Classes & Encapsulation

Homework 2

String formatting

- Use string formatting instead of multiple string chunks and + operators.
  - ex) String format contains format specifiers like `%s`, `%d`, etc. You should use a right format specifier to print a variable.

```
String str = "id: " + "2015-22222" + ", name: " + "Jane" + ", age: " + 23;
String formattedString = String.format("id: %s %s %d", "2018-11111", "Jack", 23);
```

    - use `%s` to print a string variable, use `%d` to int or long, and `%f` to float.
    - Other format specifiers: https://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html
- To print formatted string to console, use `System.out.printf()` instead of `System.out.println()`.
  - You need to add newline character `'\n'` at the end of the string format, because printf doesn't break line.

Homework 2-1 - Bank Package

- This homework will help you to understand the concept of encapsulation.
- Think about the overall program structure, and usages of access modifiers.
- Objective: Implement `Bank`, `BankAccount`, `Event` classes.

File structure is like this:

```
├── Test.java
├── bank
│   ├── Bank.java
│   ├── BankAccount.java
│   └── Event.java
└── toth
    ├── Toth.java
    └── TothAccount.java
```

Test Code

Main Function

```
Bank bank = new Bank();
int janeAccountId = bank.createAccount("Jane", "1234asdf"),
    evaAccountId = bank.createAccount("Eva", "5678ghkj", 1000);

bank.deposit(janeAccountId, "abcdefg", 100);
bank.deposit(janeAccountId, "1234asdf", 500);

bank.withdraw(janeAccountId, "abcdefg", 100);
bank.withdraw(janeAccountId, "1234asdf", 1000);
bank.withdraw(janeAccountId, "1234asdf", 300);

bank.transfer(evaAccountId, "abcdefg", 3, 300);
bank.transfer(evaAccountId, "5678ghkj", 3, 300);
bank.transfer(evaAccountId, "5678ghkj", janeAccountId, 300);

bank.printEvents(janeAccountId, "1234asdf", 3);
bank.printEvents(evaAccountId, "5678ghkj", 3);
bank.printTransactions(janeAccountId, "1234asdf", 3);
```

Test Code Output

```
[CREATE ACCOUNT] owner: Jane, balance: 0
[CREATE ACCOUNT] owner: Eva, balance: 1000
[DEPOSIT] owner: Jane, amount: 500, balance: 500
[ERROR] There is not enough balance
[WITHDRAW] owner: Jane, amount: 300, balance: 200
[ERROR] There is no account with id 3
[TRANSFER] Eva => Jane, amount: 300, balance: 700
<Events> owner: Jane
    (Receive) source: Eva, amount: 300, balance: 500
    (Withdraw) amount: 300, balance: 200
    (Deposit) amount: 500, balance: 500
<Events> owner: Eva
    (Send) target: Jane, amount: 300, balance: 700
<Transactions> owner: Jane
    (Receive) source: Eva, amount: 300, balance: 500
```

`bank` Package

- `Bank` class represents banks, `BankAccount` class represents bank accounts, and `Event` class represents account events.
- `Bank` has an array of bank accounts. A bank can store up to 100 accounts.
- `Bank` class is an entry point to bank package.
  - You can `createAccount`, `deposit`, `withdraw`, and `transfer` with a `Bank` object.
  - `Bank` class is responsible for all console prints.
- `Bank` object modify data of `BankAccount` objects when the `Bank` object call one of the above methods.
- `BankAccount` and `Event` is not exposed to outside of the bank package.
- `BankAccount` has account id, owner's name, current balance, and event histories.
- `BankAccount` create an `Event` object and add it to its event array. The event array can store up to 100 events.
- `Event` object simply saves deposit, withdraw, and transfer information.
- You can add/modify default/private members, but do not add/modify public members.
- No need to consider not-mentioned exception cases.

Bank Class Specifications

- `public int createAccount(String owner, String password)`
  `public int createAccount(String owner, String password, int balance)`
  - Create an account with a name of the account owner, password, and initial balance. If the balance is not passed, set the initial balance as 0. Return the account ID. Print a message.
  - create message: “[CREATE ACCOUNT] owner: <OWNER>, balance: <BALANCE>”
- `public void deposit(int accountId, String password,int amount)`
  - Find the proper account with the id, check the password, add amount to the account, and print deposit message.
  - message: “[DEPOSIT] owner: <OWNER>, amount: <AMOUNT>, balance: <BALANCE>”
- `public void withdraw(int accountId, String password,int amount)`
  - Find the proper account with the id, check the password, subtract amount from the account, and print withdraw message.
  - If there is not enough balance to withdraw, prompt error message.
  - withdraw message: “[WITHDRAW] owner: <OWNER>, amount: <AMOUNT>, balance: <BALANCE>”
  - error message: “[ERROR] There is not enough balance”

`Bank` Class Specifications

- `public void transfer(int accountId, String password, int targetAccountId, int amount)`
  - Find the proper account with the id, check the password, transfer amount to other account with target account id, and print transfer message.
  - transfer message: “ [TRANSFER] <SOURCE OWNER> => <TARGET OWNER>, amount: <amount>, balance: <BALANCE>”
  - If there is no account with the given id, prompt error message:
    - error message: “ERROR: There is no account with id <TARGET ACCOUNT ID>”
  - If there is not enough balance to withdraw, prompt error message.
    - error message: “ERROR: There is not enough balance”

`Bank` Class Specifications

- `public void printEvents(int accountId, String password, int maxShow)`
  - Find the proper account with the id, check the password, and print at most maxShow events in the lastest order. If the number of events is smaller than maxShow, just print existing transactions.
- `public void printTransactions(int accountId, String password, int maxShow)`
  - Print at most maxShow transactions in lastest order. If the number of transactions is smaller than maxShow, just print existing transactions.

