Lab W04 – Online Bank

Overview

This project begins the development of an online banking system using a multi-server, multi-client, socket-based architecture.

- The project follows an iterative development model
- You will make design decisions that may change over time
- Requirements may be refined to better support later phases
- Learning and implementation happen at the same time

I am coding this assignment alongside you. I will try to minimize major changes, but some may be unavoidable.

Expectations & Grading

- Each assignment includes core requirements (sometimes general, sometimes specific)
- You have some flexibility in design decisions, but that does not mean anything goes
- Discovering and fixing earlier decisions is part of the learning process
- A significant portion of your grade is based on:
    - Applying concepts from lectures and the textbook
    - Applying them correctly and appropriately
    - Minor decisions that need revision later will unlikely be penalized
- Code that merely "works" is not enough, and it should reflect the concepts we discuss.
- Example: If a value should be an Integer object, and you use a primitive int:
    - It is unlikely that points will be deducted
    - If it causes issues later, you will have to fix it in a future iteration anyway
    - This is how you learn the semantics, subtleties, and nuances of the language.

Code Visibility

After grading, your code will be visible to the combined course.

- Students may learn from each other's ideas and design approaches
- Code may not be copied
- Submit your best work both to learn from others and to avoid being an example of what not to do

Questions & Communication

Have questions? Please ask.

- Email: short, quick questions only, and combine multiple small questions into one message
- Longer or complex questions: office hours or a scheduled meeting
- I am unlikely to respond in the evenings or weekends

Managing your time and planning around limited access to your "boss" is part of the experience.

Tools & Environment

- Eclipse is not required, but recommended for boilerplate support
- Any IDE is acceptable if your project follows the required structure:

  Project folder

src folder
Packages

- See my examples for structure expectations
- Windows or Linux environments are both acceptable. I will avoid platform-specific requirements whenever possible, but I cannot guarantee they will not occur.

Generative AI Policy

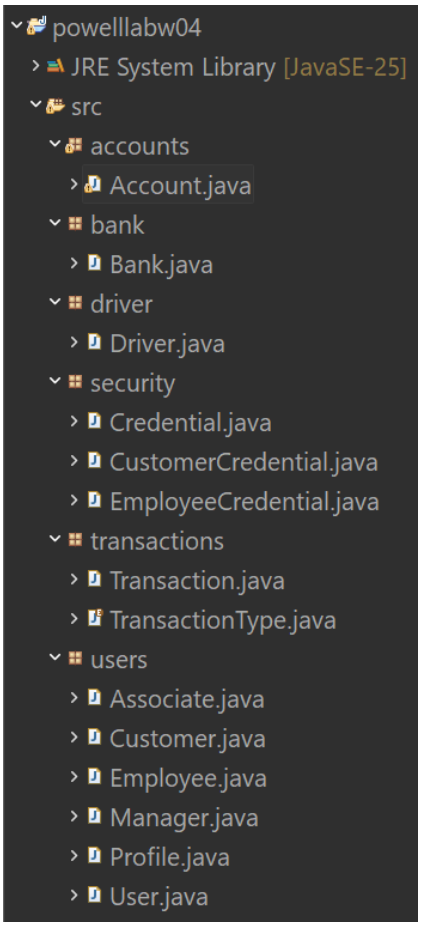
Do NOT use generative AI for these labs.

- Eclipse IDE-generated boilerplate is acceptable
- Generative AI is not acceptable

My experience with AI-generated solutions for this project has been consistently bad:

- It makes incorrect assumptions
- Uses concepts and features we have not covered
- Increases complexity and application size with buried layers of Java library objects
- Feels like the AI programmers are trying to show off complex code
- Reduces performance

Do not be a sloperator. You are fully capable of producing cleaner, more efficient, and more thoughtful code than AI. That is a goal of this assignment and future-proofing your job.



Instructions

The image shows the expected project structure and outlines some basic objects it will need. As we develop it, you will likely need to add methods not described in this assignment.

This is a course on secure programming principles, which means you should be implementing them. The book Java Coding Guidelines (which used to be a required book in this class) is a good resource and includes some really neat, simple examples. I do not expect perfection.

Encapsulate all class members and methods using appropriate access modifiers, i.e., do not make them all public. We want to limit access as much as necessary to prevent direct manipulation (private, protect, and default are the best bets).

Comment your code following the JavaDoc guidelines in Chapter 4 (Object and Classes), which facilitates the generation of HTML documentation for your program. Again, Eclipse is good at helping generate them quickly.

Name the project lastnamelabw04, e.g., powelllabw04.

- Account
    - Fields
        - Account number - integer
        - Balance - double
        - APY (annual percentage yield) - double

