CSCI 3400 Object Oriented Design
Fall 2023
Exam 01
Full Credit: 70
Assigned: Oct 10, 2023 (10:00 AM) ← Tuesday
Due: Oct 13, 2023 (11:59 PM) ← Friday

Policies:

- This is an open-book, closed-person test. You are allowed to use any resources (textbook, lecture notes, YouTube videos, etc.) while preparing your answers.

- You are **not allowed**
  - to post the questions anywhere (like online bulletin boards).
  - to discuss your progress or any question/answer with anyone other than the instructor (not even with your advisors).
- You may request a (Zoom/in-person) appointment with your instructor or email her your question(s). Please use fhamid@ncf.edu as my email id. A direct email from Canvas goes to my personal email which I forget to notice sometimes.

- Write your answers (typed, not handwritten) for problem 03 in a file named answersLastName.txt (LastName should be your last name).

- Write the following two statements with your signature on a separate file named coverSheetLastName.pdf. The cover sheet should be **handwritten**. You may take a picture of the handwritten statements (with your signature and date on it) and save it as coverSheetLastName.pdf

  ********** cover sheet statement **********
  - I have neither received nor given inappropriate assistance on this examination.
  - I am unaware of other students who have given or received inappropriate assistance on this examination.
  ****
  Note that the statements must be true; if you cannot sign either statement, please talk to your instructor at your earliest convenience. You need not reveal the particulars of the dishonesty, simply report that it happened. Note also that "inappropriate assistance" is assistance from (or to) anyone other than the course faculty.

- Your work will not be graded without a signed cover sheet.

- Put all the files in the `**examI3400LastName**` directory, compress them, and submit them on canvas.
- **Late submissions will not be accepted/graded.**
- Problem descriptions start from page 02.

****

Problem 01: Co-operating Classes

Assume we have two classes: PhoneManufacturer and NetworkServiceProvider, who would act like real-life phone manufacturers (Apple, Samsung, etc.) and network providers (AT&T, T-Mobile, Verizon, etc.):

- Each network provider has an interest in different manufacturers.
- The providers want a notice whenever a manufacturer releases a new phone model.
- Whenever the provider receives a notification, they release a message for their clients.

Simulate the situation using the most appropriate design principles and patterns. You may use the given code as a starter.

Put all the classes/interfaces in a package named **`problem01exam01`**.

```
/* starter code */
class PhoneManufacturer{
     private String name;
     private String latestModel;
     private Double price;

     // feel free to add constructors, getters, setters,
     toString, etc.

}
class NetworkServiceProvider{

     private String name;
     private Integer id;

     // feel free to add constructors, getters, setters, etc.

}
/*************************************************/
```

Problem 01 Point Chart	
Complete the code with appropriate design pattern	15
Add two new test cases (either a manufacturer or a dealer)	05
Write Javadoc comments for each implemented class	05
Total	25

Problem 02: Friendly Pizza Hut

You are about to implement a system for your friend's new Pizza store, "Friendly Pizza Hut". Your friend plans to offer the following varieties of pizzas and several topping options:

Pizza	**Calories**	**Price**
Basic Pizza Margherita (tomato, cheese)	1104	4.99
Basic Hawaiian Pizza (tomato, cheese, ham, pineapple)	1024	6.49
Basic Salami Pizza (tomato, cheese, salami)	1160	5.99
Family Size Pizza	x 1.95	+ 4.15
Toppings	**Calories**	**Price**
Cheese	92	0.69
Ham	35	0.99
Onions	22	0.69
Pineapple	24	0.79
Salami	86	0.99

