Module Code & Title: CMP2801M Advanced Programming

Contribution to Final Module Mark: 100%

Description of Assessment Task and Purpose:

Your task is to implement and critically evaluate a simple banking application in C++. The purpose of this assessment is to demonstrate your understanding of programming techniques such as low-level memory management, input/output, logical & mathematical concepts, and multi-paradigm development. The report should document your implementation and evaluation of the application, including the testing performed.

Overview

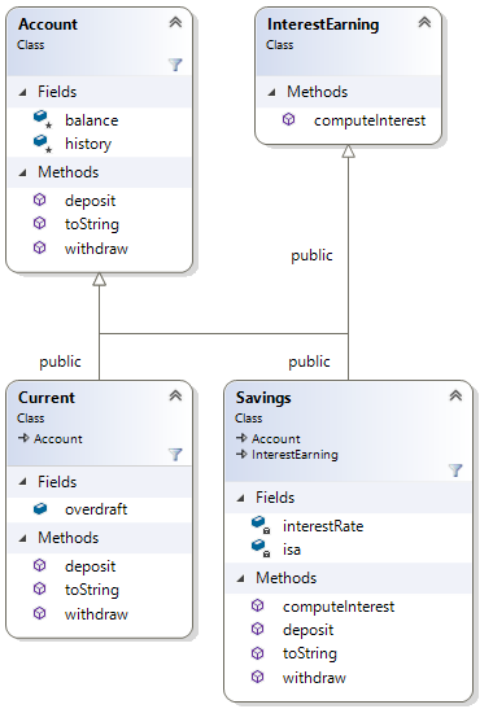
Users should be able to perform the following tasks via a simple command-line interface:

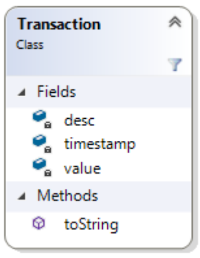
- Open a new account of a specific type
- View an account's balance and recent transactions
- Withdraw funds from an account
- Deposit funds into an account
- Transfer funds between accounts
- Project the interest earned on a savings account for a given period (in years)
- (**Stretch**) Search for transactions by their value
- See a list of all the options above.

A sample execution is provided in a separate document, which you should examine carefully. Your application does <u>not</u> need to maintain a database of any kind; all accounts and transactions can be safely discarded when the program terminates.

Implementation

You are provided with a driver program called **`LincBank.cpp`**, which should serve as a starting point for your implementation. The class diagram of the application is shown below. You may need to implement additional attributes and functions to those shown to produce a working implementation (constructors, destructors, accessors, mutators, etc.)





`Account` should be an **abstract class** that represents a generic bank account. The methods `withdraw` and `deposit` should be pure virtual functions. Current accounts do not accrue interest and can be deposited into/withdrawn from at will. They also include an (interest-free) overdraft of £500. Users can only open one current account, which can be opened with £0 balance. Withdrawals and transfer must not exceed the overdraft limit.

Savings accounts accrue interest but have no overdraft. Regular savings accounts have an interest rate of 0.85%, whereas ISAs have a *whopping* 1.15%. ISAs must be opened with a minimum initial deposit of £1,000. Users can open multiple savings accounts, but only one ISA.

`InterestEarning` is an **interface** that should be implemented by the `Savings` class and consists of one pure virtual function, `computeInterest`. This function should compute a projection of the **compound interest** that will be earned on any savings account for a period provided by the user. The compound interest equation is as follows:

$$A = P(1 + \frac{r}{n})^{nt}$$

where *A* is the final amount, *P* is the initial balance, *r* is the interest rate (as a decimal), *t* is the time the projected period, and *n* is the number of times interest is applied per unit time *t*. In this context, *n* = 12 (i.e. monthly), and the user can project *t* years into the future. The function should return *A*, but it should not be reflected in their actual balance.

Records of all withdrawals and deposits should be stored as `Transaction` objects in an **appropriate data structure** called `history`. Each transaction should store a string describing the nature of the transaction, its value, and a timestamp. For the stretch task, this collection should be searchable.

