Assignment 1 – review java & OOP

Comp310 – Object oriented data structures

# Topic(s)

* Primitive types
* Classes, Objects and Methods
* Inheritance, Polymorphism, Information Hiding
* Generic & Abstract classes

# Readings

**Carrano:** Appendix A - D

# Learning Objective

Review concepts of *object oriented programming* and syntax specific to applying those concepts in Java.

# instructions

1. **Select a partner** (strongly recommended for brainstorming design and implementation options/solutions). Please let instructor know **BEFORE YOU START** who you will work with.
2. Read over the assignment and ASK QUESTIONS about anything that you don’t understand (before you start).
3. Implement one of the [problem options](#_Problems) described below or propose your own problem (to the instructor).
4. Be sure to follow [Good Programming Practices](#_Good_programming_practices), also listed below.
5. Individually answer the [Analysis and Feedback](#_Analysis_and_Feedback:) questions as a Journal entry on blackboard.
6. Follow instructions for [submitting your work](#_Submitting_your_work:).
7. **Demo your project to instructor** (required for grading).

# problem Options

In the following problem descriptions, the fundamental requirements are included; however, a lot of the implementation details and decisions are left to the student ***(to allow creative freedom).***

## Regions

Assume you’re developing the building blocks for different geo-political regions (cities, states, countries). There are atomic regions (i.e., cities) which can’t be broken into smaller regions, and Container regions (i.e., states, countries), which are built from contained regions. For atomic regions, population and area statistics are stored directly, but for container regions they are calculated from the values in the contained regions.

Below are skeleton UML diagrams of the classes *(i.e., not all methods, instance variables, constructors that you will need are included).*

**Region (abstract)**

String name;

Abstract int getPopulation()

Abstract double getArea()

**. . .**

**Is a**

**Is a**

**ContainerRegion**

Region[] region; // array of Regions

**. . .**

int getPopulation(..); // loop through regions and call their getPopulation methods

**. . .**

**AtomicRegion**

int population; // set in the class bythe constructor

double area; // set in the class by the constructor

**. . .**

int getPopulation(); // return value in instance variable

double getArea() // return value in instance variable “area”

**. . .**

**Is a**

**Is a**

**Is a**

**State**

**Country**

**City**

**Getting started creating the classes and order of implementation:**

1. Create the abstract **Region** class (see skeleton UML diagram above)
   1. Add instance variable name
   2. Include any necessary constructors, accessor and mutator methods.
   3. (can’t create an instance, so cant test).
2. Create the class **AtomicRegion** and extend it from Region (the “base” class)
   1. Add instance variables common to all AtomicRegions (population, area, others?)
   2. Include any necessary constructors, accessor and mutator methods.
   3. (test by creating an instance of this class before you proceed)
3. Create the class **City** extended from AtomicRegion
   1. Add instance variables specific to cities (i.e., mayor)
   2. Include any necessary constructors, accessor and mutator methods.
   3. (test by creating an instance of this class before you proceed)
4. Create the class **ContainerRegion** extended from Region
   1. Add an array of regions (see UML skeleton above).
   2. Include any necessary constructors, accessor and mutator methods.
5. Create the class **State** extended from CountainerRegion
   1. Add instance variables specific to States (i.e., governor)

This is where students get confused, so please ask if you need help.

* 1. Include any necessary constructors, accessor and mutator methods.
  2. **Fill the Region array with objects of type City (remember, Cities are also Regions)**
  3. Test before you proceed

1. Create the class Country extended from CountainerRegion
   1. Add field’s specific to Countries
   2. Include any necessary constructors, accessor and mutator methods.
   3. Fill the Region array with objects of type State. (if you have gotten State to work, above, then you will have no trouble with this one).
   4. Do all your final tests.

## Avatars

**Notes:** Do this assignment only if you either already understand GUI development or if you are motivated to learn it (relatively independently).

**Assume you’re creating the building blocks to allow a user to create simple 2D Avatars.** Though these rough UML skeletons are provided, feel free to use a little creative license, as long as you include: abstract classes & methods and dynamic binding (with the draw methods).

**BodyPart (abstract)**

String name;

Abstract void draw( );

. . .

Is a

**ContainerPart**

BodyPart[] part; // array of parts (container & drawable)

. . .

void draw(..); // loop through parts …

**DrawablePart**

Coordinate[] coord; // array of coordinates

. . .

void draw(..); // Draw using coordinates

Avatar

**. . . (parts that contain other parts)**

Head

**. . . (parts without sub-parts)**

1. Create an abstract class BodyPart
   1. Add instance variables common to all body parts, including the 2D location (i.e., x and y coordinate), name and color
   2. Add an abstract draw method.
2. Derive class DrawablePart from BodyPart.
   1. Contain an array of coordinate points for drawing this part.
   2. The draw method should display the part.
3. Derive classes from DrawablePart that are primitive parts (i.e., don’t have sub-parts).
   1. For example, you might create and draw hands as a unit, so you should have a Hand class. *It’s up to you how you want to divide your avatars.*
   2. Add a main method and test before you proceed.
4. Derive ContainerPart from BodyPart
   1. Contains an array of type BodyPart.
   2. The draw method should loop through the array of BodyPart and calls its’ draw method.
5. Derive classes from ContainerPart that represent “container parts” (i.e., UpperBody could contain Torso
   1. The Avatar, itself, of course – otherwise, the number and types of container parts depends on how detailed you want to be.
   2. Add a main method to each class as you write it

# Good programming practices

**For all classes**

* Make your instance variables **private**
* Include **constructors** to initialize your instance variables.
* Derived class constructors should **leave initialization of super class instance variables** to the super classes’ constructors:
  + Remember the call to the super classes constructor is: **super( <init1>, <init2>,..)**
* Include **Accessor** and **Mutator** methods for all instance variables *(please ask if you’re not sure what these are)*
* Include a **main** method for testing (unless it’s an abstract class) and test before you proceed
* Add comments to your code, not just so it’s easier for other readers, but also so it’s easier for you to remember your logic.

# Analysis and Feedback

**In your Journal on Blackboard, answer the following questions:**

1. Why couldn’t you add a main method to the abstract class (Region or BodyPart) to test before proceeding?
2. Could we have added Generics to this assignment? If so, how? If not, what would have been a better example for adding generics?
3. Where did you have trouble? How did you move forward? What topics still confuse you?

Remember that thoroughness, thoughtfulness and creativity *(not correctness)* should be your focus. Even if you work in pairs, each student should answer these questions individually:

1. What did you learn from this assignment? *(Please be specific)*
2. Any suggestions for improving this assignment in the future?
3. If you worked in a pair, how did you divide up the work?

# Submitting your work

1. Make sure that your name(s) are in all of your project files. *(Note: for students who work in pairs, each must include a summary file with just their name on it)*
2. If you have more than one file for your solution, make a .zip file for your project
3. In Blackboard, attach your solution file to the submission for this assignment.