
- Minesweeper

The project consists of one or more **source code files** and sometimes contains other files and is provided as ZIP archive. You need to refactor the project in order to **improve its quality** following the best practices learned in the course "High-Quality Programming Code" and to **implement unit tests** that ensure that the code has correct behavior.

Detailed Assignment Description

In order to ensure the high quality of the assigned project you need to fulfill the following tasks:

1. **Perform refactoring of the entire project** (its directory structure, project files, source code, classes, interfaces, methods, properties, fields and other class members and program members and its programming logic) in order to **make the code "high quality"** according to the best practices introduced in the course "High-Quality Programming Code". The obtained **refactored code** should conform to the following characteristics:

   - **Easy to read, understand and maintain** – the code should be well structured; should be easy to read and understand, easy to modify and maintain; should follow the concept of self-documenting code; should use good names for classes, methods, variables, and other identifiers; should be consistently formatted following the best formatting practices; should have strong cohesion at all levels (modules, classes, methods, etc.); should have loose coupling between modules, classes, methods, etc.; should follow the best practices of organizing programming logic at all levels (classes, methods, loops, conditional statements and other statements); should follow the best practices for working with variables, data, expressions, constants, control structures, exceptions, comments, etc.

   - **Correct behavior** – the project should fulfill correctly the requirements and to behave correctly in all possible use cases. This means that all bugs or other problems in the project (e.g. performance or usability issues) should be fixed and any unfinished or missing functionality should be completed. The code should be very well tested with properly designed unit tests.

2. **Implement design patterns** – redesign the project to fulfil 6 of the Software Design patterns

   - **Structural patterns** – implement at least 2 of the structural design patterns (adapter, aggregate, bridge, composite, decorator, extensibility, façade, etc…)

   - **Behavior patterns** – implement at least 2 of the behavioral design patterns (chain of responsibility, command, interpreter, iterator, mediator, observer, etc…)

33, Alexander Malinov Blvd., Sofia, 1729, Bulgaria
Phone: (+359) 2 80-99-862; Fax: (+359) 2 80-99-888

- **Creational patterns** – implement at least 2 of the creational design patterns (abstract factory, builder, factory method, singleton, prototype, etc…)

3. **Follow the SOLID and DRY principles** – Single responsibility, Open/close, Liskov substitution, Interface segregation, Dependency inversion, Don't repeat yourself

- Redesign the project to fulfil the SOLID and DRY principles – each principle should be implemented at least once

4. **Design and implement unit tests** covering the entire project functionality. To ensure the project works correctly according to the requirements and behaves correctly in all possible use cases, design and implement unit tests that cover all use cases and the entire program logic. If needed, first redesign the program logic to **make the code testable**. Test the normal expected behavior (correct data) and possible expected failures (incorrect data). Put special attention to the border cases. The code coverage of the unit tests should be at least 80%. Use unit testing framework of your choice (e.g. Visual Studio Team Test, NUnit, MbUnit or other).
5. **Document the refactorings** you have performed in order to improve the quality of the project. Use English or Bulgarian language and follow the sample (see below).

Deliverables

1. The **original source code** (project files, .cs files) without executables.
2. The **refactored source code** (project files, .cs files) without executables.
3. The **unit tests** – source code (project files, .cs files) without executables.
4. The **refactoring documentation**.

Team Work Requirements

- Obligatory use **Git** as source code repository and **GitHub** (http://github.com) as project hosting and team collaboration environment. SVN or TFS are **not** allowed for this project.
- **Each team member** should have contributions to the project and **commits in the source control repository in 3 different days**. We acknowledge that this requirement seems a bit unnatural, but we want to track **how the team collaborates over the time** and that the **project is developed incrementally**, not in the "last minute".

Other Requirements

- Pack the project deliverables in a **single ZIP archive**. Be sure to avoid including large unused files in the archives (e.g. compilation binaries). Your archive should be up to 8 MB. Each team member should submit the same archive as a homework.
- Be prepared as a team to **defend your project** in front of the course lecturers. You should be able to explain what refactorings have been performed and why. The documentation will definitely help you. Be prepared to **demonstrate how the unit tests cover the project's functionality**. Preferably bring your own laptop to reduce the effort to setup your development environment and project workspace.
- Be prepared to **show the commit logs** from the source control system to demonstrate how the project development efforts are shared between the team members and over the time.

33, Alexander Malinov Blvd., Sofia, 1729, Bulgaria
Phone: (+359) 2 80-99-862; Fax: (+359) 2 80-99-888

Discussion Forum

- You can freely discuss the course projects and ask questions in the official discussion forum of the course: http://forums.academy.telerik.com/csharp-programming/c%23-qpc

Sample Refactoring Documentation for Project "Game 15"

1. Redesigned the project structure: Team "…"
   - Renamed the project to **Game-15**.
   - Renamed the main class **Program** to **GameFifteen**.
   - Extracted each class in a separate file with a good name: **GameFifteen.cs**, **Board.cs**, **Point.cs**.
   - …
2. Reformatted the source code:
   - Removed all unneeded empty lines, e.g. in the method **PlayGame()**.
   - Inserted empty lines between the methods.
   - Split the lines containing several statements into several simple lines, e.g.:

```
if (input[i] != ' ') break;
```

→

```
if (input[i] != ' ')
{
  break;
}
```

   - Formatted the curly braces **{** and **}** according to the best practices for the C# language.
   - Put **{** and **}** after all conditionals and loops (when missing).
   - Character casing: variables and fields made **camelCase**; types and methods made **PascalCase**.
   - Formatted all other elements of the source code according to the best practices introduced in the course "High-Quality Programming Code".
   - …
3. Renamed variables:
   - In class **Fifteen**: **number** → **numberOfMoves**.
   - In **Main(string[] args)**: **g** → **gameFifteen**.
4. Introduced constants:
   - **GAME_BOARD_SIZE = 4**
   - **SCORE_BOARD_SIZE = 5**.
5. Extracted the method **GenerateRandomGame()** from the method **Main()**.
6. Introduced class **ScoreBoard** and moved all related functionality in it.
7. Moved method **GenerateRandomNumber(int start, int end)** to separate class **RandomUtils**.
8. …