Program #1 Decorated Pizzas **10%**
Due: 03/18/2019

Starting files:

- prog2_starting_files.zip

Your program **must** work using the provided input files which should recreate the examples shown below. Obtain your input using the util.Keyboard class and make sure that your input gets read in correctly using < as we have done numerous times in class. **I will not grade your program if I cannot obtain the input using a text file**. I will be writing new input files using the same format to grade your programs. Your classes should also work with my GUI (provided in the zip), as this ensures that all of your methods are designed as described below.

A pizza is **composed** of a crust (with tomato sauce and cheese) and toppings.

Crust

A small pizza costs $5.99, a medium is $7.99, and a large is $9.99. A hand-tossed crust is an additional $0.50, and a deep dish pan crust is an additional $1.00.

Write a **CrustSize** enum [S(5.99), M(7.99), and L(9.99)] and a **CrustType** enum [THIN(0.00), HAND(0.50), and PAN(1.00)]. Place your enums in **CrustEnum.java**. Both enums should have cost methods.

Write a Crust class with a constructor that accepts and sets the crust size and type and provide a crustCost method. Write a toString method to report the state of the crust. Write getCrust (returns "THIN", "HAND", or "PAN") and getSize (returns a **char** 'S', 'M', or 'L').

Toppings

The toppings and their respective abbreviation and cost is as follows:

- Pepperoni P 0.99
- Sausage S 0.99
- Onions O 0.79
- GreenPeppers G 0.69
- Mushrooms M 0.79

A pizza also has toppings, such as pepperoni, sausage, mushrooms, peppers, and/or onions. One way to handle all of the various combinations of toppings that can be ordered is to have a boolean for each possible topping in a Pizza class. However, this can cause headaches if new toppings are added to the menu. Therefore, you will use the **Decorator design pattern** to handle the toppings.

Each topping will have an associated class (i.e. a Pepperoni class) that will extend an **abstract DecoratedPizza class**. The DecoratedPizza class itself has a private DecoratedPizza instance variable **next_pizza_item**, a no-argument default constructor (**next_pizza_item** set to null), a constructor that accepts a DecoratedPizza (and sets **next_pizza_item** to this DecoratedPizza), and three methods:

- public double pizzaCost() //get the cost from the "next_pizza_item" DecoratedPizza
- public String toString() //get the state of the "next_pizza_item" DecoratedPizza
- public String getImage() //get the abbreviation of the "next_pizza_item" DecoratedPizza (the topping abbreviation is used to obtain the correct pizza image)

Whenever a topping is added to the pizza, pass the current DecoratedPizza to the constructor for the associated topping (which in turn calls the appropriate constructor in the parent class). This allows a DecoratedPizza with an arbitrary number of toppings to be built up by wrapping, or decorating, each topping on top of the current DecoratedPizza (**essentially a linked list**). For example, if pepperoni is added as a first topping, then the DecoratedPizza minus the pepperoni is the instance variable stored in the Pepperoni class. When the pizza cost is required, get the pizza cost from the parent class (which gets the cost from the "next_pizza_item") and add the cost of pepperoni to it, returning the total cost including pepperoni. When the image of the pizza is required, get the image file (a String) from the instance variable, and **append** a "P" to it. The toString method works similarly.

Thus, if a new topping is added to the menu, a new class is written for that topping, and all of the current code can be used without modification. Complete classes for Pepperoni, Onions, Sausage, GreenPeppers, and Mushrooms.

Pizza

The pizza class also extends DecoratedPizza, but it has only the Crust as an instance variable, so the Pizza class will call the default constructor in its parent class (DecoratedPizza). The Pizza class represents the "base" pizza (with no toppings). The pizzaCost and toString methods will use the Crust instance variable, and the getImage method **appends** an S, M, or L to the String representing the image file. The size of the pizza will then be the first letter added to the image file String. The final image file name will look like MPOS.jpg for a medium pizza with pepperoni, onions, and sausage. This means that the pepperoni topping was selected first, followed by onions and then sausage. The GUI will append the ".jpg" for you.

