Yunfan Yang, 30067857

GitLab remote Repositiory: <https://gitlab.cpsc.ucalgary.ca/yunfan.yang/cpsc-501-assignment-01>

Access has been granted to my TA Navid Alipour and Professor Janet Leahy through invitation.

The invitation should work, but please email me to [yunfan.yang1@ucalgary.ca](mailto:yunfan.yang1@ucalgary.ca) in case it does not. :)

# Smell: Undescriptive Names

Branch: undescriptive-names

## Refactoring: Rename Field

Commit: **d1c785f**0f12743aeac621a77bc8e09a9a6404cc4

Junit Test: Test\_Bank.java

File altered: > Bank.java

The private attributes naming accounts1 and accounts2 are undescriptive names, therefore:

* First, renamed them to better names: chequingAccounts and savingsAccounts.
* Second, replace any reference to these attributes with new names. The changes are made in the same file.

The code is tested with Junit Test file as mentioned above. All three tests passed. The changing of variable name does not affect the method external functionality.

It is improved, since the names are descriptive and can be understand by their names, instead of trying to look for where they are used and predicting the purpose of their existence.

Text

Description automatically generated

Text

Description automatically generated

## Refactoring: Replace Magic Number with Symbolic Constants

Commit: **950bd22**5e738c10fac1d5b293fc7f06db40847ce

Junit Test: Test\_Transaction.java

File Altered: > Transaction.java

Transaction type represented by numbers are undescriptive, thus, this refactoring is applied:

* Replace the number with constant variables: REGULAR\_TRANSACTION and E\_TRANSFER.
* And then, replace the arguments in setType() to these constants.

The code is tested with Junit Test file as mentioned above. The tests are focusing on whether changing to constants with affects the type recognition in code. The addition of constant variable name does not affect the method outer functionality.

It is improved, since the numbers are now descriptive with the constant names and can be understand without any code documentation.

Before

Text

Description automatically generated

Text

Description automatically generated

After

Text

Description automatically generated

Text

Description automatically generated

# Smell: Inappropriate Intimacy

Branch: inappropriate-intimacy

## Refactoring: Encapsulate Field

Commit: **11974ea**6ed90098c07f61eae9e1a6afc99fa9428

Junit Test: Test\_ChequingAccount.java and Test\_SavingsAccount.java

File Altered: > ChequingAccount.java, > SavingsAccount.java

The attribute balance in both ChequingAccount and SavingsAccount class has no access modifier, which means one can change the value of balance without going through mutators (which restricts and validate the value), and this is dangerous.

It is identified as Inappropriate Intimacy, thus Encapsulate Field is applied to protect the privacy of this attribute, by adding private keyword to the attribute.

Since the code elsewhere are using getters and setters instead of directly access this attribute, luckily, therefore, adding the private keyword would not bother. The tests are passed.

Now, the attribute is safe and no one can change it without using mutator, the security improved.

Before:

Text

Description automatically generated

Text

Description automatically generated

After:

Text

Description automatically generated

Text

Description automatically generated

# Smell: Duplicated Code

Branch: duplicated-code

## Refactoring: Extract Superclass

Commit: **4549634**b5f3dc84e2af11aba48ca26885b1f6d4b

Junit Test: Test\_ChequingAccount.java and Test\_SavingsAccount.java

File Altered: > ChequingAccount.java, > SavingsAccount, + BankAccount.java

In file ChequingAccount.java and SavingsAccount.java, many lines of codes are similar or identical, this indicates the smell of duplicated code. In order to get rid of, Extract Superclass is applied to these two classes. By making a new abstract super class BankAccount, both ChequingAccount and SavingsAccount can share attributes and methods. Steps as followed:

* Create BankAccount abstract class (it is important to be abstract because a bank account has to be a specific type of account)
* Extends both classes to BankAccount
* Pull up common attributes to BankAccount, along with there getters and setters
* For similar methods, extract the common part and pull up, then using override to remain difference

The tests are passed. By creating a new instance of each type, the tests suggested that the code remained its external functionality.

This is improved by reducing the lines of duplicated code. It would now be easier to add a new type of BankAccount and to maintain the common part of all types of BankAccount.

(Code snippets on the next few pages)

Before

Text

Description automatically generated

Text

Description automatically generated

After

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

## Refactoring: Extract Method

Commit: **b4e6d36**4cecd9e0b9752b474db0d7a0a3b9e3c90

Junit Test: Test\_Bank.java

File Altered: > Bank.java

Since now we have a super class BankAccount to represent both ChequingAccount and SavingsAccount, then we could extract both addAccount into one method:

1. By doing so, the instance attribute of two lists chequingAccounts and savingsAccounts can be combined as one.
2. It further leads to the two for-loops combining into one-loop, since there is only one accounts attribute.

The tests are passed and suggested that the external functionality did not change. The accounts list keeps track of all the accounts as previously.

This is an improvement because the code is more efficient, with less amount of for-loop and less lines of code.

Text

Description automatically generated

Text

Description automatically generated

# Smell: Switch Statement / Large Class

Branch: switch-statement

## Refactoring: Extract Class

Commit: **1abacee**9fc97e0b3001078ef07c1e6fe4cc76e4a

Junit Test: Test\_Transaction.java

File Altered: > Transaction.java, + E\_Transfer.java

In Transaction.java: There are too many if-condition to determine what to do by the transaction type, and the class has too many getters and setters for e-Transfer which Regular transaction would not even use.

1. By applying Extract Class, create a new subclass of Transaction called E\_Transfer
2. Next, pull down all its related attributes and method to E\_Transfer class from Transaction class.
3. Also, remove the type constants since there is hierarchy and the types are no longer needed.
4. Finally, since there is hierarchy for BankAccounts, the parameters in the constructor can be changed to BankAccount.

The tests are passed and suggested that the external functionality did not change. The type detection in the Junit Test is modified to adapt new hierarchy structure. This is improved because the large class is shorted and have two types distinguished with hierarchy.

(Code snippet on the next few pages)

Before

A screenshot of a computer

Description automatically generated with medium confidence

After

Text

Description automatically generated

Text

Description automatically generated