CPI221 – Assignment 4
Decorator & Factory

50 Points

Topics:

- 4 Pillars of Object-Oriented Programming
- Create and use inheritance & polymorphism
- Static Methods
- Object Oriented Design Patterns
    - Factory
    - Decorator

Description

Our goal is to use a combination of the Decorator design pattern and the Factory design pattern to create a simple store to purchase complex objects. This can be entirely in the console using simple console input and output. The user should be able to pick an item from the menu and then fill that item out with ingredients until they are done.

Use the following Guidelines:

- Give identifiers semantic meaning and make them easy to read (examples numStudents, grossPay, etc).
- Keep identifiers to a reasonably short length.
- User upper case for constants. Use title case (first letter is upper case) for classes. Use lower case with uppercase word separators for all other identifiers (variables, methods, objects).
- Use tabs or spaces to indent code within blocks (code surrounded by braces). This includes classes, methods, and code associated with ifs, switches and loops. Be consistent with the number of spaces or tabs that you use to indent.
- Use white space to make your program more readable.

Important Note:

All submitted assignments must begin with the descriptive comment block. To avoid losing trivial points, make sure this comment header is included in every assignment you submit, and that it is updated accordingly from assignment to assignment.

Programming Assignment:

Instructions:

You are creating Point of Sale software for a small business with highly customizable products. The customer will order any number of items that can be customized any number of ways.

You'll employ the Decorator Pattern to allow the user to customize their products and store each "Big Fancy Object" (BF-Object) in an appropriate data-structure (probably an ArrayList).

When the order is complete. You will summarize the order into an invoice giving a breakdown of each item, the price of each item and the total.

Specifications:

You can build any type of store you wish, but the customer should have at least 3 things they can build (concrete objects) with at least 10 optional ingredients for their item.

Examples:

Pizza Shop
(demonstrated in class, pick something else)

- Pizzas
  - Small Pizza
  - Medium Pizza
  - Large Pizza
- Options
  - Pepperoni
  - Sausage
  - Chicken
  - Hamburger
  - Black Olive
  - Mushroom
  - Onion
  - Peppers
  - Tomatoes
  - Garlic
  - Basil
  - Extra Cheese

Flower Shop

- Order
  - Bouquet
  - Vase
  - Table Display
  - Large Display
- Flowers
  - Roses
    - Purple
    - Red
    - White
    - Yellow
  - Orchids
  - Tulips
  - Lilies
- Extras
  - Glitter
  - Card
  - Storage Vase

Factory

You should build a factory class to handle building your decorated BF-Objects. Pass into your factory something that represents the customer's complete item. This can be an array/arrayList of strings or integers or characters … this is up to you. Make sure you keep maintainability and extensibility in mind when designing your factory and decorator pattern.

The entire custom object order should be passed into the Factory and the Factory should handle the creation and wrapping of your BF-Object before returning it.

Some students make the mistake of passing one decorator option at a time … we want the whole order for the object!

User Interface:

Your user interface is up to you. But do keep a customer focus. The user interface should be clear and easy to use.

I recommend using some sort of sentinel value to indicate when the customer is done adding customization options.

> *I personally coded this in the past with a numerical menu interface and had them input a zero to indicate they were done customizing. Then I pass the collected customizations to my factory to build the BF-Object.*

Remember the customer can order as many items as they like and customize each item as they see fit.

When they have completed the order display a complete invoice of what they ordered.

EXTRA CREDIT OPPORTUNITY +5

Store within your Decorator Pattern a way to "extract" the order that originally generated the BF-Object.

Use this to provide the user the ability to edit an Ordered Item they have already created.

The idea here is that you would "extract" the original order, get the edits from the user, then destroy and rebuild the BF-Object from the new edited order.

Build all appropriate UI for this to happen.

EXTRA CREDIT OPPORTUNITY +2

Add an option to your program to read an order from File I/O and output your Invoice to a File.

Make sure you provide documentation on how to uses these features, i.e. how do you properly make an input file?

