European Union
European
Social Fund

learndirect

Semester 1 - Project

Software Developer L4

Coding a Bank account simulation

Scenario

You have been contracted by GBL Banking Group to write an updated version of their current banking software. The existing software is working fine however GBL are looking to launch a brand new division of the bank and they are hoping to use modern software development practices to create a robust new infrastructure. Whilst the business is hoping that this new software will be more efficient they would like to test out the theory by simulating what the software may look like first.

You will need to develop a prototype program that demonstrates how this could be achieved in a modern programming language. You should consider what facilities a bank account should offer and define these in your program. You will also need to create an interface that users can use to test the functionality of your simulated bank account program.

There are three levels of complexity that you can attempt to create the bank account simulation program. Read through all three levels first so you can get an understanding of what is required at each level before you pick one. It is not necessary to complete the levels in order (i.e. you can just attempt level 3 if you feel comfortable with it) however they do all build on top of each other. You can add any additional functionality at any level which you feel is appropriate or missing in order to create a useful simulation of your bank account program.

You will need to ensure that your code is well commented. You should also produce a test plan with expected and actual results which you can evaluate once your program is complete.

Level 1

Your program should keep a record of a customer's bank balance and all transactions (deposits / withdrawals) they make. There should be the ability to:

- Deposit money to the account
- Withdraw money
- Display the current balance
- Display previous transactions

Some variables that you may wish to keep track of from within your program are:

- Balance
- Customer username / user id
- Transactions

Your program should not allow a customer to withdraw money if they have insufficient funds i.e. withdrawing £20 from an account which only has £10 in it should not be permitted.

Your program should present the user with a menu which provides a list of options relating to the above functions. The program should run continually, asking for further options until the user enters a specific option to exit the program.

When a withdrawal or deposit is made your program should log this transaction and make a list of these available when the 'display previous transactions' option is selected.

You do not need to worry about data persistence.

You should select appropriate data types and declare functions to keep your code maintainable.

Ensure your code is well commented.

Write a test plan for your program to cover all types of data input and then test your program to ensure it behaves as expected. Document any discrepancies and any changes that are necessary.

Level 2

For level 2, you must implement all of the functionality from level 1.

To advance your program, it should be able to handle multiple bank accounts. To achieve this, you will need to persist data in to a file either by serialising/de-serialising it or just writing/reading from a text file. There is some flexibility as to what data should be stored but as a guide, you will need to store the customer's:

- Username / user id
- Pin number (or password)
- Balance

It is not necessary to store the transaction details (withdrawals/deposits) made during each session. Transactions that are logged during each session should log the date/time as well as the amount.

A user should be prompted to login when the program starts which should only accept the correct details as defined in data file. After a user has authenticated they should be presented with a list of options as described in level 1.

You will also need to add an overdraft facility on a bank account. This will allow a user to go below £0 if they try withdraw too many funds. It can be assumed that this is a standard £100 for all accounts.

Ensure your code is well commented.

Write a test plan for your program to cover all types of data input and then test your program to ensure it behaves as expected. Document any discrepancies and any changes that are necessary.

Level 3

In order to make your program more maintainable and allow other programmers to extend it in the future, you should take the description from level 2 and follow the object oriented paradigm. You will need to define classes from which objects can be created which then interact with each other in a 'main' program.

There is flexibility in which classes you choose to define but it is suggested you create a base 'bank account' class which is then subclassed by several types of bank accounts e.g. 'current account **without** overdraft', 'current account **with** overdraft', 'savings account'. You should also create a customer class which provides methods to log them in to the program when it is first run and has member variables which relate to any information which is necessary to hold about the customer e.g. username etc. Define and utilise any other classes which may be useful to your program.

Your program should also aim to keep a record of transactions that are made by a particular bank account. In other words, if **user A** logs in and deposits £50 and withdraws £20 those two transactions should be stored in a file before the program is exited. When the program is run again, the user should be able to still see those transactions listed when they choose to display previous transactions from the menu.

The main aim at level 3 is that the classes you define leave your bank account software extensible; it can be easily improved and extended by other programmers at a later point.

Ensure your code is well commented.

Write a test plan for your program to cover all types of data input and then test your program to ensure it behaves as expected. Document any discrepancies and any changes that are necessary.