Marking guidelines
The marker **must be able to compile and run your code** to test its behaviour. You must also ensure that your **class and function names match the class diagram exactly** so that automated testing can be performed. The marker will also examine your program's structure and observe how it demonstrates the following:

- Good practice (user prompts, input validation, defensive programming, comments)
- OO concepts (appropriate use of constructors/destructors, access modifiers, etc.)
- Correct implementation of the inheritance hierarchy/polymorphic types
- Use of virtual/pure virtual functions where appropriate
- Use of advanced techniques such as operator overloading, templates & lambdas

Top marks can be achieved in the following ways:

- Use of STL/contemporary C++ components in your solution
- Innovative use of advanced techniques in your implementation (as above)
- An elegant solution to the stretch task, e.g. a binary search tree
- Thorough testing of a range of possible scenarios

While you may discuss your solution with your peers, please note that this an individual assessment and therefore all code and report content must be your own. Both components of your assessment (code + report) will undergo checking for plagiarism.

Learning Outcomes Assessed:

[LO 1] Apply concepts of advanced software development and programming methods to computational problems;
[LO 2] Use advanced object-oriented principles and programming techniques in software development;
[LO 3] Apply advanced logical and mathematical techniques in the development of software solutions.

Knowledge & Skills Assessed:

- Input/output streams and C-strings
- Low-level memory management/pointers
- Evaluation strategies (call-by-value/call-by-reference)
- Effective use of encapsulation and access modifiers
- Inheritance, virtual functions and abstract classes
- Operator overloading and functional programming

Assessment Submission Instructions:

a) A ZIP file should be uploaded to *Assessment 1 – Supporting Documentation Upload* and should contain your Visual Studio project (without temporary/build files), including your source code. Do not include your report here.
b) A PDF report should be uploaded to *Assessment Item 1 Upload (Turnitin)*
    a. **A contents page (not included in the two-page limit)**
    b. **A basic design for the application:**
        i. A written description of the application
        ii. Components that have been successfully implemented
        iii. Details of your implementation of the stretch task (if relevant)
    c. **A description of your evaluation of the application:**
        i. A description of what tests you did to assure that your program works
        ii. An evaluation of the time complexity of your solution

DO NOT include this briefing document with your submission.

Date for Return of Feedback: TBC

Format for Assessment:

Your submission should consist of the following:

1) A zipped directory (.zip) containing your Visual Studio project must be submitted. Prior to compressing, remove all temporary/build files from the project directory. This will reduce the size of your submitted .zip file significantly. Other compressed formats (tar.gz, rar, etc.) will not be accepted. The zipped directory should be compressed using Windows' *Send to → Compressed (zipped) folder* option, or in UNIX-based operating systems: `zip -r compressed.zip project_folder/`.
2) A report in PDF format detailing your program structure and solution to the problem. A template for this report is given in a separate document. It should be ~2 pages long and no more than 1,500 words. You must use the supplied template and adhere to the headings and structure therein. There will be a penalty for reports longer than 1,500 words.

Feedback Format:

Feedback will be provided via Blackboard, where commentary will be provided on:

- Overall program structure and adherence to C++ conventions
- Missing or incomplete functionality, and areas where it may be improved
- Effective use of object-oriented design principles in accordance with the brief
- Any attempts at the additional tasks and use of STL/contemporary C++ components

- How well comments were used to provide explanations of the program’s logic
- The quality of your technical writing and critical evaluation of your implementation

Additional Information for Completion of Assessment:

Please make sure you have a clear understanding of the grading principles for this component as detailed in the accompanying Criterion Reference Grid. If you are stuck on any component of the assessment, please consult the recommended reading lists, lecture materials (slides, recorded lectures) and workshop tasks in the first instance. Please be advised that the delivery team is not here to debug/fix your code, but to give general advice and further explanation of concepts you may be finding difficult.

Assessment Support Information:

For general enquiries about the assessment strategy, please contact the module coordinator.

For other queries about the module content, please contact a member of the delivery team.

Details of the delivery team’s office hours can be found on the module site on Blackboard.

Important Information on Dishonesty & Plagiarism:

University of Lincoln Regulations define plagiarism as 'the passing off of another person's thoughts, ideas, writings or images as one's own. Examples of plagiarism include the unacknowledged use of another person's material whether in original or summary form. Plagiarism also includes the copying of another student's work'.

Plagiarism is a serious offence and is treated by the University as a form of academic dishonesty. Students are directed to the University Regulations for details of the procedures and penalties involved.

For further information, see www.plagiarism.org