Sample Output:

```
Welcome to Decorator Pizza!

Please begin your order:

1. Small Pizza
2. Medium Pizza
3. Large Pizza
0. Complete Order and Display Invoice

>>

What toppings would you like to add?
1.  Pepperoni
2.  Sausage
3.  Chicken
4.  Hamburger
5.  Black Olive
6.  Mushroom
7.  Onion
8.  Peppers
9.  Tomatoes
10. Garlic
11. Basil
12. Extra Cheese
0. Finish this pizza

>>


Your Order
---------------------------------
Medium Pizza
       -- Extra Cheese
       -- Black Olives
       -- Mushrooms
Sub-total:                          $13.49
Tax:                                 $1.07
Total:                              $14.56
```

Grading of Programming Assignment

The TA will grade your program following these steps:

(1) Compile the code. If it does not compile a U or F will be given in the Specifications section. This will probably also affect the Efficiency/Stability section.

(2) The TA will read your program and give points based on the points allocated to each component, the readability of your code (organization of the code and comments), logic, inclusion of the required functions, and correctness of the implementations of each function.

Rubric:

	Levels of Achievement						
Criteria	A	B	C	D	E	U	F
Specifications Weight 50.00%	100 % The program works and meets all of the specifications.	85 % The program works and produces the correct results and displays them correctly. It also meets most of the other specifications.	75 % The program produces mostly correct results but does not display them correctly and/or missing some specifications	65 % The program produces partially correct results, display problems and/or missing specifications	35 % Program compiles and runs and attempts specifications, but several problems exist	20 % Code does not compile and run. Produces excessive incorrect results	0 % Code does not compile. Barely an attempt was made at specifications.
Code Quality Weight 20.00%	100 % Code is written clearly	85 % Code readability is less	75 % The code is readable only by someone who knows what it is supposed to be doing.	65 % Code is using single letter variables, poorly organized	35 % The code is poorly organized and very difficult to read.	20 % Code uses excessive single letter identifiers. Excessively poorly organized.	0 % Code is incomprehensible
Documentation Weight 15.00%	100 % Code is very well commented	85 % Commenting is simple but solid	75 % Commenting is severely lacking	65 % Bare minimum commenting	35 % Comments are poor	20 % Only the header comment exists identifying the student.	0 % Non existent
Efficiency Weight 15.00%	100 % The code is extremely efficient without sacrificing readability and understanding.	85 % The code is fairly efficient without sacrificing readability and understanding.	75 % The code is brute force but concise.	65 % The code is brute force and unnecessarily long.	35 % The code is huge and appears to be patched together.	20 % The code has created very poor runtimes for much simpler faster algorithms.	0 % Code is incomprehensible

What to Submit?

You are required to submit your solutions in a compressed format (.zip). Zip all files into a single zip file. Make sure your compressed file is labeled correctly - <lastname>_<firstname>_Assn4.zip

The compressed file MUST contain the following:

- All your decorator classes
- All your concrete classes
- All your abstract classes
- Anything else you've created for your project
- <lastname>_shop.java

No other files should be in the compressed folder.

If multiple submissions are made, the most recent submission will be graded, even if the assignment is submitted late.

Where to Submit?

All submissions must be electronically submitted to the respected homework link in the course web page where you downloaded the assignment.

Academic Integrity and Honor Code.

You are encouraged to cooperate in study group on learning the course materials. However, you may not cooperate on preparing the individual assignments. Anything that you turn in must be your own work: You must write up your own solution with your own understanding. If you use an idea that is found in a book or from other sources, or that was developed by someone else or jointly with some group, make sure you acknowledge the source and/or the names of the persons in the write-up for each problem. When you help your peers, you should never show your work to them. All assignment questions must be asked in the course discussion board. Asking assignment questions or making your assignment available in the public websites before the assignment due will be considered cheating.

*The instructor and the TA will **CAREFULLY** check any possible proliferation or plagiarism. We will use the document/program comparison tools like MOSS (Measure Of Software Similarity: http://moss.stanford.edu/) to check any assignment that you submitted for grading. The Ira A. Fulton Schools of Engineering expect all students to adhere to ASU's policy on Academic Dishonesty. These policies can be found in the Code of Student Conduct:*

http://www.asu.edu/studentaffairs/studentlife/judicial/academic_integrity.htm

ALL cases of cheating or plagiarism will be handed to the Dean's office. Penalties include a failing grade in the class, a note on your official transcript that shows you were punished for cheating, suspension, expulsion and revocation of already awarded degrees.