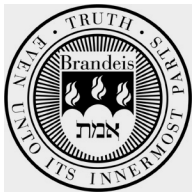

COMPUTER SCIENCE 12B (SPRING TERM, 2017)
PROGRAMMING IN JAVA

PROGRAMMING ASSIGNMENT 3

Due: Monday March 6th at 11:55pm

Overview:

This assignment will give you practice with inheritance, interfaces, and abstract classes.

Part 2: Interfaces and Abstract Classes

You are to write an inheritance hierarchy that defines the behavior of birds. You will be given a program that runs an aviary simulation of a world with many birds flying around in it.

For this assignment you will be given supporting code that runs the simulation. You are just defining the individual birds that wander around this world. This program will probably be confusing at first because this is the first time where we are giving you a specific client program. Your code will not be in control of the whole application. Instead, you are defining a series of objects that become part of a larger system. For example, you might find that you want to have one of your birds make several moves all at once. You won't be able to do that. Although this experience can be frustrating, it is a good introduction to the kind of programming we do with objects. A typical Java program involves many different interacting objects that are each a small part of a much larger system.

Each bird has a (x,y) position and a color, and each bird can fly. Different kinds of birds will fly in different ways and you are defining those differences. We are dividing this assignment into tasks to assist you with the design:

1. Your first task is to write an interface called `Bird` to represent different types of birds. The interface should have methods to do the following:

1. Get the bird's color.

2. Get the bird's (x,y) position as a `Point`.

3. Tell a bird to fly. Each time a bird is told to fly, it will move its position once.

Your `Bird` interface should work with the `Aviary` client program we provide (you can download it from LATTE). Based on the `Aviary` program you should figure out yourselves the correct names and return types for your `Bird` methods. The `Aviary` client program is using the `DrawingPanel` class that is also available on LATTE. **All .java files we provide should be placed in the same folder with your own source files.**

We are also providing an interface `AviaryConstants` (also available on LATTE) that includes two static constants needed for creating the simulation window for your flying birds. You will probably use one or both them in your own classes.

```
public interface AviaryConstants {

	// the size of the Aviary, x and y coordinates run from 0 to 19
	public static final int SIZE=20;

	// the number of pixels corresponding on one x and y coordinate unit
	// i.e., you aviary areas has a size of 200x200 pixels
	public static final int PIXELS=10;

}
```

2. If some of the methods of your interfaces are the same for all birds, you should implement an abstract class (e.g., `AbstractBird`) to hold the behavior that will be common to all bird classes. Also, if there are some fields common to all birds (e.g., color, etc) , you should place them in this abstract class. Please justify your choice in the readme.txt file.

You should explain in your readme file why you choose to have or not to have an abstract class.

3. Write a class called `Cardinal` that represents cardinals. A cardinal is red in color. The cardinal's movement is vertical. Initially a cardinal is moving up. Each time the cardinal is told to fly, it will move its position one unit upward on the y-axis (remember that upward is negative – see Hint 2). If the cardinal hits the edge of the aviary (a y-coordinate of 0 or 19), it turns around and flies in the opposite direction. Remember that the size of your aviary is 20 and is stored as a static constant in the `AviaryConstants` interface.

4. Write a class called `Hummingbird` that represents hummingbirds. A hummingbird is magenta in color. Its movement is random: each time the hummingbird is told to fly, it will pick a new random (x,y) position in the range of (0,0) to (19,19).

For the random moves, you may use either a `Random` object or the `Math.random()` method to obtain pseudorandom values.

5. Write a class called `Bluebird` that represents bluebirds. A bluebird is blue in color. Its movement is in a zig-zag pattern. Initially the bluebird faces right. The bluebird moves in an alternative pattern of up-right, down-right, up-right, down-right, and so on until it hits the right edge of aviary (x-coordinate of 19), at which point it turns around. Subsequent calls to fly will cause the bird to move up-left, down-left, up-left, down-left, and so on until it hits the left edge of the aviary. **In other words the Bluebird always alternates between up/down in the y-axis**.

6. Write a class name `Vulture` that represents vultures. A vulture is black in color. Its movement is in a counter-clockwise circle pattern. Initially the vulture faces up. The first time it flies it moves up by one, then it turns to face left. Its second move, it moves left by one and turns to face down. Its third move, it moves down and turns to face right. Its fourth move, it moves right by one and turns to face up. The pattern repeats in this fashion.

Hint 1: To represent the color of each bird you can use the `Color` class of the `java.awt` package. Each color (`BLACK`, `RED`, `WHITE`, `YELLOW`) has an explicit value which you can get it with the statement `Color.<name of color>`. E.g., `Color.BLACK` returns the value for black. You should look up the `Color` class on the Java API to see the full list of available colors and decide which one you need to use.

Hint 2: In graphics upwards direction is negative. This means that the origin point (0,0) is at the top-left corner of your screen and the y-coordinates increase as you move towards the bottom of your screen. So, if you want a point to move upward you have to reduce its y-coordinate.

Running your program: To test if your classes work you should run the `Aviary` class. You may change (for testing/debugging purposes) the `Aviary.java` client program but keep in mind that we will test your classes with the `Aviary.java` program we gave you, so you should make sure your classes work with our version.

Stylistic Guidelines:

You need to apply most (if not all) the concepts we've been discussing so far regarding inheritance and interfaces/abstract classes (using the `super` keyword, figuring out which fields should go in the abstract class and which in each subsclass, which methods belong in the interface and/or abstract class and which need to be implemented/overridden, avoid redundant code, proper usage of static constants, field and method encapsulation.)

We will be graded on your appropriate definition of hierarchies, the correct usage of method overriding, constructors, and proper interaction with the superclass.

You should keep in mind the ideas we have been stressing in class so far. You do not want to have redundant code. You should properly encapsulate your classes, use constants when appropriate to avoid hard-coded numbers or hand-coded strings in your code. The choice of what constants to use is up to you, but they should represent important values that are used in your classes.

Follow general stylistic guidelines, such as indentation and whitespace, meaningful identifier names, and localization of variables.

Include a comment at the beginning of your program with basic information and a description of the program and include a comment at the start of each method. Also put brief comments inside longer methods explaining the more complex sections of code.

Submission:

Your Java source code should be submitted via Latte. For late policy check the syllabus.

Below is the list of files you should submit. : `Aviary.java`, `AviaryConstants.java`, `DrawingPanel.java`, `Bird.java`, `Cardinal.java`, `Hummingbird.java`, `Bluebird.java`, `Vulture.java`, `readmepart2.txt` and the abstract class (if you have one). The file `readmepart2.txt` should include your name, the date and the homework number and describe what the program of part1 is designed to do, and what it actually does.

Grading:

You will be graded on

- **External Correctness:** The output of your program should match exactly what is expected. Programs that do not compile will not receive points for external correctness.
- **Internal Correctness:** Your source code should follow the stylistic guidelines shown in class. Also, remember to include the comment header at the beginning of your program.
- **One-on-one interactive grading:** By the end of the day that the assignment is due, please make an appointment with your TA for an interactive 10-15 minute grading session. You will receive an email notifying you of the name of the TA who has been assigned to you for this assignment with further instructions on setting up the appointment. (You will be meeting with a different TA for each assignment). One-on-one interactive grading will help you improve your programming skills and avoid repeating mistakes from one assignment to the next.