SE2821 Midterm Exam

You may use an 8.5x11 note-sheet, printed on both sides. Review all questions before you get started. Show all work.

Student's Name: _____

I. Design Patterns (30 pts)
II. Strategy Pattern (15 pts)
III. Factory Pattern (15 pts)
IV. Singleton Pattern (10 pts)
V. Observer Pattern (15 pts)
VI. Multithreading (15 pts)

I. Design Patterns [30 pts]

1. (10 pts) Select the pattern that best meets the given application. If two seem equally close, ask for clarification.
    a. You are designing a video game involving multiple characters. Every time a character moves, you want the other characters to adjust their strategies. Which pattern would you use to keep the other characters informed?
        i. Strategy
        ii. Factory Method
        iii. Singleton
        iv. Observer
    b. Java's Swing and AWT APIs make use of an EventListener. This is part of which pattern?
        i. Strategy
        ii. Factory Method
        iii. Singleton
        iv. Observer
    c. In a game, you have a variety of characters (e.g. king, dwarf, hobbit, etc.) that perform a variety of different attacks (e.g. swing sword, throw lasso, shoot bow, etc.) but only one should happen when the player presses the "attack" button.
        i. Strategy
        ii. Factory Method
        iii. Singleton
        iv. Observer
    d. You would like a centralized logging location that all parts of your program can access.
        i. Strategy
        ii. Factory Method
        iii. Singleton
        iv. Observer
    e. In an email application, you would like to support future encryption algorithms without having to go back to change your "encrypt" method every time a new and improved encryption algorithm comes along.
        i. Strategy
        ii. Factory Method
        iii. Singleton
        iv. Observer

2. (15 pts.) For each of the following, explain why you would use the given pattern
    a. In the weather application used in lab, the Observer pattern

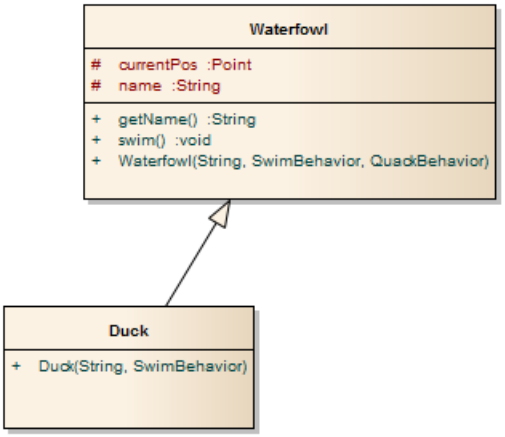
    b. In the weather application used in lab, the Singleton pattern

    c. In the Bee application, the Strategy pattern

3. (5 pts.) Briefly define the cohesion and coupling. Make sure your definitions distinguish between them.

II. [15 pts.] The Strategy Pattern

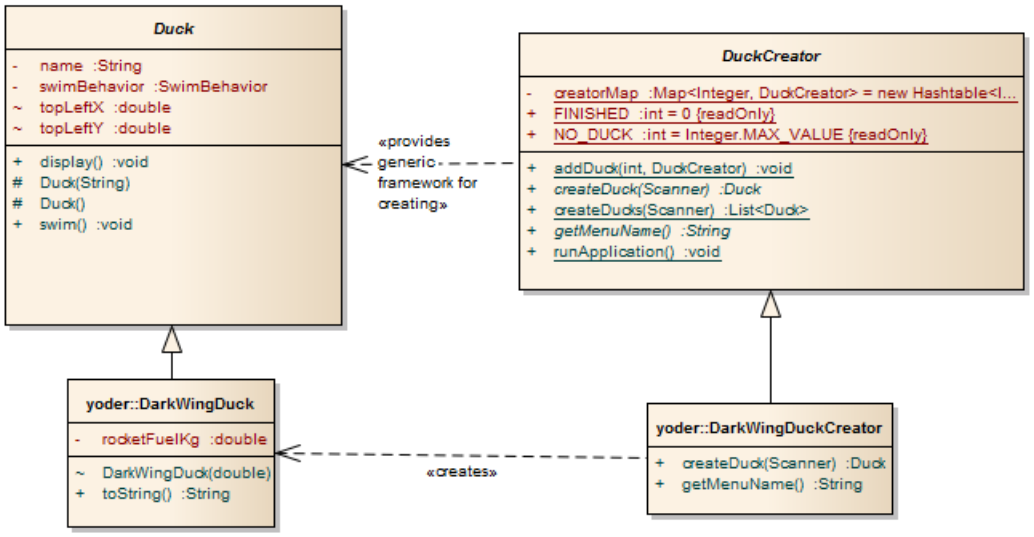
1. (10 pts.) Complete the following diagram to include a Circular Swimming behavior, using the strategy pattern to allow other behaviors to be stuck in later.



2. (5 pts.) Write the swim() method of the Waterfowl class based on your diagram above

III. [15 pts.] Factory Method Pattern

In the following UML diagram, as discussed in class, the addDuck method registers a new DuckCreator with the application, the the createDucks method interactively adds different duck types, and the *createDuck* method adds a single duck type.



1. (15 pt.) Implement the createDuck method of the DarkWingDuck Factory. The name of the "double" that the DarkWingDuck constructor takes is "fuelKg".

IV. [10 pts.] Singleton

1. (10 pts.) Complete this DatabaseConnection class demonstrating the Singleton pattern. The constructor does not need to do anything.

```
public class DatabaseConnection {

	public static synchronized DatabaseConnection getInstance() {
		if(uniqueInstance==null) {
			uniqueInstance=new DatabaseConnection();
		}
	}
}
```

V. [15 pts.] Observer

1. (15 pt.) Write a concrete implementation of the notifyObservers method in a class implementing a Subject interface in the observer pattern. You may reference instance variables as needed.

VI. (15 pts.) Multithreading

1. (3 pts.) Name one way to <u>implicitly</u> create a thread in Java (i.e. without the "new" keyword.)

2. (2 pts.) Which of the following is not shared between threads?
    a. a local variable
    b. an instance variable
    c. a class variable
    d. a named constant

3. (10 pt.) The GUI below crashes while setting up the squares because the event dispatch thread is already trying to use the squares while `setUpSquares()` is still running. Demonstrate how to use `SwingUtilities.invokeLater(…)` to resolve the problem. You may cross out code if desired.

```
public class GameFrame extends JFrame {
    public static void main(String[] args) {
        new GameFrame();
    }

    private GameFrame() {
        super("Game Frame");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);









        setUpSquares();









    }
// …
}
```