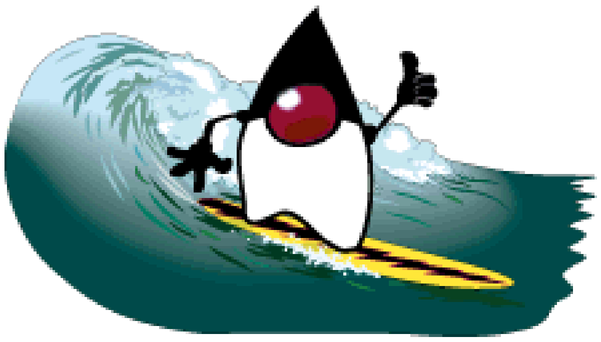
T5 - Java Seminar

T-JAV-500

Day 10

The Backery



3.0

Day 10

language: Java

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.

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

Exercise 01

Files to hand in: ./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 we will create the our 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.

`FrenchBaguette`	`SoftBread`
`price:      0.80`	`price:      1.20`
`calories:   700`	`calories:   500`
`bakingTime: 20`	`bakingTime: 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 and Dessert

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

The `Drink` class must have a `boolean` *aCan* attribute which is set to false by default and his getter **isACan**.
The `Sandwich` class has a `boolean` attribute named *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**.
Now, create two classes named `AppleSmoothie` and `Coke` inherited from `Drink` with the following characteristics:

```
AppleSmoothie

price: 1.50
calories: 431
aCan: false
```

```
Coke

price: 1.20
calories: 105
aCan: true
```

Create two more classes named `HamSandwich` and `Panini` which inherit from the `Sandwich` class.

```
HamSandwich

price: 4.00
calories: 230
vegetarian: false
ingredients: tomato, salad, cheese,
    ham, butter
```

```
Panini

price: 3.50
calories: 120
vegetarian: true
ingredients: tomato, salad, cucumber,
    avocado, cheese
```

Finally, create two classes named `Cookie` and `CheeseCake` inherited from `Dessert`.

```
Cookie

price: 0.90
calories: 502
```

```
CheeseCake

price: 2.10
calories: 321
```

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

Exercise 02

Files to hand in: ./Food.java
./Bread.java
./BackeryExceptions.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%.
You will now create some real implementations of `Menu`: `Breakfast`, `Lunch` and `AfternoonTea`.

- We should only be able to instanciate a `Breakfast` with a *drink* which is a subclass of `Drink` and a *meal* which is a subclass of `Bread`.
- We should only be able to instanciate a `Lunch` with a *drink* which is a subclass of `Drink` and a *meal* which is a subclass of `Sandwich`.
- We should only be able to instanciate a `AfternoonTea` with a *drink* which is a subclass of `Drink` and a *meal* which is a subclass of `Dessert`.

Exercise 03

Files to hand in: ./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.
In order to do this, you have to create the logic side of a cash register application (you can imagine that it will be linked to a graphical interface 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 a `int getNumberOf(Class<? extends Food>)` methods to retrieve the number of items of a specific food and two other methods `boolean add(Class<? extends Food>)`, `boolean remove(Class<? extends Food>)` that respectively increment and 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 following 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)**: add a food item to the order and returns wether it has been added or not (depending on the stock status)
  - The added item should be added to the Order and removed from the stock (we don't want two client to take the same item.)
- **boolean removeItem(Food)**: removes the item from the order and put it in the stock. Returns false if the item wasn't in the order.
- **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 (see example).

```java
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.
```