Assignment 2

Instructor: Musard Balliu

September, 2016

In this assignment, you will write an object-oriented application for a simple online banking program. The application allows users (customers) to create user accounts, login as users, list the users of the online bank, or just terminate the application. Upon login, the program should allow a user to check their balance, deposit a given amount of money, withdraw a given amount of money, transfer money to another user account and list the transactions performed by the user.

The task

In this assignment, you are given the specification of the online banking application which you need to implement. The application consists of the following classes:

- A `Bank` class representing users and transactions of the online bank
- A `Account` class representing a unique account id and the account balance
- A `User` class representing the username, password and Account for a given user (customer)
- A `Transaction` class representing a log of transactions/operations performed by a user
- A `BankMain` class representing the interface between the user(s) and the online banking application

The specification of each method is provided as commented java files included in the following sections and available for download from
http://www.cse.chalmers.se/ musard/teaching/programming2016/dit948-2016/Assign2_2016.zip.

The interface, example session

The class `BankMain` has a `main` method which, when executed, should lead to an interactive user session of the following kind:

```
Welcome to the DIT948 online bank

Main menu
1: Create user account
2: Login as user
3: List users
4: Exit
Enter a choice: 1
Please enter username
musard
Please enter password
123
New user created

Main menu
1: Create user account
2: Login as user
3: List users
4: Exit
Enter a choice: 1
Please enter username
omar
Please enter password
456
New User created

Main menu
1: Create user account
2: Login as user
3: List users
4: Exit
Enter a choice: 2
Please enter username
musard
Please enter password
123
```

```
Successful user login

SubMenu
1: Check balance
2: Deposit
3: Withdraw
4: List of transactions
5: Transfer to other account
6: Close user session
Enter a choice: 2
Enter the deposit:400

SubMenu
1: Check balance
2: Deposit
3: Withdraw
4: List of transactions
5: Transfer to other account
6: Close user session
Enter a choice: 3
Enter the amount to withdraw:100

SubMenu
1: Check balance
2: Deposit
3: Withdraw
4: List of transactions
5: Transfer to other account
6: Close user session
Enter a choice: 1
Your balance is 300

SubMenu
1: Check balance
2: Deposit
3: Withdraw
4: List of transactions
5: Transfer to other account
6: Close user session
Enter a choice: 5
Enter the username of the recepient: omar
```

```
Enter amount to transfer: 200

SubMenu
1: Check balance
2: Deposit
3: Withdraw
4: List of transactions
5: Transfer to other account
6: Close user session
Enter a choice: 1
Your balance is 100

SubMenu
1: Check balance
2: Deposit
3: Withdraw
4: List of transactions
5: Transfer to other account
6: Close user session
Enter a choice: 4
Type: Create

Type: Deposit 400

Type: Withdraw 100

Type: Transfered 200 to omar


SubMenu
1: Check balance
2: Deposit
3: Withdraw
4: List of transactions
5: Transfer to other account
6: Close user session
Enter a choice: 6

Main menu
1: Create user account
2: Login as user
```

```
3: List users
4: Exit
Enter a choice:
```

Additional test cases are available in the file Testcases.txt.

The classes

Class BankMain

```
import static dit948.SimpleIO.*;

/**
 * Class representing a online banking application for the second assignment of
 * DIT948, 2016 edition. This is the main class for the application, interacting
 * with the user, creating accounts and performing various transactions More
 * information about the interface can be found in Testcases.txt
 */

public class BankMain {

public static void main(String[] args) {

println("Welcome to the DIT948 online bank");

Bank bank = new Bank();

// Implement here the interaction between the user and the
// online banking system.

// As described in the Assignment text, the interface contains a Main
// menu, and a SubMenu. The program should return to the Main Menu
// whenever the user exits the SubMenu, without terminating.
}
}
```

Class Bank

```
/**
 * This class implements the data structures of the bank. An array of users and
 * array of transactions are used to keep track of users and transactions
 * Note: you are allowed to use ArrayLists, but you don't need to
 */

public class Bank {

// Declaration of class variables,
// id: a public static integer representing a user id, initially 0
// users: a private array of User objects
// transactions: a public static array of Transaction objects
// Declare your own variables, if needed

// code here

// default constructor
// you can assume there won't be more than 100 users
// and 100 transactions
public Bank() {
// code here
}

/**
 * Checks whether a user is present in the User array, given the username
 * and the password
 *
 * @param u  username
 * @param p  password
 * @return true or false, accordingly
 */

public boolean existsUserByUsernamePwd(String u, String p) {
// code here
}

/**
 * String representation of the list of users of the
```

```
 * online bank
 */

public String toString() {
// code here
}

/**
 * Returns the User associated with a username and a password
 *
 * @param u  username
 * @param p  password
 * @return a User object or null
 */

public User getUserFromUsrPwd(String u, String p) {
// code here
}

/**
 * Returns the User object associated with a given username
 *
 * @param u username
 * @return the User with username u, if present in the array of
 * Users; null otherwise
 */

public User getUserByUsr(String u) {
// code here
}

/**
 * Prompt the user to enter a username and a password from the console. Then
 * use the chosen username and password to create a new user and add it to
 * the array of users. Hint: Each user must be assigned a new id.
 */

public void addUser() {
// code here
}
}
```

