The University of Texas at El Paso
Department of Computer Science
CS 3331 – Advanced Object-Oriented Programming
Instructor: Dr. Bhanukiran Gurijala
Fall 2024

Project Part 2

Academic Integrity Statement:
This work is to be done as a team. It is not permitted to share, reproduce, or alter any part of this assignment for any purpose. Students are not allowed to share code, upload this assignment online in any form, or view/receive/modify code written by anyone else. This assignment is part of an academic course at The University of Texas at El Paso and a grade will be assigned for the work produced individually by the student.

Instructions:
Your code must be written in Java. In the comment heading of your source code, you should write the names of all team member(s) who worked on this file, date last changed, course, instructor, project part 2, project description, and honesty statement. The honesty statement must state that you completed this work entirely on your own without any outside sources including peers outside of your team, experts, online sources, or the like. Only assistance from the instructor, TA, or IA will be permitted. Generate Javadoc for your complete code.

Scenario:
You have set up the foundation for your bank. You now have many customers who are using the bank.

Part A:
Read the requirements described in Part B to complete Part A. Part A must be completed before implementing the requirements in Part B

1. Write a refactored level II UML Use Case Diagram for your system with at least the following:
    a. 4 Use Cases.
    b. 3 Actors.
    c. 2 extends.

    d. 3 includes.
2. Write 2 additional use case scenarios based on Part B
3. Write a refactored UML Class Diagram to structure your code using the classes, requirements, and concepts described in Part B

<u>Part B</u>

1. Refactor your existing code.
    a. Your code should handle all functionality from Project Part 1 (PA1)
    b. Fix anything that should be corrected.
        i. Appropriate data structures
        ii. Appropriate use of objects
        iii. Relationships between objects
        iv. Algorithms and complexity
2. Handle receiving an input file (CS 3331 – Bank Users.csv) that does not have the information listed in standard column orders as in the previous assignment. (For instance, Identification Number is not column 1, First Name is not column 2, etc.).
3. Create and use at least one interface in your system.
    a. The interface should be used in at least 2 different classes.
    b. It may be necessary to refactor your code.
    c. Consider using a design pattern to meet this requirement.
4. Use at least one design pattern as part of your refactoring process.
    a. Select one of the design patterns discussed in class or use one that you have researched on your own.
    b. Discuss its use in the lab report.
5. Add new bank user functionality (create users from console)
    a. Users should complete all fields provided in the CSV: Name, DOB, Address, City, State, Zip, and Phone Number.
    b. Users should be given a Savings, Checking, and Credit Card Account
        i. User ID numbers should be automatically generated (Identification numbers should increment by one from the last user created).
        ii. Account numbers should be generated automatically.
        iii. Users should be given a credit limit (randomly from the range) based on their credit scores using the table given below:

Credit Score Range	Random Credit Limit
<= 580	\$100 – \$699
581 – 669	\$700 – \$4999
670 – 739	\$5000 – \$7499
740 – 799	\$7500 – \$15999

>= 800	$16000 – $25000

6. Handle users with the same first name or same last name (assume that users will have either the same first names or the same last names but not both)
   a. Handle looking up users by name and returning the correct user.
7. Create a User Transactions file (txt) with the following information:
   a. Account Information
   b. Starting Balance (You can assume beginning of session)
   c. Ending Balance (At the time requested)
   d. All transactions for that person
   e. Date of Statement (date of running the code)
8. Extend Bank Manager functionality.
   a. Keep all functionality from PA1.
   b. Transaction reader
      i. Handle all transactions from the input file (Transactions.csv). Format:

From First Name	From Last Name	From Where	Action	To First Name	To Last Name	To Where	Action Amount
Mickey	Mouse	Checking	pays	Donald	Duck	Checking	100

Action Words (not changing):

1. Pays
   a. A User pays to another user.
2. Transfers
   a. A User transfers from one account to another (savings -> checking, etc.)
3. Inquires
   a. Inquires only have "From" components (on a specific account)
4. Withdraws
   a. Withdraws only has "From" components (on a specific account)
5. Deposits
   a. Deposits only have "To" components and amounts (like cash)

      ii. Your code should handle the input of the file. For example, when it reads "pays", your system should call the appropriate methods to pay someone.
      iii. Successfully complete all transactions from the input file.
      iv. Consider the possibility of having transactions that will fail (i.e., withdrawing more than the account balance)

1. Write a message that this transaction failed in the console along with the reason (why).
2. Continue working on the remaining transactions.

c. Generate a Bank Statement file for a specific user.
   i. Choose a user by name/ID.
   ii. The formatting is up to you (Google sample bank statements for inspiration). Does not have to be fancy, but functional.
   iii. All information about the user should be included in a statement (Name, Address, Phone Number, etc.)
   iv. All transactions should be written (at least for a particular session of running the code).
   v. Only the Bank Manager can generate a Bank Statement.

9. The user can exit the program by writing “EXIT” while in the main menu only. When the user exits the program
   a. Write a new updated Bank Users sheet (new CSV file – do not overwrite the original input) with the updated values.
   b. This new CSV file may be overwritten each time the code is run but do not overwrite the initial (original) input file.
10. Handle all exceptions appropriately:
   a. Users cannot have negative money in an account.
   b. A user cannot pay themselves. For example, Mickey Mouse should not be able to “transfer” or “pay” from his checking account to his checking account.
   c. A credit card should not be able to accept more money than the balance. For example, if the outstanding balance on a credit card is $20, the system should not let the user pay or transfer $20.01 or more. This case should fail, and the transaction should not take place.
   d. The outcome (success or failure) of all transactions should be notified to the user.
   e. All other common exceptions should be handled.
11. Create automated test cases and test suites using JUnit.
   a. Create JUnit tests for at least 5 methods.
   b. Create a JUnit test suite to run all test case files at once (created in the previous step, a).
12. Write Javadoc for your system.
13. Write a lab report describing your work (template provided)
   a. Any assumptions made should be precisely commented on in the source code and described in the lab report.

   b. The lab report should contain sample screenshots of the program being run in different circumstances, including successful and failing changes.
14. Complete an individual code review on your code (template provided)
15. Schedule a demo with the TA/IA
16. **If the submission is past the deadline** Your report must have an additional section entitled “Why I submitted late”. In that section, explain the reason why your submission was late. (Note: you will still be penalized by the typical late penalty)

<u>Deadlines:</u>

<u>Check-in – November 3, 2024, at 11:59 pm:</u>

1. UML Class Diagram Progress (pdf)
2. UML Use Case Diagram Progress (.pdf)
3. UML Use Case Scenario(s) (pdf)
4. Current Progress Source Code (java)
5. JUnit test case for at least one function

For each item (1-4)

a. Does not have to be complete.
b. There should be a significant amount of work done (as determined by the instructional team).
c. Demo with TA/IA after 11/3. All the check-in demos should be completed by 11/6. The TA/IA will review progress and provide informal feedback. The check-in will be worth 50 points and will count towards your project grade. This is to ensure that all teams are making timely progress towards completion of the project.
d. The TA/IA working with the team will announce their availability for check-ins on Blackboard.

<u>Final Deliverables – November 12, 2024, at 11:59 pm:</u>

1. UML Class Diagram (pdf)
2. UML Use Case Diagram (pdf)
3. UML Use Case Scenarios (pdf)
4. Lab report (.pdf file)
5. Source code (java files)
6. JUnit files (.java files) – you can include them along with your zipped source code in a separate package (folder)
7. Javadoc (entire doc folder)
8. Updated Bank Users Sheet (.csv)

9. User Transaction file for three customers (you, and two of your team members) (txt)
10. Log (txt)