Part One: Implementing the Bank Account Example

In a recent video on how to design solutions to problems, one of the examples was about bank accounts. In this exercise, you will walk through the process of creating an application for the bank example. The key point here is that we never think about how to code a bank. Instead, we thing about how to divide a bank into different concerns. Then we can think about how to code one concern. In that way build a solution out of simpler parts. What could have been a very complicated problem becomes a sequence of much simpler problems.

When a problem looks complicated, do not think about code. Think about concerns, and divide the problem into separate concerns. We can divide the problem by creating a separate class for each concern. That is called the "architecture". Think of it as choosing the rooms of the house. We also divide concerns by sequencing the order in which to address the concerns. Implement the basic concerns first and get them to work. Then add concerns in order, where each new concern represents only a small addition. This approach is like adding the furniture and decorations. The difference between the two approaches is that in the first approach, we may write a significant amount of code, but all in one class. In the second approach, we may have to add code to several classes, but only a small amount in each class.

Let's get started.

Step 1:

Create a project, using your name, like FirstBankOfMike. Create a cpp file of that name. Add your header information (file, purpose, author, history, copyright) and the main function.

```
int main() {
    return 0;
}
```

Even though it does nothing, make sure it compiles before going forward.

Now let's start constructing a bank.

From the description of the bank, we observe that major concepts were accounts, customers, and transactions. Which one is the easiest for starters? Let's start with a customer, since we can do that without knowing anything about a bank.

A customer has an identity and some connection to the world, like a business or an address. Keep it simple. We can always change it later. We will give the customer a name and an address.

Step 2:

Implement a Customer class. Use Project -> add Class. The private part of the customer has two strings, name and address. The variables of a class are called "properties". We can't have a customer with no identity (how could we distinguish different customers?) so we will require a name when the Customer is created – in the constructor. Then we just need two more methods, setAddress(), which is a setter for the address property, and display() which displays the customer properties in a console, using cout.

Remember, in order to use the string type, the Customer.h file needs to have #include <string> and using namespace std;. Since it is in the Customer.h file, you don't need to repeat these two lines in the .cpp file. In the .cpp file, remember that you have to tell the compiler that the methods belong to the Customer class by using the Customer:: prefix for each of the methods you define.

Visual Studio starts you with a constructor and a destructor. We are not going to do anything with the destructor, yet. So leave it as it is. But we will change the constructor from Customer(void) to Customer(string name). Here's a useful trick. Please try this. Use the word for the property and for the input parameter of the constructor – in this case "name". Then, in order to assign the value of the input parameter called "name" to the property that is also called "name", we use a C++ feature, called "this", to specify the one that belongs to the class.

```
this->name = name;
```

Be careful that you do not put any spaces around the `->`.

You can use the same trick for the implementation of the setter, setAddress(string address).

To test your class, use the following code in main (please invent your own names and addresses):

```
#include "Customer.h"

int main() {
    Customer person1("Mike Van Hilst");
    Customer person2("Sponge Bob");
    person1.setAddress("1600 Pennsylvania Avenue");
    person2.setAddress("124 Conch Street");
    person1.display();
    person2.display();
    return 0;
}
```

The output should be:
Mike Van Hilst
1600 Pennsylvania Avenue
Sponge Bob
124 Conch Street

Now let's create the Account. Keep it simple. We can add more later.

What does every account have? It has a balance. It also needs an account "holder", so we know who's account it is and who can use it.

What can we do with an account? Deposit and withdraw. It might also help if we can make the account display itself, and display the holder. For starters, there isn't much to show – just the balance.

Step 3:

Implement another class called Account. Again, use Project -> Add Class. The Account has two properties, balance and holder. Since we don't have a currency type, we make it the balance a double. Make the holder a Customer. Remember, that to use the Customer type, the Account.h file must have #include "Customer.h".

This time, the constructor does not take any input, so leave it Account(void). However, in the implementation of the Account constructor, add a line to set the balance to 0.0, so that the account starts with a 0 balance.

You will need methods for deposit(double amount) and withdraw(double amount). Make these two methods have a return type of bool. In their implementation, test the amount to make sure it is greater than 0. In the withdraw method, also make sure that the balance is at least as big as the amount. If everything is OK, add or subtract the amount from the balance, and return true. If any of the conditions fails, do not change the balance, but just return false.

We also want a getter for the balance, called getBalance() that just returns the balance. It's return type should, of course, be double.

We also need a setter for the holder called setHolder. Here's a problem for which we must use something new. A customer can be the holder of more than one account. So we don't want to have separate customer information in their different accounts. We want one instance of the customer, and then just have each of their accounts point to that one instance. To do that, we will use a Customer reference rather than a Customer value. We do that with a *. In the .h file, the holder property is `Customer* holder` (instead of `Customer holder`). Similarly, the setter is `setHolder(Customer* holder)`. Since the holders are bother references (of the same type), the assignment is still:

```
this->holder = holder;
```

You will also want methods to display the holder information, called displayHolder(). There is no need for us to figure out how to display the holder's information. The Customer class already does that. Here the implementation can just call the holder's display() function. Note, however, that the holder is a Customer* reference. We can't use `holder.display()` the way we would if we have an instance of a Customer. Instead, we have to use the following form:

```
holder->display();
```

Now that we have talked about using `->` for object references, we can see what "this" is. Within the implementation of a class in C++, the key word "this" is the implicit reference pointing to the current instance of the class. In other words, it's a reference to "this" object, the same way "holder" was a reference to a Customer object.

Add the following code to your main (below the code that is already there and before return 0;)

```
Account account1;
account1.setHolder(&person2);
cout << account1.deposit(100) << endl;
cout << account1.withdraw(1000) << endl;
cout << account1.withdraw(50) << endl;
cout << account1.getBalance() << endl;
account1.displayHolder();
```

Note two things. First, when we create account1, since the constructor for Account does not take an input parameter, we do not use parens the way we did when we created person1. Second, since setHolder needs a reference to person2 and not a copy of person2, we have to use the reference operator "&" which is prepended (no space) to the variable name. In C++ we read the "&" as "pointer to".

If you implemented the Account class correctly, the output should appear as below, except that your names and addresses will be different:

```
Mike Van Hilst
1600 Pennsylvania Avenue
Sponge Bob
124 Conch Street
1
0
1
50
Sponge Bob
124 Conch Street
```

That is the end of Part 1. When you have it working correctly, zip up the FirstBankOf<me>.cpp, Customer.h, Customer.cpp, Account.h, and Account.cpp files and submit them for this assignment.