EPITECH. TECHNOLOGY

JAVA SEMINAR

DAY 10 - THE BAKERY

JAVA SEMINAR

Contents

✓ Contents
- Exercise 01
  * Food
  * Bread
  * Drink - Sandwich - Dessert
- Exercise 02
  * Menu
- Exercise 03

You already know a lot about programming.
Let's put it all together today to create a simple program to manage a bakery shop.
Everybody likes pastries.

Exercise 01

Delivery: `./Food.java`, `./Bread.java`, `./FrenchBaguette.java`, `./SoftBread.java`, `./Drink.java`, `./AppleSmoothie.java`, `./Coke.java`, `./Sandwich.java`, `./HamSandwich.java`, `./Panini.java`, `./Dessert.java`, `./Cookie.java`, `./CheeseCake.java`

First, create the food items.

Food

Create a `Food` interface.
Add the `getPrice` (float) and `getCalories` (int) public methods to your interface.

Bread

Create a `Bread` abstract class which implements `Food`.
This class must have a `price` and a `calories` attributes.
These two attributes must be passed as parameters to the constructor.

Your class must also have a `bakingTime` attribute (int). By default, it is set to 0.
Every attribute has a getter but no setter.

Now, create two classes `FrenchBaguette` and `SoftBread` which both inherit from `Bread`.
Their constructors take no parameters.

attribute	`FrenchBaguette`	`SoftBread`
`price`	0.80	1.20
`calories`	700	500
`bakingTime`	20	30

```
public class Example {
    public static void main(String[] args) {
        Food bread = new SoftBread();
        System.out.println("The softbread costs " + bread.getPrice() + " euros and
            contains " + bread.getCalories() + " calories.");
    }
}
```

```
Terminal
T-JAV-500> java Example
The softbread costs 1.2 euros and contains 500 calories.
```

Drink - Sandwich - Dessert

Create three abstract classes named `Drink`, `Sandwich` and `Dessert` which all implements `Food`.

The `Drink` class must have a boolean attribute `aCan`, set to false by default, and his getter `isACan`.
The `Sandwich` class has a boolean attribute `vegetarian`, also set to false by default.
It also has a *List* of *String* which describes the `ingredients` of the sandwich.
Each attribute should have its getter: `isVegetarian`, `getIngredients`.

Create the `AppleSmoothie` and the `Coke` classes, inherited from `Drink`, with the attributes:

attribute	`AppleSmoothie`	`Coke`
`price`	1.50	1.20
`calories`	431	105
`aCan`	false	true

Create the `HamSandwich` and the `Panini` classes, inherited from `Sandwich`, with the attributes:

attribute	`HamSandwich`	`Panini`
`price`	4.00	3.50
`calories`	230	120
`vegetarian`	false	true
`ingredients`	tomato salad cheese ham butter	tomato salad cheese avocado cucumber

Create the `Cookie` and the `CheeseCake` classes, inherited from `Dessert`, with the attributes:

attribute	`Cookie`	`CheeseCake`
`price`	0.90	2.10
`calories`	502	321

Sure, that's a lot of classes, but at least you have a good level of abstraction.

Exercise 02

Delivery: `./Food.java`, `./Bread.java`, `./FrenchBaguette.java`, `./SoftBread.java`, `./Drink.java`, `./AppleSmoothie.java`, `./Coke.java`, `./Sandwich.java`, `./HamSandwich.java`, `./Panini.java`, `./Dessert.java`, `./Cookie.java`, `./CheeseCake.java`, `./Menu.java`, `./Breakfast.java`, `./Lunch.java`, `./AfternoonTea.java`

Menu

Add a `Menu` generic abstract class which must have two attributes, `drink` and `meal`, of a templated type that implement `Food`. Every attribute has a getter but no setter.

It will also have a public `getPrice` function which returns a float representing the sum of the `drink` price and `meal` price, the total diminished by 10%.

Now create some real implementations of `Menu`, such as `Breakfast`, `Lunch` and `AfternoonTea`.

we should only be able to instanciate:

- ✓ a `Breakfast` with a `drink` subclass of `Drink` and a `meal` subclass of `Bread` ;
- ✓ a `Lunch` with a `drink` subclass of `Drink` and a `meal` subclass of `Sandwich` ;
- ✓ a `AfternoonTea` with a `drink` subclass of `Drink` and a `meal` subclass of `Dessert`.

Exercise 03

Delivery: `./Food.java`, `./Bread.java`, `./FrenchBaguette.java`, `./SoftBread.java`, `./Drink.java`, `./AppleSmoothie.java`, `./Coke.java`, `./Sandwich.java`, `./HamSandwich.java`, `./Panini.java`, `./Dessert.java`, `./Cookie.java`, `./CheeseCake.java`, `./Menu.java`, `./Breakfast.java`, `./Lunch.java`, `./AfternoonTea.java`, `./Stock.java`, `./NoSuchFoodException.java`, `./CustomerOrder.java`

Now you have your products to sell, you need a business logic to register the sales.

To do so, let's create the logic side of a cash register application (you can imagine that it will be linked to a GUI and used in a store). First, create a `Stock` class to register the stocks.

This class has `Map<Class<? extends Food>, Integer>` attribute to store the number of items for each type of food in a generic way.

Using the default constructor, each of the known food product of the stock should have 100 items.

It has various methods, such as:

- ✓ a `int getNumberOf(Class<? extends Food>)` to retrieve the number of items for a specific food ;
- ✓ a `boolean add(Class<? extends Food>)` to increment the counter by one ;
- ✓ a `boolean remove(<?Class extends Food>)` to decrement the counter by one.

If the stock doesn't contain the food type given in parameter, these methods should throw a `NoSuchFoodException` exception containing the message `No such food type: [class name].`

`add` and `remove` return *true* if the operation was successful.

Your stock can't go below 0!

Now, create a `CustomerOrder` class that contains the following methods:

- ✓ `boolean addItem(Food)`:
  - returns wether it has been added or not ;
  - removes a food item from the stock ;
  - adds a food item to the order.

✓ `boolean removeItem(Food)`:

- returns false if the item wasn't in the order ;
- removes a food item from the order ;
- adds a food item to the stock.

✓ `float getPrice()`:

- returns the total price of the order.

✓ `boolean addMenu(Menu)`:

- add the menu to the order ;
- returns true if the stock had enough items to make this menu ;
- all the item composing the menu should be removed from the stock.

✓ `boolean removeMenu(Menu)`:

- removes the menu from the order.

✓ `void printOrder()`:

- pretty print the order.

```
public class Example {
    public static void main(String args[]) {
        Breakfast<AppleSmoothie, SoftBread> breakfast = new Breakfast<>(new AppleSmoothie
            (), new SoftBread());
        Food food = new Cookie();
        Stock stock = new Stock();
        CustomerOrder order = new CustomerOrder(stock);
        try {
            order.addItem(food);
            order.addMenu(breakfast);
        } catch (NoSuchFoodException e) {
            System.out.println(e.getMessage());
        }
        order.printOrder();
    }
}
```

```
Terminal
T-JAV-500> java Example
Your order is composed of:
- Breakfast menu (2.43 euros)
-> drink: AppleSmoothie
-> meal: SoftBread
- Cookie (0.9 euros)
For a total of 3.33 euros.
```

v 3.2