Objective

Practice selection and repetition.

Introduction

This project is an extension of Project 4; you will make use of the class `DividendAccount` (as you modified it last week) but to solve a new problem.

You are to write a C++ program that allows the user to create exactly one `DividendAccount` object and then to specify as many withdrawals and deposits as desired. When the sequence of deposits and withdrawals is complete the dividend rate should be applied to the account and the final account status printed. Use the following dialogue as a guide to this problem's specification:

```
Enter account name: home
Enter balance: 500.0
Enter dividend rate (between 0 and 100.00): -3
Enter dividend rate (between 0 and 100.00): 3

W)ithdraw, D)eposit, or Q)uit: D
Enter deposit ammount: 250.0
W)ithdraw, D)eposit, or Q)uit: D
Enter deposit ammount: -100.0
Enter deposit ammount: -10.0
Enter deposit ammount: 100.0
W)ithdraw, D)eposit, or Q)uit: W
Enter withdrawal amount (between 0 and 850.00): 1000
Enter withdrawal amount (between 0 and 850.00): -100
Enter withdrawal amount (between 0 and 850.00): 100
W)ithdraw, D)eposit, or Q)uit: A
Enter W, D, or Q: B
Enter W, D, or Q: Q

Account Name:          home
Account Balance:       $751.88
Number of withdrawals: 1
Number of deposits:    3
```

All input data must be validated — there are three different situations for this validation.

- The dividend rate and withdrawal amount must fall between zero and an upper bound. For the dividend rate the value must a real value between 0 and 1; the withdrawal amount must be between 0 and the current balance.
- The initial balance and deposit amount must be validated to be positive.
- Menu data must be validated to be one of the upper-case letters `W`, `D`, `Q`.

Details

This project is significantly larger than previous projects in the course. To make it more manageable, here are some suggestions.

1. Implement the problem assuming all data entered is correct. This means you can write input functions which just prompt, input a value, and then return the value. You can make use of the function `getDouble` from last week.
2. Rewrite the function from Project 4 called `getAccountInfo` so that it doesn't take a prompt parameter and so that it reads the initial balance and dividend rate by calling `getDouble`.
3. Implement a function `getAnswer` with signature.

   ```
   char getAnswer(const string & prompt)
   ```

   which prompts (as indicated in the sample output) and inputs a character value — return that character value. Remember at this point that you should assume the input value is valid.
4. Implement the `main` function so that it initializes the account and then allows the user to make a sequence of withdrawals and deposits.

   Complete the program so that it works as described so far.
5. Modify `getDouble` so that it has the following interface.

   ```
   double getDouble(const string & prompt,
                    double lower)
   // Post: val is the first input value such that
   //           val >= lower
   //       return val
   ```

The new function returns when a valid value is entered – that means a value which is greater than or equal to the parameter `lower`. Add this function to your implementation and test it. In this program the lower value is always zero.

6. Implement a new input function `getBoundedDouble` based on `getDouble` which has the following signature.

```
double getBoundedDouble(const string & prompt,
                        double lower,
                        double upper)
// Pre:  lower < upper
// Post: val is the first input value such that
//           lower <= val <= upper
//       return val
```

   The new function returns a data value only if it falls within the specified range.

   Add this function to your program, modify the appropriate calls (i.e., for inputting dividend rate and withdrawal amount), and test the program.

7. Finally, modify `getAnswer` so that it will return only when a valid character is entered. Here's the appropriate new interface.

```
char getAnswer(const string & prompt)
// Post: val is the first input value such that
//           val == 'D' OR val == 'W' OR val == 'Q'
//       return val
```

8. This assignment must adhere to the CSCI 203 Style Guidelines.

It is important to know that you can use the same make file as you did last week as well as the same `DividendAccount.cc` and `DividendAccount.h` files. The file containing `main` and the other functions mentioned above should have a similar structure to the corresponding file from Project 4.

Relax — apply the techniques discussed class and if you have any questions or difficulties talk to your instructor.

What to Hand In

Hand in the code for the final version of your program along with an execution which illustrate that the program works correctly. You should demonstrate that the program correctly handles invalid input data, as required in the project.