Lab 04 Specification – Inheritance and Polymorphism
Due (via Bitbucket) Wednesday, 11 October 2017
50 points

Lab Goals

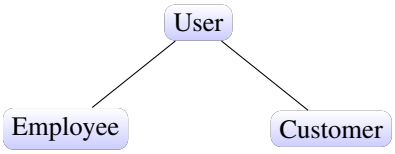
- To extend the ATM simulator that you have developed in the last lab.
- To create a bank simulator and get hands on experience in implementing different user levels.
- To get implicit learning on how inheritance and polymorphism works.

Assignment Details

Implement a Bank Simulator with the following core functionalities. Additionally the functionalities of the ATM Simulator from lab 03 should be present (like customer login, transactions, etc); these can be rewritten if desired. The lessons you learned creating the ATM Simulator will be useful here. You must use at least one inheritance hierarchy, one overloaded method, and one overridden method. Ensure that you do not modify the lab 03 folder, but instead create a new folder for lab 04 in your Bitbucket repository.

User Redirection Module: (10 points)

The users of our Bank Simulator are categorized into Employees and Customers.



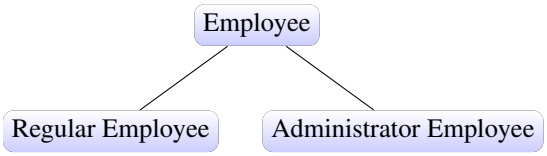
To start a "Main Module" should be created that can start all the other modules when needed. Then the Bank Simulator needs to prompt the user like below:

```
Are you a bank customer (enter 1) or an employee (enter 2)?
>>
```

If the user enters 1, then redirect to Customer login (refer to lab 03), and if the user enter 2, then redirect to Employee login. As a hint, you can think of a User class as being the parent of two child classes: Employee and Customer.

Employee Login Module: (25 points)

Employees at the bank are categorized into Regular Employees or Administrator Employees.



This module is similar to the Customer Login Module from lab 03. Use `employees.txt` to store employee logins and refer to the stored information to determine what kind of employee they are after login. Follow the below requirements in creating this module.

Only an adminstrator employee can add other employees. The initial administrator's login credentials (userid: admin, password: admin123) are stored in the `employees.txt` file. You must create this inital file using the below sample. The file will then only be modified by administrators within the Bank Simulator.

```
admin;admin123;admin
employee;password;regular
```

An administrator should login using the admin default credentials shown above. After login success the following options should be provided to the administrator, and they could change the default password or create a new admin account:

```
1) Manage Accounts
2) Manage Customers
3) Manage Employees
4) Exit
```

These options (besides exit) should each link to another menu which allows the administrator to perform the following actions:

- Manage Accounts: Provide an option to delete or update existing accounts, along with an option to add new accounts. The list of accounts are managed through the `accounts.txt` file. (Refer to lab 03 for format)
- Manage Customers: Provide an option to delete or update existing customer logins, along with an option to add new customers. The list of customers are managed through the `customers.txt` file. (Refer to lab 03)
- Manage Employee: Provide an option to delete or update existing employee logins, along with an option to add new employees. The list of employees are managed through the `employees.txt` file. (Refer to sample employees.txt file above)

After a regular employee logs in, the following options should be provided:

```
1) Add Account
2) Perform Deposit
3) Perform Withdrawal
4) Exit
```

These options (besides exit) should each link to functionality that fulfills the following descriptions:

- Add Account: prompt for necessary information and add a line with those details to the `accounts.txt` file.
- Perform Deposit: prompt for information (like customer id, etc) and execute the transaction. (Refer lab 03)
- Perform Withdrawal: prompt for information (like customer id, etc) and execute the transaction. (Refer to lab 03)

Transaction History Module: (15 points)

After every transaction (withdrawal or deposit, whether initiated by the customer or an employee), store the transaction details into the `transactions.txt` file by adding a line that includes `customerId`, `transactionId`, `transactionDateTime`, `transactionType`, and `amount`. `transactionId` is an unique number that is generated for every transaction. For instance, get the number of currently stored transactions

and increment it for the next transaction. The `transactionType` is the type of the transaction; for example `deposit` or `withdrawal`. The `transactionDateTime` is the current date and time (24 hr, no AM/PM) when the transaction occurs and is formatted like below:

```
mm/dd/yyyy hh:mm:ss
```

A sample of how to format a Date object to get a String of this form, along with how to parse a String of this form back into a Date object is shown below.

```
//create a formatter (check javadoc for parameters)
SimpleDateFormat formatter = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss");

//create a Date object that references now
Date now = new Date();

//format date object into string
String datetime = formatter.format(now);

//parse the string into a Date object
Date parsed = formatter.parse(datetime);
```

Submission Details

For this assignment, please submit the following to your `cs112f2017-<your user name>` repository (and ensure that the instructor has access to your repository):

1. Commented source code from the “User Redirection Module”.
2. Commented source code from the “Employee Login Module”.
3. Commented source code from the “Transaction History Module”.
4. Commented source code from the “Main Module” that allows execution of the entire program.
5. You are required to use inheritance (at least one hierarchy) and polymorphism (at least one overloaded and one overridden method). Failure to do so will result in only partial credit. Use of abstract classes and interfaces in your lab assignment is not required but is encouraged.