Due Date: 3/12/2024, 11:59pm

CMP_SC 3330 – Object-oriented Programming
Homework 5

This assignment is an automation of a Pizza Store called FakeSpeare. The assignment involves the implementation of Factory and Strategy design patterns, Enum classes with methods, and assigning values to enums.

Class Definition and Program Requirements:

AbstractPizza Class:

- Implement an **abstract** base class ***AbstractPizza*** with protected attributes/fields *toppingList*(**List<Toppings>**), *priceWithoutToppings*(**double**), *totalPrice*(**double**), *pizzaOrderID*(**int**), *orderIDCounter*(**static int**), *cookingStrategy*(**ICookingStrategy**), *cookingPrice*(**double**). Also, a constructor should be implemented to instantiate the toppingList as an ArrayList and respective setter and getter methods.

MargheritaPizza, VegetarianPizza, HawaiianPizza, and SupremePizza Classes:

- Create subclasses of the ***AbstractPizza*** class called ***MargheritaPizza***, ***VegetarianPizza***, ***HawaiianPizza***, and ***SupremePizza***, each representing a different type of pizza. Make sure that all classes have a constructor. The subclasses must use the super keyword to initialize the attributes. Also, implement the respective copy constructors, and the respective setter and getter methods.
- Implement copy constructors for each pizza to prevent information leaks.
- Implement the **toString()** method that will display all the field information.
- The following abstract methods are provided for implementation and overriding:
    - `protected double addTopingsToPrice(double priceWithoutToppings);` // calculates the total price of the pizza using priceWithoutToppings and the prices of each topping in the toppingsList. It also assigns totalPrice attribute to the calculated total price, and priceWithoutToppings attribute to the passed parameter. This could be called once to add the default toppings.
    - `public double updatePizzaPrice();` // calculates and updates the totalPrice of the pizza using priceWithoutToppings attribute and the prices of each topping in the toppingsList. You can use this method on each update you make with the pizza.
- Once an object is created from one of the subclasses of pizzas, it should add the default toppings respectively:
    - ***MargheritaPizza:*** TOMATO, CHEESE
    - ***VegetarianPizza:*** TOMATO, CHEESE, BELL_PEPPER, BLACK_OLIVE, MUSHROOM
    - ***HawaiianPizza:*** CANADIAN_BACON, CHEESE, PINEAPPLE
    - ***SupremePizza:*** TOMATO, CHEESE, BELL_PEPPER, ITALIAN_SAUSAGE, PEPPERONI, BLACK_OLIVE, MUSHROOM

- Also, once an object is created from one of the subclasses of pizzas, it should have a default price without the toppings (you can consider it as preparation cost). The default pizza prices without toppings are given below:
  - ***MargheritaPizza price without toppings:*** $2.50
  - ***VegetarianPizza price without toppings:*** $1.50
  - ***HawaiianPizza price without toppings:*** $3.00
  - ***SupremePizza price without toppings:*** $3.50

- Implement a public **interface *ICookingStrategy*** with `public boolean cook(AbstractPizza pizza)` method to be implemented.
- Create subclasses implementing ***ICookingStrategy: BrickOvenCookingStrategy, ConventionalOvenCookingStrategy****, and* ***MicrowaveCookingStrategy****.*
- The subclasses should implement and override the `cook` method, which sets the cookingPrice, cookingStrategy and updates the pizzaPrice of the pizza that is being cooked and passed as a parameter. There could only be one cooking strategy selected for a pizza. The cooking strategy changes the pizza price as below:
  - ***BrickOvenCookingStrategy:*** Additional $10.0
  - ***ConventionalOvenCookingStrategy:*** Additional $8.0
  - ***MicrowaveCookingStrategy:*** Additional $1.0

PizzaOrder Class:

- Implement a public class ***PizzaOrder*** that manages the pizza orders with private attributes *pizzaFactory*(**PizzaCookingFactory**), *cookingStrategy*(**ICookingStrategy**), pizzaOrderList(**List<AbstractPizza>**).
- Implement a constructor that instantiates the *pizzaCookingFactory* and *pizzaOrderList* attributes.
- Implement a method `public void printListOfToppingsByPizzaOrderID(int orderID)`. This method gets the pizza order with the given pizza order ID and prints the toppings of that order.
- Implement a method `public void printPizzaOrderCart(int orderID)`. This method prints the pizzas in the *pizzaOrderList*.
- Implement a method `public AbstractPizza getPizzaByOrderID(int orderID)`. This method finds the pizza order with the given pizza order id and returns it.
- Implement a method `public boolean addPizzaToCart(PizzaType pizzaType)`. This method creates a new pizza with the given PizzaType and adds it to the pizzaOrderList .
- Implement a method `public boolean addNewToppingToPizza(int orderID, Toppings topping)`. This method finds the pizza order with the given ID and adds the given topping to its topping list if it doesn't already exist in the list. If the given topping is added, it also updates the pizza price and returns true. If the topping already exists in the topping list of the pizza, it returns false.
- Implement a method `public boolean removeToppingFromPizza(int orderID, Toppings topping)`. This method finds the pizza order with the given ID

