Applications Programming

Lab - Classes

The process:

1. Read the specification.
2. Analysis and Design:
    1. Identify the classes and fields. Use the NOUNS.
    2. Identify the constructors. Intialise from defaults / read pattern / parameters?
    3. Identify the goals. Use the VERBS.
    4. Spread the goals across classes.
3. Code:
    1. Code the classes and fields
    2. Code the constructors
    3. Code each goal as a procedure, supported by helper functions where appropriate.
4. Submit your project to PLATE

Tutor demo

Main class: `Customer`

Specification: A customer has a savings account and a loan account. Each account has a type (savings/loan) and a balance. The initial balance is read from the user. The customer menu provides the following options:

1. deposit into the savings account
2. withdraw from the savings account
3. transfer money from the savings account into the loan account
4. view the balance of all accounts
5. exit

Sample I/O

```
Initial Savings balance: $1000
Initial Loan balance: $-10000
Customer menu (d/w/t/s/x): ?
Menu options
d = deposit
w = withdraw
t = transfer
s = show
x = exit
Customer menu (d/w/t/s/x): d
Amount to deposit: $50
Customer menu (d/w/t/s/x): s
Savings account has $1,050.00
Loan account has $-10,000.00
Customer menu (d/w/t/s/x): t
Amount to transfer: $100
Customer menu (d/w/t/s/x): s
Savings account has $950.00
Loan account has $-9,900.00
Customer menu (d/w/t/s/x): w
Amount to withdraw: $22
Customer menu (d/w/t/s/x): s
Savings account has $928.00
Loan account has $-9,900.00
```

```
Customer menu (d/w/t/s/x): x
```

The dollar sign is part of the read prompt. This makes negative dollar amounts look strange, but this is for simplicity's sake.

Break it down

Classes:	Customer	Account
Fields:		
Constructors:		
Goals:		

Build it up

Your tutor will code the classes, fields, constructors and goals. Pay attention to the development sequence used by your tutor.

Student specification

Main class: `Store`

A store sells one product. The name of this product is "Sticky tape", it has an initial stock of 200, and a price of $2.99. The store has a cash register which stores all cash from sales of the store's product. The store clerk uses the following menu:

1. sell
2. restock
3. view stock
4. view cash
5. exit

Only sell if the product has enough stock, otherwise give an error.

Sample I/O:

```
Choice (s/r/v/c/x): ?
Menu options
s = sell
r = restock
v = view stock
c = view cash
x = exit
Choice (s/r/v/c/x): v
200 Sticky tape at $2.99
Choice (s/r/v/c/x): c
Cash register: empty
Choice (s/r/v/c/x): s
Number: 10
Choice (s/r/v/c/x): v
190 Sticky tape at $2.99
```

```
Choice (s/r/v/c/x): c
Cash register: $29.90
Choice (s/r/v/c/x): s
Number: 200
Not enough stock
Choice (s/r/v/c/x): r
Number: 50
Choice (s/r/v/c/x): v
240 Sticky tape at $2.99
Choice (s/r/v/c/x): s
Number: 200
Choice (s/r/v/c/x): v
40 Sticky tape at $2.99
Choice (s/r/v/c/x): c
Cash register: $627.90
Choice (s/r/v/c/x): x
```

Break it down

Classes:	Store	Product	CashRegister
Fields:			
Constructors:			
Goals:			

Build it up

When PLATE marks your code, it will look for obviously named classes, fields and methods, with obvious parameters, according to principles taught in the lecture. When not obvious, hints are provided in the skeleton code. When in doubt, submit to PLATE to see the expected class, field and method names, and parameters.

1. Code the classes and fields. Follow good naming conventions for your classes and fields. Submit to PLATE to get initial marks. If necessary, correct your class and field names to match PLATE's.
2. Code the constructors. Initialise the fields from either defaults / read pattern / parameters. Submit to PLATE to get initial marks. If PLATE says "NoSuchMethodException", it means your constructors don't have the same parameters as PLATE's. If necessary, correct your constructors to match PLATE's.
3. Code the goals in the order in which they are tested on PLATE. This is:
    1. The store menu. This involves creating two methods in the Store class. A use() method to implement the menu pattern, and a static main method to create and use the store.
    2. The help menu choice.
    3. View stock. Spread the goal across the classes.
    4. Sell. Spread the goal across the classes.
    5. Restock. Spread the goal across the classes.
    6. View cash. Spread the goal across the classes.

       Hint: the toString() function needs an 'if' statement. See the following example for how to build a string with conditional parts:

```
public String toString() {
  String s = "part a ";
  if (yourcondition)
    s += "yes";
  else
    s += "no";
  s += " part c";
  return s;
}
```

In the above example, if yourcondition is true, then "part a yes part c" is returned. If yourcondition is false, then "part a no part c" is returned.

There are other ways to achieve the same effect.

In your assignment, you may need to use a similar technique. If your string gets too long, you may split the toString() function into helper functions.