HW3 – Reducing Complexity with Appropriate Use Design Patterns

Estimated time: 24-30 hours

Objectives

- Become more familiar with UML modeling
- Become familiar with applying the Singleton, Factory, Flyweight, Command, and Undo patterns
- Improve unit testing skills to reduce complexity and improve performance
- Strengthen programming skills in area of user interfaces

Overview

In this assignment, you will build a simply drawing program that allows a user to paint a scheme with instances of different kinds of objects. You can choose what kind of drawing program to create and what kinds of objects to have in the user's palette. For example, you might decide to develop a program is for drawing painting, and therefore give the user some pre-defined high-level objects, like a mountain, a tree, a house, a cloud, etc. If your program is for drawing cartoon characters, you might want to include a head, eyes, noses, ears, mouths, etc. If your program is for drawing UML class diagrams, you would want your palette to include class symbols, binary associations, specializations, etc. The objects represented in the palette can be complex objects with significant amounts of intrinsic state information and some extrinsic state information.

Functional Requirements

1. The user should be able to create a new drawing at any time, through either a menu option or a hotkey.
   1.1. On creation, the user should be able to select an initial background color or image that fills the entire drawing space.
2. The user should able to create instance of object types from a palette and place them on the drawing. You are free to decide what drawings your program with make and therefore what kinds of objects your program will support.
   2.1. Your program must allow the user to instantiate and places at least six different kinds of objects on drawing.
3. The user should be able to select an existing object on the drawing and perform an action on that object.
   3.1. The user should be able to move the selected object on the drawing
   3.2. The user should be able to remove the selected object from the drawing
   3.3. The user should be able to scale the object

   3.4. If applicable, the user should be able to change other properties of the object, like its line color or fill color
   3.5. The user could be able to duplicate a selected object
4. The UI should allow multiple gestures (keystrokes, mouse movements, and button clicks) for some of the actions.
5. The system should keep a history of actions (commands) that the user performs to since the drawing was created or last opened.
6. The user must be able to undo previous drawing actions in reverse order, all the way back to the initial drawing creation or opening of the drawing.
7. The user should be able to save a drawing in a persistent store and re-open it later
   7.1. A saved drawing should be a set of objects (with their state information), not a bitmap
   7.2. The persistent store should be either a database in a cloud (e.g., the AWS RDB services) or a serialized object in a cloud (e.g., the AWS S3 services)
8. The user should be able to open a saved drawing.

Instructions

To build this system, you will need to do the following:

1. Decide what you'd like your drawing program to be for, e.g. landscapes, people, blueprints, a type of UML diagram, or something else of interest to you.
2. Design your system. Look for **good** opportunities to apply the singleton, factory, flyweight, command, and undo patterns. Also, don't forget about the other patterns that you've learned, like strategy, decorator, and observer. Use the patterns to help manage the inherent complexity of the problem and eliminate accidental complexity.
   2.1. Be sure to separate your GUI Layer from your Application Logic Layer. The classes that represent your drawing or the objects in your drawing should in your Application Logic layer and should not need to depend on your GUI components.
3. Implement and test the classes in your Application Logic Layer. Your unit tests for the components in this layer must be executable and thorough
4. Implement your GUI
5. Test your software at the system level using ad hoc testing methods.

Submission Instructions

Zip up your entire solution, including test cases and sample input files, in an archive file called CS5700_hw3_<*fullname*>.zip, where fullname is your first and last names. Then, submit the zip file to the Canvas system.

Grading Criteria

Criteria	**Max Points**
A clear and concise designs consisting of UML class, interaction, and state diagrams that describe your system well enough that another developer could implement it from your diagrams.	20
A working implement, with good encapsulation, abstractions, inheritance, and aggregation; readable code; low coupling and high cohesion.	30
“Good” application of design patterns in a way that helped manage complexity and ensure quality; no obvious missed opportunities; and no forced or awkward uses of design patterns.	30
Thorough executable unit test cases for core application logic	30
Reasonable systems test using ad hoc methods	10