Assignment 6 - Refactoring, Improving Software Architecture

Deadline June 2nd 23:59

Background

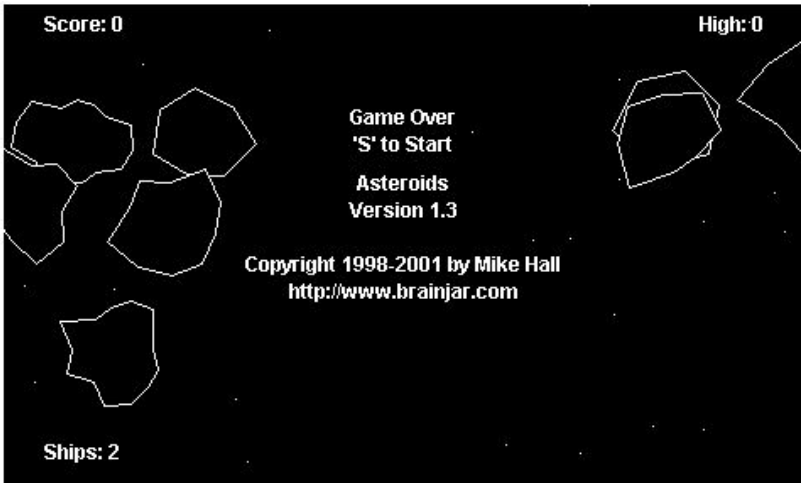
Getting a program to work is only half of the problem. A good program should be readable, maintainable, reusable, distributable, configurable, fast enough, secure, compatible, and so on. To obtain this, the program must be given a good architecture, which is much more work.

Purpose

The purpose of this problem is to familiarize you with common architectural principles and patterns used in the design of software, such as *cohesion*, *separation of concerns*, and *layering*. You will exercise this by recognising problems in an existing source. Another purpose is to give you hands-on experience with defining architectural views within UML tools, and with reverse engineering features.

Our Case – an Asteroids Game from the Internet

We will consider an implementation of the computer game classic Asteroids, found on the internet. (Asteroids.zip on Canvas).



By playing this game we see that the game is fully functional. The game was originally a Java Applet.Normally you could run the game by opening the index.html file in a browser (after unzipping Asteroids.zip), but since most browsers disabled java applets by default this is not directly possible anymore. The source is modified and now it is a Java Application.

Looking at the program's source (Asteroids.java), we see that the program does not have a nice architecture at all: application logic and GUI rendering is intertwined, only two classes are used (even though the domain contains asteroids, ships, bullets, scores and so on), all methods and attributes are public.

Problem

Your problem is to improve the the architecture of the asteroids game.

Requirements on the solution - see also 'reporting':

- The program should be functional, i.e., it should be possible to play the game.
    - First import the source code project into Eclipse and compile/run it before modifying the code.
- The architecture should be represented as a UML model from a tool that includes one or more class diagrams.
- The architecture should be layered (use packages) and include a UI layer, an Application Layer, a Domain Layer, a foundation layer and a technical services layer.
- Classes in the models should be mapped to these layers in a sensible way. Depending on how much your design improved the original it could be that not all layers are covered.

- The improved **design** should have the following improvements:
    - classes for UFOs, photons, and other objects recognizable in the files Asteroids.java.
- The improved **program** should have the following improvements:
    - An implementation of one of the proposed improvements (at least 3 new classes).
- The solution should at least **discuss** a proposal for the use of one appropriate software design pattern that was discussed in the lecture (so, no implementation)

***Optional**:*

- *The solution should at least **use** (so implemented in code) one appropriate software design pattern that was discussed in the lecture*

Hints on approach

- There are several ways to restructure the program: (1) start with a new model in your UML tool, and add bits and pieces from the old program as you build the new program; (2) read in the program in the UML tool and transform the model.
- Print out the imported (reversed) model and **group/highlight responsibilities** **(see also reporting)**
- Search for 'bad smells' in the code and class model with the help of https://www.openlearning.com/courses/COMP461394/LectureNotes/Week2.0/badSmells.pdf?action=download
- Create a small domain model of the program, not looking at the code.
- Print out the program on paper, cut it in pieces, sort the bits in related groups of functionality.
- Successively rewrite your program towards the desired goal in small steps, while testing that your program still runs.
- You may further simplify some components by adding Statecharts.

Practicalities

- Models are read into the UML tools using the tools 'Reverse Engineering' feature of Visual Paradigm (or any other tool that can do this)
- In the code, Attributes and Operations can be moved between Classes in the explorer of Eclipse.

Reporting your solution to the problem

Report your solution by submitting a .zip file in the assignment area of Canvas.

The hand in should include

- A document (doc(x),odt, or pdf) containing
    - a (reversed engineered) class diagram of the original source
    - a page containing the responsibility analysis (highlighted responsibilities)
    - a clear explanation of the problems (you should at least identify 3 major problems) that were found and what design solutions were chosen.
    - diagrams for your new architecture/design (at least a class diagram with layers and newly organized classes)
- The .java files for your new program (one file per class).

Deadline: Sunday 2nd of June 23:59