By 2019092824 Lee kyeong Jun

In this Document

- In this Document
- Pizza.java
  - Pizza(int _size, boolean _hasPeperoni, boolean _hasMushrooms, boolean _hasCheese)
  - public String toString() (pizza)
  - public double getPrice()
- Order.java
  - Order()
  - public void addPizza(int size, boolean hasPeperoni, boolean hasMushrooms, boolean hasCheese)
  - public double calculateOrderPrice()
  - public String toString() (order)
- PizzaStore.java
  - PizzaStore()
  - public void addCash(double amount)
  - public void createOrder(int pizzaSize, boolean hasPeperoni, boolean hasMushrooms, boolean hasCheese)
  - public void restockX(int amount)
  - public String toString() (pizzastore)
- MainClass.java
  - public void showErrorMsg(int errcode)
  - public void selectAction(PizzaStore store, Scanner input)
  - public void placeOrder(PizzaStore store, Scanner input)
  - public boolean selectIngredient(PizzaStore store, Scanner input, String label)
  - public void buyIngredients(PizzaStore store, Scanner input)
  - public static void main(String[] args)

Pizza.java

> Pizza.java determines the properties (size, haspeoni, hasmushrooms, hascheese) and price of pizza.

```
public class Pizza {
    private int size;
    private boolean hasPeperoni;
    private boolean hasMushrooms;
    private boolean hasCheese;

    /* methods below ... */
}
```

Pizza(int _size, boolean _hasPeperoni, boolean _hasMushrooms, boolean _hasCheese)

```
Pizza(int _size, boolean _hasPeperoni, boolean _hasMushrooms, boolean _hasCheese){

    size = _size;
    hasPeperoni = _hasPeperoni;
    hasMushrooms = _hasMushrooms;
    hasCheese = _hasCheese;

    if(size <= 0) {
        System.out.println("error! : Pizza has Invalid Size : " + size);
        return;
    }
    if(hasPeperoni == false && hasMushrooms == false && hasCheese == false) {
        System.out.println("error! : Pizza has no Ingredients.");
        return;
    }
}
```

- An error message was output when an invalid size (less than or equal to 0) was attempted to be set to size.
- If all toppings were not included, an error message was printed.

> The error message does not interfere with the process of declaring an object in the pizza class. That is, even if the size is negative and not all materials are included, the pizza object is normally generated. However, when other methods were applied later, stability was not guaranteed, so an error message was output.

public String toString() (pizza)

```
public String toString() {
    String ret;

    // ret example : "Pizza : [Size = 28 / Ingredients = Peperoni Mushrooms ]"

    ret = "Pizza : [Size = " + size + " / Ingredients = ";
    if(hasPeperoni) ret += "Peperoni ";
    if(hasMushrooms) ret += "Mushrooms ";
    if(hasCheese) ret += "Cheese";
    ret += "]";
    return ret;
}
```

- The process of executing Pizza's toString() directly on the main function is not included, so it was arbitrarily configured.
- It prints like `Pizza: [Size = 28 / Ingredients = Peperoni Mushrooms]`.

public double getPrice()

```
public double getPrice() {
    double offset = 0;

    if(hasPeperoni) offset ++;
    if(hasMushrooms) offset ++;
    if(hasCheese) offset ++;

    double price = size*0.25*((100-(10*offset))/100);
    return price;
}
```

- In the case of Price, you can multiply the size by 0.25 and apply the discount rate per topping.
    - The discount rate offset for the number of toppings was increased to be calculated as a percentage.

Order.java

> Order is an object containing a pizza object.

```
public class Order {
    private Pizza pizza;
    /* methods below... */
}
```

Order()

```
Order(){
}
```

- The construction of the Order does not guarantee the construction of Pizza.
    - This is because addpizza exists as a separate method.

public void addPizza(int size, boolean hasPeperoni, boolean hasMushrooms, boolean hasCheese)

```
public void addPizza(int size, boolean hasPeperoni, boolean hasMushrooms, boolean
hasCheese) {
    pizza = new Pizza(size, hasPeperoni, hasMushrooms, hasCheese);
}
```

- addpizza takes the argument and creates the pizza object inside the Order object.
    - The construction of pizza depends on the generator of the Pizza Class.

public double calculateOrderPrice()

```
public double calculateOrderPrice() {
    double price = pizza.getPrice();
    return price;
}
```

- CalcurateOrderPrice accesses its pizza object and calculates the price.
    - The calculation of pizza price depends on the Pizza Class.

public String toString() (order)

```
public String toString() {
    String ret;
    // ret = pizza.toString();
    ret = "Your total will be $" + calculateOrderPrice() + ".";
    return ret;
}
```

- Configure the order string according to the format presented in the task and return it.
    - Since the order has a pizza object, it is reasonable to also output information about pizza, but it was not presented in the task, so it was annotated.

