(48024) Applications Programming Lab 5 guide

Lab 5 is due before the next lecture. Because of StuVac, you have two weeks.

Constructors

The Lab5.jar skeleton code includes the tutor's bank example which you should use as a reference.

Customer and Store are similar because they are both clients containing lists.

- A Customer has a list of Accounts
- A Store has a list of Products

Account and Product are similar because they are both suppliers.

- An account contains data and a toString() function
- A product contains data and a toString() function

Code the Product constructor first while referring to the Account constructor. Just as the Account constructor initialises the type field from the parameter with "this.type = type;" your Product constructor should initialise 3 fields from the 3 parameters. ALL 3 fields should be initialised from parameters.

Code the Store constructor while referring to the Customer constructor. Just as the Customer constructor initialises the list of accounts, your Store constructor should initialise the list of products. Remember to create each new Product with 3 parameters. If you forget how to create a new Product, refer back to your Lab 4 solution.

Note that the list of products is not the only field in Store. There is another field. The constructor should initialise both fields. If you forget how to initialise the other field, look at your Lab 4 solution.

Goals

Use the tutor's bank example as a reference. Use your solution to last week's lab as a reference.

As usual, code the main method and use method first.

View stock

Refer to the bank example.

If you get output like this:

```
[Whiteboard Marker -- 85 at $1.50, ……..
```

It means you did this:

```
System.out.println(products);
```

This is wrong because `products` is an ArrayList. The ArrayList class implements its own toString() function that displays all elements on a single line separated by commas and surrounded by a pair of [square brackets]. You don't want that string representation.

Instead, you need to loop over each product in the list and print out each product. You will find an example of this in the bank demo example. Importantly, each time through the loop, the parameter passed into println() should be a single product, not the products list.

Sell

Use last week's solution as a reference:

```
int number = readNumber();
 if (product.has(number)) {
       double sale = product.sell(number);
       cashRegister.add(sale);
 }
 else
       System.out.println("Not enough stock");
```

Last week's store had one product stored in a field named `product`. The solution above refers to that field. This week, there is no `product` field, so the above code won't work immediately. This week, there is instead a list of `products`, and you need to ask the user which product they want to sell. The above code will work if you insert some new code above it. You need to:

1. Read the product name from the user.
2. Look up that product in the list.
3. Print a message to say that you are selling this product.

Restock

No tips provided

Prune

There is a pattern in the lecture slides to remove all matches from a list.

Advanced goals

As you begin to deal with partial matches, you will find you need a new pattern to replace lookup. Lookup can only find one product. Another pattern in the lecture notes can be used to find multiple matches. This pattern replaces the lookup pattern because whenever you lookup a product by name, there is always the possibility that it may return more than one match. It is suggested that you define a function with the header:

```
private LinkedList<Product> products(String partialName)
```

This function says, “Give me a partial name and I will return you a list of products that match”. There is no need to pass the original list of products as a parameter because your function already has direct access to the `products` field.