Name: *(as it would appear on official course roster)*	
Umail address: **@umail.ucsb.edu**	**section** **9:30am or 11am**
Optional: name you wish to be called if different from name above.	
Optional: name of "homework buddy" (leaving this blank signifies "I worked alone"	

1

h10: The Decorator Pattern (HFDP 3, part 2), and the Open-Closed Principle

h10

CS56 M18

ready?	assigned	due	points
true	Fri 08/31 12:20PM	Thu 09/07 12:20PM	100

You may collaborate on this homework with AT MOST one person, an optional "homework buddy".

MAY ONLY BE TURNED IN IN THE LECTURE/LAB LISTED ABOVE AS THE DUE DATE,
OR IF APPLICABLE, SUBMITTED ON GRADESCOPE. There is NO MAKEUP for missed assignments;
in place of that, we drop the three lowest scores (if you have zeros, those are the three lowest scores.)

This homework, and the preceding one, H09, both concern HFDP Chapter 3. Please read that chapter, and then answer these questions.

If you didn't yet, please also read this short blog post by Robert C. Martin (Uncle Bob), author of the popular book "Clean Code", and "Clean Coder". I'll ask a question about this article on this homewwork assignment.

- https://8thlight.com/blog/uncle-bob/2014/05/12/TheOpenClosedPrinciple.html

1. (10 pts) Please fill in the information at the top of this homework sheet, including your name and umail address. Put the time your discussion section starts (9:30am or 11am) in the space indicated (the one you are registered for—even if you usually attend a different one.) If the other two items apply, please fill them in as well. Please do this every single time you submit homework for this class.

2. (10 pts) Explain the connection between the Open-Closed principle, and the idea of a "plug-in" for various software products such as IDEs (Eclipse, Netbeans, IntelliJ), image editors (Adobe Photoshop, Illustrator), or for that matter, browser extensions to Chrome, Firefox, etc. The Bob Martin blog post linked to above may be helpful in answering this question.

3. (20 pts) Provide a *short* explanation (1-2) sentences that captures the main idea of the decorator pattern, one that you might commit to memory and whip out at a job interview or exam question. (Keep it *short*—the idea is that you'd expand on this description if you were asked for details.)

2

h10

CS56 M18

4. (20 pts) The chapter has a section that describes some of the problems that can occur with decorators. One of those is when “client code” (which means code that calls the classes we are designing and implementing) relies on specific types. This might be code that looks for specific types of `Beverage`, for example, such as `HouseBlend`, `DarkRoast`. For example, inside a for loop the processes every element of an `ArrayList<Beverage> drinks`, such as `for (Beverage b: drinks)` it might say something like `if (b instanceof DarkRoast)`.

   Explain why code such as this might break when we are using the Decorator pattern.

5. (20 pts) In the "interview" called "Confessions of a Decorator", the decorator pattern character describes a way of thinking about the Java I/O system and all of the various little classes used for input. The character indicates a way of thinking about these in terms of the decorator pattern that makes them easier to understand. Describe this way of thinking.

6. (20 pts) Suppose you were called upon to implement decorator classes for a `Student` class with methods `public String getName()`, `public int getPerm()`. Assume that there are already subclasses of `Student` called `GradStudent` and `UndergradStudent`, and those are so embedded in the code that it isn’t feasible to change those.

   The main thing you want to be able to provide through your decorator classses is a way to implement a method called `public String getDescription()` that could get a description of the student with as many details as the various decorators provided. Decorators might be used to add details to the description of the student, such as a major, college, year in school, transfer student, international student, research project title, etc.

   Write the code for a base class `StudentDecorator` that your various decorators (e.g. `MajorStudentDecorator`, `CollegeStudentDecorator`, will inherit from.)