- Common: If the password is not authenticated during the password authentication process, an error message is displayed.
  - message: “ERROR: Wrong account id or password”

- `public boolean authenticate(int accountId, String password)`
  - authentication api for external libraries.
  - (Only for this method) Don’t print message.

`BankAccount` Class Specifications

- `BankAccount` has an event array to store event histories.
- `boolean authenticate(String password)`
  - Return true if and only if when the password is authenticated.
- `void deposit(int amount)`
  - Add the balance by the amount, and add `DepositEvent` to the event array.
- `boolean withdraw(int amount)`
  - If current balance is larger than the withdrawing amount, subtract the balance by the amount, add `WithdrawEvent` to the event arrray, and return true. Otherwise, return false.
- `void receive(int amount, String sourceOwner)`
  - Add the balance by the amount, and add `ReceiveEvent` to the event array.
- `boolean send(int amount, String targetOwner)`
  - If current balance is larger than the withdrawing amount, subtract the balance by the amount, add `SendEvent` to the event arrray, and return true. Otherwise, return false.

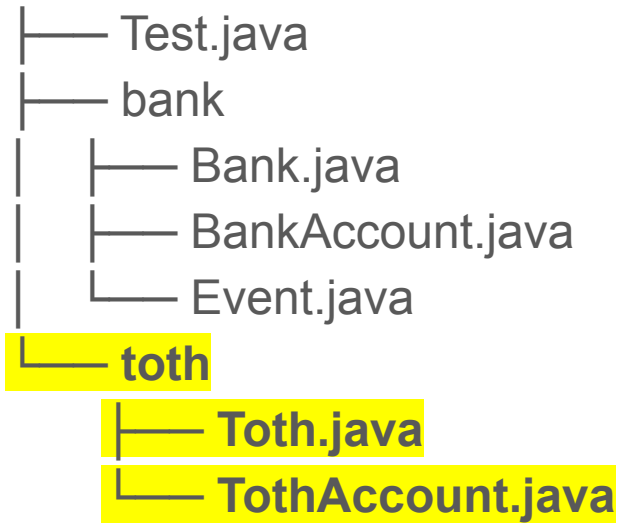
`Event` Class Specifications

- There are four types of `Event` classes, `DepositEvent`, `WithdrawEvent`, `SendEvent`, and `ReceiveEvent`.
- All four event classes has different format string.
- You can override `toString` method to define different representation format.
  `DepositEvent`: “(Deposit), amount: <AMOUNT>, balance:<BALANCE>”
  `WithdrawEvent`: “(Withdraw), amount: <AMOUNT>, balance: <BALANCE>”
  `SendEvent`: “(Send) target: <TARGET OWNER>, amount: <AMOUNT>, balance: <BALANCE>”
  `ReceiveEvent`: “(Receive) target: <SOURCE OWNER>, amount: <AMOUNT>, balance: <BALANCE>”

Homework 2-2 Toth Package

- To call bank methods, you need to pass an account id and the passrod, which is quite annoying.
- A new generation banking service Toth appeared.
- When you pass an account id and the password to Toth, you can get a TothAccount, and you can do banking without your account id or password.

File structure



Test Code

Main Function

```
TothAccount wrongIdTothAcount =
        Toth.createTothAccount(100, "1234asdf", bank),
    wrongPwdTothAccount =
        Toth.createTothAccount(janeAccountId, "abcdefg", bank),
    janeTothAccount =
        Toth.createTothAccount(janeAccountId,
                               "1234asdf", bank);
janeTothAccount.deposit(200);
janeTothAccount.withdraw(100);
janeTothAccount.transfer(evaAccountId, 100);
```

Test Code Output

Output

```
TOTH ERROR: Wrong account id or password
TOTH ERROR: Wrong account id or password
[DEPOSIT] owner: Jane, amount: 200, balance: 700
[WITHDRAW] owner: Jane, amount: 100, balance: 600
[TRANSFER] Jane => Eva, amount: 100, balance: 500
```

`Toth` Class Specification

- `public static TothAccount createAccount(int accountId, String password, Bank bank)`
  - authenticate input `accountId` and `password`, and return a new `TothAccount` object. If the authentication failed, prompt an error message and return `null`.
  - message: “[TOTH ERROR] Wrong account id or password”

`TothAccount` Class Specifications

- `public void deposit(int amount)`
  `public void withdraw(int amount)`
  `public void transfer(int targetAccountId, int amount)`
  - Deposit/withdraw/transfer with the real bank account.