Improving Software Architecture

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 used in the design of software, such as *cohesion*, *separation of concerns*, and *layering*. 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. (File Asteroids.zip on GUL).

By playing this game (which we may do by unzipping Asteroids.zip and starting the game from Asteroids.html) we see that the game is fully functional. 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 (for a G):
- The program should be functional, i.e., it should be possible to play the game.
- The architecture should be represented as a UML model from a tool that includes one or more class diagrams.

- The program should have classes for UFOs, photons, and other objects recognizable in the files Asteroids.java.

Additional requirements for a VG:

- The architecture should be layered 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.

Hints

- 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.
- 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 you 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.
- UML tools you can use are : Papyrus, Visual Paradigm
    - other tools that can do reverse engineering and code generation are allowed, but only after agreement with the supervisor or lecturer.
- Attributes and Operations can be moved between Classes in the explorer

Reporting your solution to the problem

Report your group's solution by submitting a .zip file in the assignment area of GUL. The file-name should be on the form *G1-groupnameP6.zip* for group 1, etc. The hand in should include

- Diagrams for your architecture (in the form of a .rtf or a .jpeg files).
- The .java files for your new program (one file per class).

The presentation should include a title page listing everyone that actively contributed to the hand-in.

Deadline: P6 - Fredag 9th of March 23:59.