        - Overdraft protection (indicates if protected or not) - boolean
        - Owner user ID – integer
        - List of authorized user IDs (other users allowed to use the account) - ArrayList
    - Constructor - Need at least one, but we may add more later
    - Accessors and mutators for each field
        - Must guard against overdraft depending on whether the protection is set
    - Deposit method
        - Must guard against a negative amount parameter
        - Adds the amount to the balance
    - Withdraw method
        - Must guard against a negative amount parameter
        - Must guard against overdraft depending on whether the protection is set
        - Subtracts the amount from the balance
    - Interest method
        - Add monthly interest to the account using the APY (Monthly Interest = (APY / 12) * Principal Amount)
    - For parameter validation (deposit, withdrawal, setBalance, setAPY), throw an IllegalArgumentException if the parameter leads to a violation. Example statements:
        - Balance cannot be negative
        - APY cannot be negative
        - Deposit amount cannot be negative
        - Withdraw amount cannot be negative
    - Methods to add, search, and remove authorized user IDs
    - Overrides
        - String formatting like the Java record class
        - Equals and hash based on the account number
        - Deep cloning
        - Comparing based on the account number
- Credential
    - Fields
        - Username - string
        - Password - string
        - Pin – integer
    - Constructor - Need at least one, but we may add more later
    - Accessors and mutators for each field
    - Overrides
        - String formatting like the Java record class
        - Equals and hash using all the fields
        - Deep cloning
- Customer Credential
    - Does not add any fields to the credential
    - Constructor - Need at least one, but we may add more later
    - Accessors and mutators for all fields
    - Overrides
        - String formatting like the Java record class for all credential information
        - Equals and hash based on all credentials

        - Deep cloning
- Employee Credential
    - Fields: Adds a special authorization code (integer)
    - Constructor - Need at least one, but we may add more later
    - Accessor and mutator for all fields
    - Overrides
        - String formatting like the Java record class for all credential information
        - Equals and hash based on all credentials
        - Deep cloning
- Transaction Type
    - Enum with types of DEPOSIT, WITHDRAW, TRANSFER, INTEREST
- Transaction
    - Fields
        - Account number for the deposit
        - Account number for the withdrawal
        - The amount of the transaction
        - The transaction type
    - Constructor - Need at least one, but we may add more later
    - Accessor and mutator for each field
    - Overrides
        - String formatting like the Java record class
        - Equals and hash based on all the fields
- Profile
    - Fields
        - ID number – integer
        - First name - string
        - Last name - string
        - Email address – string
    - Constructor - Need at least one, but we may add more later
    - Accessors and mutators for each field
    - Overrides
        - String formatting like the Java record class
        - Equals and hash based on the ID
        - Deep cloning
        - Comparing based on the last name and first name
- User
    - Fields
        - Profile
        - Credential
    - Constructor - Need at least one, but we may add more later
    - Accessors and mutators for each field
    - Overrides
        - String formatting like the Java record class
        - Equals and hash based on all profile and credential information
        - Deep cloning
        - Comparing based on last name and first name

- Customer
    - Fields - Does not add any fields to the user
    - Constructor - Need at least one, but we may add more later
    - Accessors and mutators for all fields
    - Overrides
        - String formatting like the Java record class
        - Equals and hash based on all profile and credential information
        - Deep cloning
        - Comparing based on last name and first name
- Employee
    - Fields – Adds an employee title
    - Constructor - Need at least one, but we may add more later
    - Accessors and mutators for all fields
    - Overrides
        - String formatting like the Java record class
        - Equals and hash based on all profile and credential information
        - Deep cloning
        - Comparing based on last name and first name
- Associate
    - Fields - Does not add any fields to the employee
    - Constructor - Need at least one, but we may add more later
    - Accessors and mutators for all fields
    - Overrides
        - String formatting like the Java record class
        - Equals and hash based on all profile and credential information
        - Deep cloning
        - Comparing based on last name and first name
- Manager
    - Fields – Adds a vault access pin to the employee
    - Constructor - Need at least one, but we may add more later
    - Accessors and mutators for all fields
    - Overrides
        - String formatting like the Java record class
        - Equals and hash based on all profile and credential information
        - Deep cloning
        - Comparing based on last name and first name
- Bank
    - Fields
        - ArrayList of Account
        - ArrayList of User
    - Constructor - Need at least one, but we may add more later
    - Accessors and mutators for all fields
    - Methods needed to insert, search, and remove accounts
    - Methods needed to insert, search, and remove users
    - Overrides
        - String formatting like the Java record class

Your driver class should instantiate a bank and add some users and accounts. I do not have the data files ready yet because they need some updates and regeneration. The following are rough examples of what will be in the data files. I will likely have to update things as I generate them for this project, but this will give you an idea.

Customers

```
ID,First,Last,email,username,password,pin
2846341,Alina,Santiago,Santiago@cis3970.edu,factoryshabby,x7zMHkCOG09fX%sR,3311
2203845,Amanda,Emerson,Emerson@cis3970.edu,yelpstatuesque,C&$L2td@fiS39!gG,2257
```

Managers

```
ID,First,Last,email,username,password,pin,passcode,title
1934208,Inigo,Benton,Benton@CISBank.edu,bentonin,dQJHn$8D0YcEu&TO,1246,148962,Loan Manager
1193449,Ivy-Rose,Buchanan,Buchanan@CISBank.edu,buchananiv,dX0bs&fL#hSiIQWo,6068,118395,CEO
```

Associates

```
ID,First,Last,email,username,password,pin,title
1957396,Draga,Loncar,Loncar@CISBank.edu,loncardr,h86$C+T~WZ5r9dx&,4276,Account Manager
1846604,Florianus,Herczog,Herczog@CISBank.edu,herczogfl,8NX7rz_85Ekt.<9c,6089,Account Manager
```

Accounts

```
Number,Manager,Balance,APY,number of owners,owners
7222671,1957396,81000.97,4.15,2,2300011,2394174
9660676,1957396,85028.25,4.25,3,2203845,2683025,2262403
```

Submission

Submit your project to your cis3970work repository.