Final Project

Introduction

The Final Project gives you the opportunity to apply the object-oriented software development skills you have acquired during the course to a **significant** but **moderately-sized** application. The following sections describe the general behavior of the application and the deliverable requirements for the project.

Starter code can be found on D2L > Content > Final Project.

Overview

We will be building an “MS Paint”-like application in Java called JPaint. It will have the following features:

- Pick a shape
  - Ellipse
  - Triangle
  - Rectangle
- Pick a primary color
- Pick a secondary color
- Click + drag to draw a shape
- Be able to change color after it’s drawn
- Select shading type (outline only, filled-in, outline and filled-in)
- Select shapes
- Click and drag to select shapes
- Right click to draw with secondary color (flip primary and secondary color)
- Copy
- Paste
- Delete shape
- Undo
- Redo
- Move shape
- Support keyboard shortcuts
  - Ctrl + C (Copy)
  - Ctrl + V (Paste)
  - Delete or backspace (Delete)
  - Ctrl + Z (Undo)

    - Ctrl + Y (Redo)

I am providing the GUI. You should need to write minimal GUI code. You will need to write code for drawing shapes, which may involve some bits of UI code. You will also need to write some handlers for click and drag events.

Behavior

Pick a shape.

- Selecting ellipse, rectangle, or triangle will cause a click and drag event on the canvas to draw that shape.

Pick a primary color. You will need go get the primary color from the UI.

Pick a secondary color. You will need go get the secondary color from the UI.

Select shading type (outline only, filled in, outline and filled in).

- Outline Only – Only shape outline will be drawn. Use Primary Color to draw this.
- Filled-In – Only the inside of the shape will be drawn – there will be no visible outline. Use Primary Color to draw this.
- Outline and Filled-In – Both the inside and the outline will be drawn. Use Primary Color for the inside and Secondary Color for the outline.

Draw Mode

- Click + drag while in Draw Mode to draw a shape – click on the canvas and drag. This will draw the selected shape within those boundaries.
- Right click and drag to draw a shape with flipped primary and secondary colors – see the Select Shading Type section above and replace all “Secondary” with “Primary” and vice versa.

Select Mode

- Click a shape (while in Pointer mode) to select it.
- Click and drag to select multiple shapes within those boundaries.
- Click canvas to deselect.
- Selected shape(s)
    - Be able to change primary and secondary colors
    - Change shading type (outline only, filled-in, outline and filled-in)

Move Mode

- Clicking and dragging moves the selected shape(s).

Copy

- Copies the selected shape(s)

Paste

- If there is anything copied, paste it on the screen somewhere. You can paste it at origin (0, 0), the middle of the canvas, or somewhere else that makes sense.

Undo

- Undo the last operation. Keep in mind, some operations do not need to be undo-able – for instance, “Copy” probably doesn’t need to be undoable (though “paste” probably should).

Redo

- Redo the last operation. See undo.

Delete

- Deletes the selected shape(s)

Support keyboard shortcuts – you can substitute for Mac-specific key combos if you’re developing on Mac.

- Ctrl + C (Copy)
- Ctrl + V (Paste)
- Delete or backspace (Delete)
- Ctrl + Z (Undo)
- Ctrl + Y (Redo)

Extra Credit

Add a CUI (Console User Interface). It should support the same functionality as above. Displaying shapes will show the (unique) ID of the shape and the dimensions/colors. Add the ability to save to a file and read from a file. Be able to open a file that you created from the CUI in the GUI.

Suggestions and Hints

Use D2L to talk to your fellow students, bounce ideas off each other, etc. Collaboration is encouraged! However, the code you turn in must be your own.

Unit tests will be your best friend. If you post on D2L asking for help in figuring out why something isn't working, I will advise you to write a few unit tests around your method so you can verify it's doing exactly what you want. The benefit is this will help you fulfil your Unit Test requirement.

Use Dependency Injection and follow SOLID practices! This will help you keep your code manageable, make it easier to unit test, and allow you to refactor and replace modules more easily.

Use Source Control and an IDE! Source Control will help you go back in time when you realized you really screwed up and need to undo. Check in frequently. An IDE is an incredibly useful tool for refactoring. You can extract code into a method or rename a file and all references very easily. It will also help you organize your classes and interfaces.

