|  |  |
| --- | --- |
| Course | Advanced Software Design – CS525 |
| Assignment | Lab 2 |
| Week | 02 |
| Due | Feb 24, 2020 |
| Student name | Quan Hong Doan |
| Student ID | 986956 |

Online version:

<https://github.com/zithiat/asd/blob/master/assignments/Assignment_CS525_Week02_986956.docx>

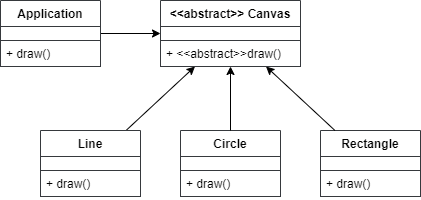
**Problem 1**:

Problem 1 is the question a from the PDF file.

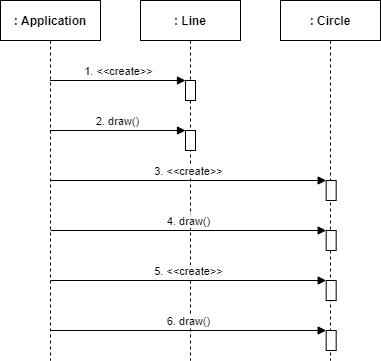
**Answer**:

There’re few ways to design. I start to change the DrawingCanvas class into an abstract class, then 3 other sub-classes implementing the abstract function from the abstract Canvas class.

Therefore, from the Application class, we just need to call draw(), and the JVM will understand to call the function draw() of the relevant class, respectively.



The sequence diagram would be for drawing a line, then 2 circles as below.



**Problem 2**:

Problem 2 is the combination of question b, c, and d.

**Answer**:

Like the problem 1, we need to change some points. We can notice:

* Account has 2 types: Saving and Checking
* Interest is added and calculated based on the account type.

To have minimal changes in our existing code, I would go in this way:

* Add an enum class for account type, which will be called in Account class.
* Add an interface for interest calculation for each account type.
* Then add implementation classes for those calculations
* Then we use a class to run the calculations based on the account type.

Note: I think, this would be a problem later if we want to add more types, and it looks a bit ugly, because calling a lot of classes. To solve this issue, a Factory pattern is usually applied here, but I wouldn’t use here because we are talking about Strategy and Template patterns.

The updated class diagram will be as below.

A screenshot of a cell phone

Description automatically generated

Therefore, the sequence diagram for the interest calculation will be as below. Note: I exclude the part of account type assigned and InterestCalculator object creation.

A screenshot of a cell phone

Description automatically generated

The implementation is following. Source code is available on my [GitHub repository](https://github.com/zithiat/asd/tree/master/labs/codeforlabs/lab2/bank) <https://github.com/zithiat/asd/tree/master/labs/codeforlabs/lab2/bank>.

**AccountService class**

**public** **class** AccountService **implements** IAccountService {

**private** IAccountDAO accountDAO;

**private** InterestCalculator interestCalculator = **new** InterestCalculator();

…

// addInterest with accountNumber

**public** **void** addInterest(**long** accountNumber) {

Account account = accountDAO.loadAccount(accountNumber);

**double** balanceWithInterest = interestCalculator.calculateInterest(account.getType(), account.getBalance());

System.***out***.println(

"Account: " + account.getAccountnumber() +

"\nAccount Holder: " + account.getCustomer().getName() +

"\nType: " + account.getType() +

"\nBalance with interest: " + balanceWithInterest);

}

// addInterest with account object

**public** **void** addInterest(Account account) {

**double** balanceWithInterest = interestCalculator.calculateInterest(account.getType(), account.getBalance());

System.***out***.println(

"Account: " + account.getAccountnumber() +

"\nAccount Holder: " + account.getCustomer().getName() +

"\nType: " + account.getType() +

"\nBalance with interest: " + balanceWithInterest);

}

}

**InterestCalculatorStrategy interface**

**interface** InterestCalculationStrategy {

**double** calculateInterest(**double** accountBalance);

}

**InterestCalculator class**

**public** **class** InterestCalculator {

**private** **final** InterestCalculationStrategy savings = **new** SavingsInterestCalculator();

**private** **final** InterestCalculationStrategy checkings = **new** CheckingsnterestCalculator();

**public** **double** calculateInterest(AccountTypes accountType, **double** accountBalance) {

**switch** (accountType) {

**case** ***SAVINGS***:

**return** savings.calculateInterest(accountBalance);

**case** ***CHECKINGS***:

**return** checkings.calculateInterest(accountBalance);

**default**:

**return** 0;

}

}

}

**SavingsInterestCalculator class**

**public** **class** SavingsInterestCalculator **implements** InterestCalculationStrategy {

@Override

**public** **double** calculateInterest(**double** accountBalance) {

**return** (accountBalance < 1000) ? accountBalance \* 1.01

: ((accountBalance < 5000) ? accountBalance \* 1.02 : accountBalance \* 1.04);

}

}

**CheckingsInterestCalculator class**

**public** **class** CheckingsnterestCalculator **implements** InterestCalculationStrategy {

@Override

**public** **double** calculateInterest(**double** accountBalance) {

**return** (accountBalance < 1000) ? accountBalance \* 1.015 : accountBalance \* 1.025;

}

}

**Problem 3**:

Problem 3 is the combination of question e, and f.

**Answer**:

We will see the payment processor from some points:

* Payment method abstract class to call different payment methods with its validation. So in the future, if we want to add a new payment method, we just need to implement a new method.
* For currency converter, we will extend it into sub-classes for different currencies.

The modified class diagram would be:

A screenshot of a cell phone

Description automatically generated

Then, the sequence diagram is:

A screenshot of a cell phone

Description automatically generated

The implementation is placed in [my GitHub repository](https://github.com/zithiat/asd/tree/master/labs/codeforlabs/lab2/templatemethodlab2) <https://github.com/zithiat/asd/tree/master/labs/codeforlabs/lab2/templatemethodlab2>. The reason is if I copy here, it would be long list for the document.