Class `Account`

```
/**
 * This class implements a user account, represented by a user id
 * (unique for each account) and an amount (or balance)
 */

public class Account {

// private instance variables
// id: an integer representing the account id
// amount: the account balance


// constructor with parameters

public Account(int amount, int id) {
// code here
}

/**
 * Withdraw a given amount of money from the account and record the
 * transaction
 * @param deductAmount the amount to withdraw
 * @return true if the withdraw succeeds, false otherwise
 */

public boolean withdraw(int deductAmount){
// code here
}

/**
 * Deposit a given amount to the account and
 * record the transaction
 * @param addAmount amount to be deposited
 * @return true
 */

public boolean deposit(int addAmount){
// code here
```

```
}

/**
 * Transfer money from this account to user B, and
 * record the transaction
 * @param B user to transfer money to
 * @param amountToTransfer the amount to transfer
 * @return true if the transfer is possible, false otherwise
 */

public boolean transferMoney(User B,int amountToTransfer){
// code here
}

/**
 * Returns the balance and records the transaction
 * @return amount
 */
public int getAmount() {
// here code
}

/**
 * Returns the account id
 * @return id
 */
public int getId(){
// code here
}
}
```

Class `User`

```
/**
 * This class represents the data for a user of the bank
 */

public class User {

// private instance variables
```

```
// name: a String representing the username
// password: a String representing the password
// account: an Account representing the account

// Constructor with parameters
// Create a new account and record the "transaction"
public User(String name, String password, int accountId) {
// code here
}

/**
 * Returns the account id
 * @return id
 */

public int getId() {
// code here
}

/**
 * Returns the username
 * @return name
 */

public String getUsername() {
// code here
}

/**
 * Returns the password
 * @return password
 */
public String getPassword() {
// code here
}

/**
 * Returns the user Account
 * @return account
 */
```

```
public Account getAccount() {
// code here
}
}
```

Class `Transaction`

```
/**
 * This class implements a transaction performed by the bank users
 */

public class Transaction {
// private instance variables
// type: a String to record the type of transaction
// account: an Account used by the transaction

// constructor with parameters
public Transaction(Account account, String type) {
// code here
}

/**
 * Returns the Account object
 * @return account
 */
public Account getAccount() {
// code here
}

/**
 * String representation of a transaction
 */
public String toString() {
// code here
}
}
```

Remarks and Notation

You are allowed to used methods from java.util.ArrayList, however, they are not strictly needed, and usual arrays, as studied in the lectures, would work just fine. In that case, you can assume a maximal length of 100 elements both for the array of transactions and the array of users.

You can assume that users are uniquely identified by their usernames, and no pair of users share the same username

Use the method equals() to check whether or not two strings are the same

The program must return to the Main Menu and not terminate, whenever the user exits the SubMenu (see Testcases.txt)

It is sufficient to just terminate the program with an informative message whenever an error occurs, for example, a user trying to withdraw or transfer more money than his balance, ecc. However, fancy error handling is very much appreciated.

Additional cases of interaction between the users and the banking system can be found in the file Testcases.txt

Grading

- To get a G you can ignore the Transaction class and skip the recording/logging of any user transactions in the corresponding methods.
- To get a VG it is necessary implement all tasks in the assignment.

  Note that these requirements are necessary, but not sufficient, to get the respective grades (see Administrative matters).

Administrative matters

Strive for readable code with appropriate comments! While the ultimate test of a program is that it does what it is supposed to do, **we should be able to read the program and understand it**.

The assignment is to be completed in groups of two students. (Groups of different size should first be agreed with the course instructor.)

Solutions must be uploaded to the GUL system **by 23:55 on October 5**.

Please note: **The submission must be made via GUL. It's no good sending it to me via email, either before or after the deadline!**

Only Java source files should be uploaded, no class files. Your Java files should be put in a zip-archive with the following name:

```
assign2_author1_author2.zip
```

where author1, author2 are the surnames (family names) of the group members. You must also supply a README-file (README.txt) containing the names of the authors and their social security numbers, together with a brief statement about each author's contribution. For example, "author1 has been responsible for user input and author2 for the program logic". A statement such as "All authors have contributed equally to all aspects of the program" is not acceptable.

Note that **each author must submit individually via the GUL** (even though it is the same program!). Only students who submit via the GUL can be graded. All comments etc. will be posted on the course web page.

Once the deadline has passed, each group **must explain solutions to the TAs during the first Thursday supervision session**. Exceptions are to be agreed with the instructor and the TAs. Group members are expected to know and explain all parts of the code despite their statement of contribution.