Run your build and unit tests frequently. The sooner you detect that a unit test is failing, the easier it is to pinpoint which change you made that broke it.

Don't procrastinate. ☺

Code Requirements

1. The application must be written in Java using the Java2 SDK 1.10 or higher.
2. Only features and capabilities that are part of the Java2 SDK may be used in the application. *No third-party software* such as BlueJay or JBuilder class libraries or COM/CORBA components.
3. You must write unit tests for at least three classes that have some reasonable amount of behavior (e.g. NOT classes with simple getters and setters), but you are encouraged to write more!!!
4. The application must use at least ~~five~~ four different design patterns that we have discussed in class. You will be expected to demonstrate and explain the patterns in your final written description.
5. You are STRONGLY ENCOURAGED to use an IDE (Eclipse, IntelliJ IDEA or something else) and Source Control. The IDE will help you immensely with refactoring and running unit tests, and Source Control will help you revert changes as needed and avoid catastrophe.

Report

Thirty points of your grade is based is based on a written report. Although no specific style guidelines are being enforced, the report must be presented in a neat, legible, and consistent format.

All text must be typed. Diagrams must be drawn on the computer using a program like draw.io or Visio. **Do not use diagrams generated from your IDE.** They never come out right and if I can't read the diagrams, I will take off points. The diagrams should conform to the UML notational conventions presented in class.

The written report should be structured as follows:

1. **List of missing features, bugs, extra credit, and miscellaneous notes.** List features that **don't** work and make me aware of any bugs in the code. List any extra credit or any other miscellaneous notes as well. Essentially, list anything I should know when grading.
2. **Notes on design**. Indicate ~~five~~ four design patterns used in your project. There should be some variety in patterns you discuss – you can't discuss the same pattern more than twice.

   For each pattern, in a **few sentences**, describe the classes involved, why you chose to implement that pattern, and what problem it solved. Include a design class diagram as well. Be sure to include all significant class relationships: realization, specialization, and association. Show associations as dependencies, aggregations or compositions when appropriate. Show attributes and methods *only* if they are crucial to understanding the class relations (e.g. for Dependency relationships). *You do not need to show all fields and methods!*

   These notes/diagrams should take approximately 2-3 pages.

3. **Successes and Failures.** Discuss what went right with your project? What went wrong? Note design issues that arose during development, such as specific decisions, use of design patterns, failures, successes, etc. This should take 1 page or less.

Deliverables

1. Week 10 (11/12/2018): Final release.

Submit a `jar/zip/rar/7z` file containing your src folder and writeup.

Test your `jar/zip/rar/7z` file by downloading it from D2L and unzipping it into a fresh directory. Make sure everything works. **IF WHAT YOU TURN IN DOESN'T COMPILE, YOU WILL GET A 0%!**

You will submit your project deliverables on D2L. Your report must be a *single* file in PDF or DOC format. This requires that you somehow get diagrams into the report. Microsoft word and OpenOffice are very good at this. Please use one of them. **IF I CAN'T OPEN YOUR REPORT, YOU WILL GET A 0%.**

Grading

Grading will follow these guidelines:

- **Application: 70 points**
  - Satisfies requirements
    - Correctly implemented design patterns for at least five classes.
    - Complete and quality unit tests for at least three classes.
  - Correctness (compilation & execution)
  - Quality of code – follow the SOLID principles!
- **Report: 30 points**

Academic integrity

I do encourage collaboration; however, all submitted work must be your own. If the work was duplicated, it will be reported to the university as an Academic Integrity violation. I don't mind copying and pasting of utility functions.

As general rules of thumb, I would say the following guidelines dictate whether sharing is allowed or not:

- It is functionality you would reasonably expect to find on Stack Overflow (like how to draw a shape)
- It is some sort of mathematical algorithm (like determining the math for drawing a triangle).
- It is an abstract description of how your code is working (like "I used the factory pattern to create the individual shapes in x class").

I want to avoid being heavy-handed about this. I would prefer you collaborate than not due to being afraid to violate these rules. When in doubt about whether you can share something or not, send me an email and ask. In 99% of cases where you aren’t sure, I will likely say it’s acceptable.