Pizza Builder

There is a lot of input validation that must be done in this program. You will use the **Builder design pattern** to handle input validation. The Builder design pattern places all of the input validation in a separate class, making the core classes much more readable. PizzaBuilder will need to keep track of some information as instance variables. One of these is the **top link** in the DecoratedPizza (the head of the linked list). Use as few instance variables as possible (I will count off for bad designs).

Write the following methods in PizzaBuilder:

- protected void buildPizza()

  //create a Crust and a Pizza using that Crust based on the user's specifications (the Pizza is now ready for toppings)

- public PizzaBuilder() //start with a small, thin pizza with no toppings as the default
- public boolean setSize(char try_size) //returns true if the input was valid ("S" or "small", etc., not case sensitive, use the String charAt method to get the first character)
- public boolean setCrust(String try_crust) //("thin", "hand", or "pan", not case sensitive)
- public void addTopping(char topping_char) //compare the topping abbreviation to topping_char to determine which topping to add (using void here is convenient for the PizzaDriver, ignore invalid abbreviations)
- public DecoratedPizza pizzaDone() //return the final DecoratedPizza and reset to the default pizza if another pizza is desired

Specialty Pizzas

- Ham H 0.89
- Pineapple A 0.89

Extend PizzaBuilder and override the buildPizza() method (use super) to add the toppings required to create various specialty pizzas like **MeatLover's**, **VeggieLover's** and **Hawaiian** (ham and pineapple). Note that you are only adding new behavior in the overridden method. These classes should be very short.

Pizza Driver

Write a driver, **PizzaDriver.java**, to allow the user to order an indefinite number of pizzas. Place PizzaDriver in the pizza package. This is done by creating a PizzaBuilder, which will return the completed DecoratedPizza when pizzaDone is called. Include the following methods:

- private static int menu() //show the menu choices, wait for and return the valid selection
- private static void requestSize(PizzaBuilder pizza_builder) //request the crust size, wait for a valid response confirmation from PizzaBuilder
- private static void requestCrust(PizzaBuilder pizza_builder) //request the crust type, wait for a valid response confirmation from PizzaBuilder
- private static void requestToppings(PizzaBuilder pizza_builder) //ask for toppings until Done indicated (invalid toppings are ignored)
- private static void showOrder(DecoratedPizza dec_pizza) //display the pizza and its total cost
- public static void main (String[] args) //allow the user to order multiple pizzas if desired, call the other methods, track total cost and number of pizzas

After a pizza has been specified, display the order details and the cost for the pizza. Format your cost output to display two decimal places. Use the DecimalFormat class (java.text.DecimalFormat). Prompt for another pizza and either restart the order process (you will need to create a new PizzaBuilder object), or stop execution, based on the user response.

When the user is done ordering, report the number of pizzas ordered and the total cost of all the pizzas ordered.

Pizza Factory

The individual topping classes really just hardcode in the values for that topping. This is not a good class design. A much better class would allow these values to be passed to the constructor.

Improve your design by using a **PizzaTopping** class and a **PizzaToppingFactory** class (with static methods and no constructor) instead of individual topping classes (**but don't delete your previous classes**, I want to see them). New toppings can be added to the menu by adding methods to the PizzaToppingFactory class rather than writing a new class.

- **PizzaTopping constructor**: public PizzaTopping(DecoratedPizza pizza_component, String topping_string, String topping_letter, double topping_cost)
- **PizzaToppingFactory example method**: public static DecoratedPizza addPepperoni(DecoratedPizza dec_pizza) //create a PizzaTopping with the Pepperoni values and add it to the passed DecoratedPizza, returning the result

PizzaDiscount

- **PizzaDiscount constructor**: public PizzaDiscount(DecoratedPizza pizza_component, String msg, double discount) //discount is assumed to be between 0.0 and 1.0

A PizzaDiscount extends DecoratedPizza adjusting the final cost of the pizza by the discount. Add an addDiscount method to PizzaBuilder. **Adjust addTopping to make sure that a topping is not added after the discount**. That is, a PizzaTopping can only be connected to other PizzaToppings or a Pizza (use instanceof). PizzaBuilder really helps manage this complexity, keeping this code out of the other classes.

