SE2811 Half-exam 1 Name:

This is a 25-minute, 100-point half-exam. The exam is printed double sided. If you use a note-sheet (prepared by yourself), please turn it in with your exam.

1. (10 points) ***Write*** the notifyObservers() method for a Subject class in the Observer pattern. You do not need to use exact names for other methods and variables. Assume push to observers rather than observer pull.

```
public void notifyObservers(double position) {




}
```

2. (10 points) Consider an application with a single Car class containing both the code for managing observers and for modeling the motion of the car. Now imagine that that code is separated into two classes: An abstract class for managing the observers, and a concrete class for modeling the motion of the car. ***Circle one*** option below and ***explain*** how that term describes the improvement in the program when moving to the second version.
   a. Increases coupling
   b. Decreases coupling
   c. Increases cohesion
   d. Decreases cohesion

3. (10 points) ***Re-write*** the code

```
        SwingUtilities.invokeLater(()->label.setText("Complete"));
```

   to use the full anonymous inner class syntax instead of a lambda expression. Recall that invokeLater takes a `Runnable` as an argument, and `Runnable` has a single method `run()`.

```
SwingUtilities.invokeLater(




);
```

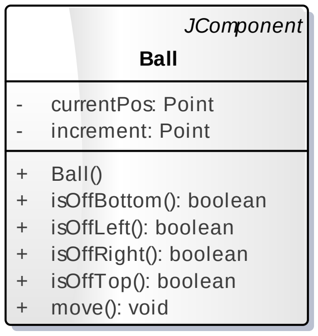
4. (10 points) ***Explain*** why `SwingUtilities.invokeLater` must be used instead of a `synchronized` block in a multithreaded GUI application.

5. (10 points) You are working on a (physical) architecture program that allows the user to design a building, and view it in multiple ways – floor plan, photo-realistic, floor plan, electrical wiring, etc. If the user moves a wall in the view floor plan, that wall should to move in the other views as well. ***Circle one*** pattern and ***explain*** how it allows you to reduce the coupling between the floor plan and the other views.

   a. Strategy c. Singleton
   b. Factory Method d. Observer

6. (40 points) Consider both the UML diagram and existing code. ***Revise*** your parts to this problem to be consistent with each other. We desire to extend the Pinball machine to incorporate different kinds of gravity. Each ball will experience different gravity. Some may fall down. Others may fall to the right. Others may be attracted to some "planet" on the screen. Use the strategy pattern to rewrite this code.

   a. ***Rewrite*** the UML diagram to allow balls that fall down or to the right. Include any methods that you use in part b in the diagram.



   b. ***Edit*** the method below to encapsulate what varies.

```
public void move() {

        if(isOffBottom() || isOffTop()) {
                increment.y = -increment.y; // bounce off bottom
        }
        if(isOffLeft() || isOffRight()) {
                increment.x = -increment.x; // bounce off side
        }

        currentPos.x += increment.x; // increment the position of the duck
        currentPos.y += increment.y;

        increment.y++; // acceleration downwards.

        setLocation(currentPos); // ...and update its location in the container

}
```