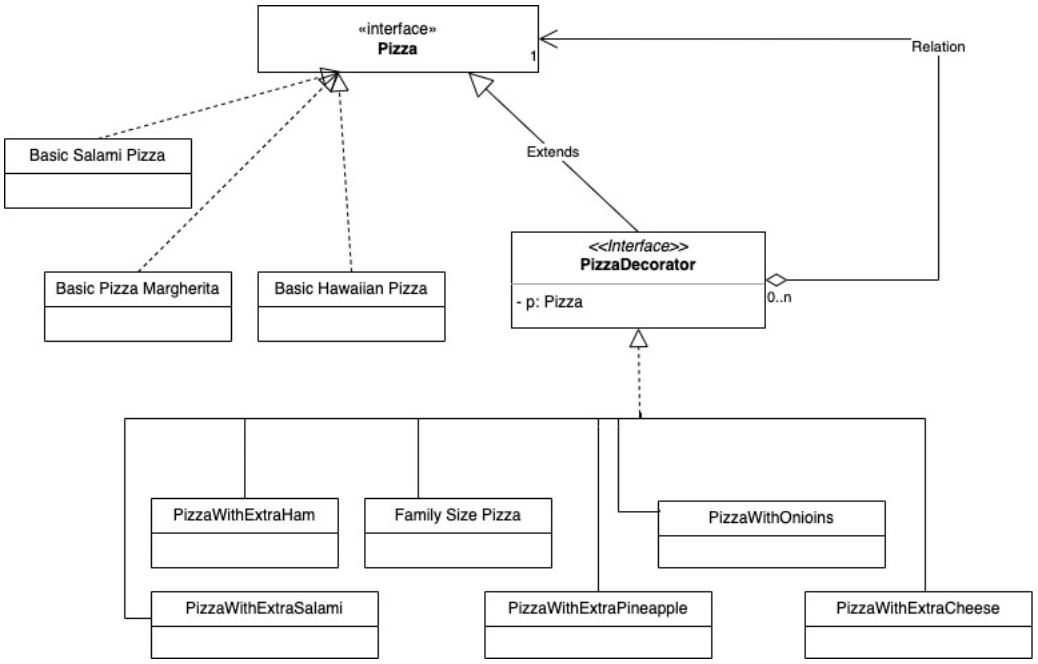
As a first milestone, you plan to complete the following tasks:

- Implement several units (classes, interfaces, etc.) in Java for the Friendly Pizza Hut that lets users order different types of pizza (with or without toppings).
    - You want to choose properties and methods carefully.
- Your design should use immutable objects, i.e., orders and pizzas are never changed in place. For example, adding a topping to a pizza creates a new pizza object.
- You plan to use the Decorator Pattern to implement the different variations of pizza (additional toppings, family-sized). [A tentative UML is attached at the end.]
- Once the pizza is ready, your system prints some information about the pizza. For example, for a family-size Hawaiian Pizza with no Extra Toppings it may print something like the following:

```
Here is your family-size Hawaiian Pizza with tomato, cheese,
pineapple, and ham, which costs 10.64 with 1996.8 calories.
Enjoy!!!
```

- Put your units in a package called **`problem02exam01`**.
- Factory is also a relevant pattern for this problem. But if you don't want to, you may avoid using the Factory pattern for now.
- Create the following pizza instances in your test derive (in the main method):

Problem 02 Point Chart	
Complete the code with the appropriate design pattern	20
Add some test cases in the test drive	05
Write Javadoc comments	05
Total	30



A tentative UML for the system

Problem 03: Narrative Questions

a) For each case, give an example of when
   - You prefer a class adapter over an object adapter.
   - You choose not to implement/use the decorator pattern.

b) State (with proper examples) the differences between the Abstract Factory and Factory Method Pattern.

c) Provide a concrete example of how good software design can benefit a software development team.

Point Chart	
Problem 03 (a)	05
Problem 03 (b)	05
Problem 03 (c)	05
Total	15

Submission:
Place all the files in **examI3400LastName** directory, compress and submit.
Sample Files and Directories/packages to be placed in the **examI3400LastName**:

Packages:

```
problem01exam01 (your source code here)
problem02exam01 (your source code here)
```

Files:

```
coverSheetLastName.pdf
answersLastName.txt
```

In every instance, your last name should replace the LastName term.