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

Programming Assignment 3

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 permitted to share code, upload this assignment online in any form, or view/receive/modifying 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 team member(s) who worked on this file, date last changed, course, instructor, programming assignment 3, 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.

1. Fix any issues from previous parts.
    a. Does your bank system fulfill all requirements?
    b. Are you using appropriate data structures?
    c. Is your bank system robust?
    d. Have you tested your bank extensively?
    e. Are you using a design pattern?
    f. Did you implement your design pattern correctly?
    g. Does your code follow the open-closed principle?

2. Allow for user interaction
    a. Ask if the user is a “Customer” (not Bank Manager i.e. Mickey Mouse, Donald Duck)
        i. If the user is a “Customer", then the user will be prompted to log in (see requirement 3 below).
        ii. Once logged in, the “Customer” should be able to do the following transactions (all functionality from parts 1 & 2):
            1. Inquire about balance
            2. Deposit money into an account
            3. Withdraw money from an account
            4. Transfer money between accounts
            5. Pay money to some other customers’ accounts
            6. User Transactions file
    b. Ask if the user is a “Bank Manager”, then the user should be able to do the following (all functionality from parts 1 & 2):
        i. Add new user functionality (with Saving, Checking, and Credit Card Accounts)
        ii. Inquire about the chosen account by name
        iii. Inquire about the chosen account by type/number
        iv. Transaction reader from file
        v. Generate a Bank Statement
3. Add a login prompt for users
    a. When selecting “Mickey Mouse” a prompt for a password should be given
        i. Does not have to be salted/hashed. Simple functionality (if correct, enter otherwise do not let them have access to information on the account)
    b. Bank Managers do not need a login prompt
4. Refactor the level II UML Use Case Diagram if needed (based on feedback) with at least the following:
    a. 5 Major Use Cases.
    b. 3 Actors.
    c. 2 extends.
    d. 3 includes.
    e. Write in complete detail.
        i. Add all actors and use cases for your system.
        ii. The model must be unambiguous and consistent.
        iii. Includes and extends relationships should be correct.
        iv. Follow the correct syntax of a use-case diagram.
5. Write a UML State Diagram for the following:

    a. Main Menu to complete Depositing Money into an account.
    b. Main Menu to complete Add New Bank User functionality.

  Write in complete detail:
    a. Include all the events of your system.
    b. Follow correct syntax for state diagrams.
6. Write refactored (if needed) Use Case Scenarios (2 from the previous part) for the system in complete detail.
    a. The scenario should be unambiguous.
    b. The scenario should cover all steps of that use case.
    c. There should be interactions between the actor and the system.
    d. Follow correct syntax for use-case scenarios.
7. Write a refactored UML Class Diagram to structure your code using the classes, requirements, and concepts described in the project parts.
    a. Write in complete detail.
        i. Add all classes, attributes, and methods from your code.
        ii. The model must be unambiguous and consistent.
        iii. All relationships and multiplicities should be correct.
        iv. Follow the syntax of a class diagram.
8. Refactor your existing code to be a singular cohesive program.
    a. Remove redundant code.
    b. Add comments to your code at all levels.
    c. Your code should handle all functionality from previous parts (Project Part 1 and Project Part 2).
    d. Verify that you are properly using the concepts of Object-Oriented Programming. Verify the following:
        i. Proper usage of abstraction.
        ii. Proper usage and implementation of inheritance.
        iii. Proper usage and implementation of aggregation and/or composition.
        iv. Proper usage and implementation of associations
        v. Proper creation of objects
        vi. Proper and appropriate interaction among objects.
    e. Verify methods are written correctly.
        i. Consider method overriding wherever appropriate.
        ii. Consider method overloading wherever appropriate.
        iii. Proper and consistent usage of naming conventions.
        iv. Appropriate logic inside the methods.
    f. Readability/Style
        i. Ensure that you are using a standard style for coding throughout.

ii. Conduct a code review of the entire source code.
iii. Follow industry-level coding standards. The following is a standard coding style used by Google:
1. https://google.github.io/styleguide/javaguide.html

g. Fix anything that should be corrected.
i. Appropriate data structures
ii. Appropriate use of objects
iii. Relationships between objects
iv. Algorithms and complexity

9. Use an additional 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.
c. At least 2 design patterns in total should be used in your project.

10. Make prompts self-explanatory.
a. Users should not need any background knowledge to use your bank system.
b. Users should intuitively be able to use the system. For example, if the prompt says "Amount", consider making it more intuitive and user-friendly such as "Please enter an amount to deposit/withdraw".
c. Think about how you may use a bank system.

11. Handle all exceptions appropriately.
a. Consider using exception handling that is separate from normal programming tasks as discussed in class.
b. Create at least one user-defined exception and use it.

12. Create a JUnit test suite.
a. Create JUnit tests for at least 5 other functions.
b. Create a JUnit test suite to run all test case files at once (created in the previous step, a).

13. Write Javadoc for your system.

14. 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.
c. Clearly specify:
i. How did you come up with JUnit test cases? Is it a black-box or white-box approach?

ii. What are various test cases considered for testing the correctness of chosen functions?
iii. Why do you think those test cases are enough to test the correctness of the functions?
iv. A screenshot of running your JUnit test cases.

15. Complete an individual code review on your code (template provided)
16. Schedule a demo with the TA/IA
    a. If you are done early, you may demo before the due date.
    b. During the demo, along with showing the features you will need to do the presentation (see attached template) to the instructional team member.

Deadlines:

Final Deliverables due on December 1, 2024, by 11:59 pm [No Late Submissions]:

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