PizzaStore.java

> PizzaStore has stock, cash, and order

```
public class PizzaStore {
    private double cash;
    private Order currentOrder;
    private int peperoniStock;
    private int mushroomStock;
    private int cheeseStock;

    /*Methods below...*/
}
```

PizzaStore()

```
PizzaStore() {
    cash = 2.5;
    peperoniStock = 1;
    mushroomStock = 1;
    cheeseStock = 1;
}
```

- The initial value of the constructor was defined as suggested in the task.

public void addCash(double amount)

```
public void addCash(double amount) {
    if(amount >= 0) cash += amount;
}
```

- In the Main function, when the Pizza is sold, the cash increases by that amount.
    - As part of the Setter, the cash of the amount alone was added immediately.

public void createOrder(int pizzaSize, boolean hasPeperoni, boolean hasMushrooms, boolean hasCheese)

```
public void createOrder(int pizzaSize, boolean hasPeperoni, boolean hasMushrooms,
boolean hasCheese){
    currentOrder = new Order();

    boolean availableOrder = true;

    int tmpPep = peperoniStock;
    int tmpMus = mushroomStock;
    int tmpChe = cheeseStock;

    if(hasPeperoni && availableOrder) {
        if(peperoniStock >= 1){peperoniStock -= 1;}
        else availableOrder = false;
    }

    if(hasMushrooms && availableOrder) {
        if(mushroomStock >= 1) {mushroomStock -= 1;}
        else availableOrder = false;
    }

    if(hasCheese && availableOrder) {
        if(cheeseStock >= 1) {cheeseStock -= 1;}
        else availableOrder = false;
    }

    if(availableOrder) {
        currentOrder.addPizza(pizzaSize, hasPeperoni, hasMushrooms, hasCheese);
```

```
            System.out.println(currentOrder.toString());
            addCash(currentOrder.calculateOrderPrice());
        }
        else {
            System.out.println("The order is Not Available.");
            peperoniStock = tmpPep;
            mushroomStock = tmpMus;
            cheeseStock = tmpChe;
        }
    }
```

- Before creating the pizza, it was checked whether the argument was available compared to the current situation of the pizzastore.
- Declared the tmpX variable in advance to save the stock of the pizzastore before the order, and restored the stock if it was not an available order.
- The available argument is checked by the following procedure:
    - If the addition of toppings requested (hasX is True), it is available if the Xstock of the pizza store is 1 or more.
    - In the case of an order that is not once available, it is considered an order that is not available even though other elements are available.

public void restockX(int amount)

```
public void restockPeperoni(int amount) {
    double price = amount * 1;
    if(cash >= price) {
        cash -= price;
        peperoniStock += amount;
        System.out.println(amount + " of peperoni purchased. Your current Cash is
" + cash);
    }
    else {
        System.out.println("Not enough Cash. Your current cash is " + cash + " and
the price is " + price );
    }
}

public void restockMushrooms(int amount) {
    double price = amount * 1.5;
    if(cash >= price) {
        cash -= price;
        mushroomStock += amount;
        System.out.println(amount + " of mushrooms purchased. Your current Cash is
" + cash);
    }
    else {
        System.out.println("Not enough Cash. Your current cash is " + cash + " and
the price is " + price );
    }
}
```

```
public void restockCheese(int amount) {
    double price = amount * .75;
    if(cash >= price) {
        cash -= price;
        cheeseStock += amount;
        System.out.println(amount + " of peperoni purchased. Your current Cash is
" + cash);
    }
    else {
        System.out.println("Not enough Cash. Your current cash is " + cash + " and
the price is " + price );
    }
}
```

- The functions of restockX all have the same structure, and only the price per material (offset multiplied by mount) is different.
- Since there are many overlapping phrases, it is reasonable to abbreviate the function by using offset as an argument, but it was defined as a separate function in the task and followed accordingly.
- If the current cash is less than the amount of money you can buy the material, print an error message and exit.

public String toString() (pizzastore)

```
public String toString() {
    String ret = "PizzaStore : cash: $" + cash + ", peperoni:" + peperoniStock +
", mushrooms:" + mushroomStock+ ", cheeses:" + cheeseStock +".";
    return ret;
}
```

- followed the format provided in the assignment document.

MainClass.java

> Main Class.java implemented the PizzaStore Management System using the three classes (Pizza, Order, and PizzaStore) defined above.

```
package Assignment1_code_Leekyeongjun_2019092824;
import java.util.Scanner;
```

- Package and java.util.Scanner were imported to use the scanner and custom classes.

public void showErrorMsg(int errcode)

```
public void showErrorMsg(int errcode) {
    String msg;
    if(errcode == 0) {
        // Number error.
        // if size has unavailable size, than this message appears.
        msg = "Unavailable Size, Please Retry.";
    }
    else if(errcode == 1) {
        // Command error.
        // if user has typed unavailable command, than this message appears.
        msg = "Unavailable Command, Please Retry.";
    }
    else if(errcode == 2){
        // Ingredient error.
        // if there is no Ingredients in Pizza, than this message appears.
        msg = "Pizza has no Ingredients. Please Retry.";
    }
    else {
        // Unknown error.
        // This should not be appeared.
        msg = "ErrorCode Index error.";
    }
    System.out.println("[Error] : " + msg);
}
```