In PizzaDriver, ask the user if they are senior citizens. If so, apply a 10% discount to the order.

PizzaFee

- **PizzaFee constructor**: public PizzaFee(DecoratedPizza pizza_component, String msg, double fee)

A PizzaFee extends DecoratedPizza, adding on a flat fee to the total order at the very end. Add an addFee method to PizzaBuilder.

Adjust addDiscount so that PizzaDiscounts can only be connected to other PizzaDiscounts, PizzaToppings, or Pizza. As PizzaDriver will probably ask for these items where appropriate, I will look at your code to make sure this check is performed.

In PizzaDriver, ask the user if they want delivery. If so, add a $2.50 delivery fee to the order.

```
Would you like to order a pizza (y/n)? y

1. Meat Lover's
2. Veggie Lover's
3. Hawaiian
4. Build Your Own

Select from the above: 4
What size pizza (S/M/L)? ZWhat size pizza (S/M/L)? m
What type of crust (thin/hand/pan)? thin
(P)epperoni,(O)nions,(G)reen Peppers,(S)ausage,(M)ushrooms,(D)one
m
(P)epperoni,(O)nions,(G)reen Peppers,(S)ausage,(M)ushrooms,(D)one
g
(P)epperoni,(O)nions,(G)reen Peppers,(S)ausage,(M)ushrooms,(D)one
s
(P)epperoni,(O)nions,(G)reen Peppers,(S)ausage,(M)ushrooms,(D)one
m
(P)epperoni,(O)nions,(G)reen Peppers,(S)ausage,(M)ushrooms,(D)one
q
(P)epperoni,(O)nions,(G)reen Peppers,(S)ausage,(M)ushrooms,(D)one
d

Are you a senior citizen (y/n)? n
Do you need this pizza delivered (y/n)? y
Your pizza:
Size: M
Crust: THIN
Toppings:
mushrooms
green peppers
sausage
mushrooms
delivery

The cost of your pizza is $13.75

Would you like to order another pizza (y/n)?
1. Meat Lover's
2. Veggie Lover's
3. Hawaiian
4. Build Your Own

Select from the above: 1
What size pizza (S/M/L)? L
What type of crust (thin/hand/pan)? pan
Are you a senior citizen (y/n)? y
```

```
Do you need this pizza delivered (y/n)? n
Your pizza:
Size: L
Crust: PAN
Toppings:
pepperoni
sausage
ham
senior discount

The cost of your pizza is $12.47

Would you like to order another pizza (y/n)?
1. Meat Lover's
2. Veggie Lover's
3. Hawaiian
4. Build Your Own

Select from the above: 4
What size pizza (S/M/L)? S
What type of crust (thin/hand/pan)? HAND
(P)epperoni,(O)nions,(G)reen Peppers,(S)ausage,(M)ushrooms,(D)one
P
(P)epperoni,(O)nions,(G)reen Peppers,(S)ausage,(M)ushrooms,(D)one
O
(P)epperoni,(O)nions,(G)reen Peppers,(S)ausage,(M)ushrooms,(D)one
J
(P)epperoni,(O)nions,(G)reen Peppers,(S)ausage,(M)ushrooms,(D)one
d

Are you a senior citizen (y/n)? n
Do you need this pizza delivered (y/n)? y
Your pizza:
Size: S
Crust: HAND
Toppings:
pepperoni
onions
delivery

The cost of your pizza is $10.77

Would you like to order another pizza (y/n)? n
You ordered 3 pizza(s) for a grand total of $36.99.
```

Program Specification:

- Creation of a custom classes using composition and a design pattern
  - Encapsulation (private instance variables of different types, including Object types)
  - Multiple constructors
  - Get methods to report specific states (message forwarding)
  - Set methods to control state changes
  - Custom toString to report the state of the object
  - Computation using state information
  - **Decorator design pattern** //easy to add new classes to the project
  - **Builder design pattern** //input validation external to the core classes
- Driver class
  - Use of conditional statements
  - Use of a while loop with a sentinel value
  - Creation and use of helper methods to simplify problem solving
  - Object parameters
  - **Polymorphism**

Submission:
Electronic Submission: **All files needed to run your program!**