and removes the given topping from its topping list if it exists in the list. If the given topping is removed, it also updates the pizza price and returns true. If the topping doesn't exist in the topping list of the pizza and cannot be removed, it returns false.

- Implement a method `public boolean isThereAnyUncookedPizza()`. This method checks the pizzas in the pizzaOrderList and checks their cooking strategies. It returns true if there are any pizzas without any assigned pizza cooking strategy. It returns false if there are no pizzas without an assigned cooking strategy.
- Implement a method `public double checkout() throws Exception`. This method checks if there are any uncooked pizzas. If all pizzas are cooked, it calculates the total price of all pizzas and returns the total cart price. However, if there is at least one uncooked pizza it throws an exception (Use the general Exception class). The checkout method calls the isThereAnyUncookedPizza method to check for uncooked pizzas and throws an exception.
- Implement a method `public boolean selectCookingStrategyByPizzaOrderID(int orderID, CookingStyleType cookingStrategyType)`. This method gets the pizza with the given order ID, instantiates the *cookingStrategy* according to the cookingStrategyType parameter. Calls the cook function for the pizza of the pizza order with the given order ID.

PizzaCookingFactory Class:

- Implement a public class ***PizzaCookingFactory*** with a method `public AbstractPizza createPizza(PizzaType pizzaType).` It creates an AbstractPizza instance and instantiates it according to the pizzaType parameter. It also sets the pizzaOrderID of the pizza using the current orderIDCounter of the pizza. The orderIDCounter will be incremented and assigned automatically on every creation of pizza, to assign unique pizza order IDs.

PizzaType Enum:

- The types are:
    - HAWAIIAN,
    - MARGHERITA,
    - SUPREME,
    - VEGETARIAN
- Enum class also needs to have a private *toppingPrice*(**double**) attribute, a constructor and getter.

Toppings Enum:

- The toppings are:
    - TOMATO(1.50),
    - CHEESE(2.00),
    - PINEAPPLE(2.50),
    - BLACK_OLIVE(1.25),
    - ITALIAN_SAUSAGE(3.50),
    - PEPPERONI(3.00),
    - BELL_PEPPER(1.00),
    - MUSHROOM(1.50),
    - CANADIAN_BACON(4.00);

CookingStyleType Enum:

- The cooking styles are:
    - MICROWAVE,
    - CONVENTIONAL_OVEN,
    - BRICK_OVEN

Main Class:

```
public class Main {
  public static void main(String[] args) {
    // Instantiate a pizzaOrder, perform operations based on the requirements.
    PizzaOrder order = new PizzaOrder();

    // Adds pizzas to the cart, selects cooking strategies for the pizzas in the cart,
prints pizza order cart. Calls checkout to calculate the bill, throws exception if
triggered.
    // TODO
  }
}
```

Submission Guidelines:

- Each team is required to create a GitHub repository for the project.
- The repository should include all the required Java files (Main.java, AbstractPizza.java, HawaiianPizza.java, MargheritaPizza.java, PizzaCookingFactory.java, PizzaType.java, SupremePizza.java, Toppings.java, VegetarianPizza.java, BrickOvenCookingStrategy.java, ConvenctionalOvenCookingStrategy.java, CookingStyleType.java, ICookingStrategy.java, MicrowaveCookingStrategy.java, PizzaOrder.java, any dependencies, and a brief explanation of the project.
- Verify that the repository is accessible and properly organized, allowing anyone to clone and run the program without additional configuration.
- Your program must use the classes with described methods, given prototypes and signatures exactly. You are allowed to implement additional helper methods and classes.
- Late submission between 0hrs < late <= 24hrs will lose half of the grade. After 24 hours, submissions will receive a grade of 0 for the assignment.
- **Not following the submission guidelines will result in a penalty on your grades.**

Note:

- Ensure that your program handles cases where the file is not found or if there are any issues during file reading.
- Make use of the concepts you've learned, such as constructors, getter/setter methods, static fields/methods, and the toString() method.
- Add additional helper methods or classes if needed.
- Test your program with different scenarios, including cases where the object is not found and the update is unsuccessful.