- Errors that may occur on the MainClass were divided by type and unified.
- Each type is divided through integer errcode.

public void selectAction(PizzaStore store, Scanner input)

> The PizzaStore Management System is, after all, a repetition of the Select Action.

```
public void selectAction(PizzaStore store, Scanner input) {

    int cmd = 0;
    System.out.println(store.toString());
    System.out.println("What would you like to do:");
    System.out.println("1: Place an order, 2: buy ingredients");

    cmd = input.nextInt();
    input.nextLine();

    while(!(cmd == 1 || cmd == 2)) {
        showErrorMsg(1);
        cmd = input.nextInt();
        input.nextLine();
    }

    if(cmd == 1) {
        placeOrder(store, input);
```

```
    }
    else if(cmd == 2) {
        buyIngredients(store, input);
    }
}
```

- In the Select Action, the user's command is input into the cmd variable, and if the cmd is available, the corresponding action is taken.
  - 1 was defined as placeOrder and 2 as buyIngredients.
  - If it is not available cmd, output an error message and receive it until available cmd comes out.
- The pizzastore object and scanner object are supposed to be received as arguments, so they must be defined on the main function.

public void placeOrder(PizzaStore store, Scanner input)

```
public void placeOrder(PizzaStore store, Scanner input) {
    int size;
    boolean hasPep= false, hasMus = false, hasChe = false;
    System.out.println("What size pizza do you want?");

    size = input.nextInt();

    input.nextLine(); // flush nextLine after nextInt();
    while(size < 0) {
        showErrorMsg(0);
        size = input.nextInt();
        input.nextLine(); // flush nextLine after nextInt();
    }

    hasPep = selectIngredient(store, input, "peperoni");
    hasMus = selectIngredient(store, input, "mushrooms");
    hasChe = selectIngredient(store, input, "cheese");

    while(hasPep == false && hasMus == false && hasChe == false) {
        showErrorMsg(2);

        hasPep = selectIngredient(store, input, "peperoni");
        hasMus = selectIngredient(store, input, "mushrooms");
        hasChe = selectIngredient(store, input, "cheese");
    }

    store.createOrder(size, hasPep, hasMus, hasChe);
}
```

- PlaceOrder receives an argument through a scanner and checks that it is available, and if it is an available argument, the corresponding order is generated using the createOrder method.
- The T/F of each material is checked through the select Ingredient function below.

public boolean selectIngredient(PizzaStore store, Scanner input, String label)

```
public boolean selectIngredient(PizzaStore store, Scanner input, String label) {
    String cmd;
    System.out.println("Do you want "+ label +" on your pizza? Y/N");
    cmd = input.nextLine();
    while(!(cmd.toUpperCase().equals("Y") || cmd.toUpperCase().equals("N"))) {
        showErrorMsg(1);
        cmd = input.nextLine();
    }

    if(cmd.toUpperCase().equals("Y")) {
        return true;
    }else return false;

}
```

- Arguments
  - store : PizzaStore object
  - input : Scanner object
  - label : The part of the string to be displayed corresponds to the topping
- In the selectIngredient, the boolean is returned so that it can be checked on the placeOrder whether each material is selected or not.
- Only the Y or N input is available cmd, but lowercase letters are allowed on the task, so apply toUpperCase() to inspect it.

public void buyIngredients(PizzaStore store, Scanner input)

```
public void buyIngredients(PizzaStore store, Scanner input) {
    int cmd;

    System.out.println("What ingredients do you want to buy?");
    System.out.println("1: peperoni, 2: mushrooms, 3: cheese, 4: none.");

    cmd = input.nextInt();
    while(cmd < 1 || cmd >= 5) {
        showErrorMsg(1);
        cmd = input.nextInt();
        input.nextLine();
    }

    if(cmd == 1) {
        store.restockPeperoni(1);
    }
    if(cmd == 2) {
        store.restockMushrooms(1);
    }
    if(cmd == 3) {
```

```
            store.restockCheese(1);
        }
        if(cmd == 4) {
            store.addCash(1);
        }
    }
```

- For available cmd, use the restockX function of pizzaStore to add materials and subtract the cash.
    - In the case of restockX, the amount is fixed to 1, so it is transferred to the argument.
- When cmd is 4, it was confirmed that cash of 1 increased. Therefore, Cash is added using addCash (1).

public static void main(String[] args)

```
public static void main(String[] args) {
    Scanner scanner =  new Scanner(System.in);
    PizzaStore myStore = new PizzaStore();
    MainClass mainClassInstance = new MainClass();

    while(true) {
        mainClassInstance.selectAction(myStore, scanner);
    }
}
```

- Instance was created because the function inside the mainclass, including selectAction, was not static.
    - This was a decision to improve the portability of the code because it is the same as using Instance when the mainclass is imported into another project in the future.
        - For example), if a person using the above management system wants to apply a system other than pizza store at the same time, it would be reasonable to limit the mainclass to a controller for pizza store.
- The main function continues to execute the selectAction in the while loop.
    - There is no separate condition for termination of the loop, so it is repeated indefinitely.