# Getting Started with Objects

In this tutorial, we'll create our first C++ class/object by refactoring some code that uses structures. We'll step through parts of this together as an additional reminder about using Visual Studio and C++.

## **1)** Simple Bank Account Example

* Create yourself a new C++ project using either the Empty Project Template
* Add a new cpp file to the project
* Copy across the basic bank account example code from the end of this document

2) Look at the code, run it and make sure you understand what is going on - ask questions about stuff you are not sure about. The application simply sets up a basic bank account scenario where you can view your account, make deposits and withdrawals.

3) Add two new files to the project: a .cpp file (source) and a .h file (header). (You can also use the Add Class option within Visual Studio to help with this process)

* You will be refactoring the BANK\_ACCOUNT structure from the example code so think about what would be a sensible name for these files (they can, and should, have the same filename, barring the extension)

4) Refactor the BANK\_ACCOUNT structure into an object (/class), adding appropriate methods and revising the main code.

Tips

* When writing getter and setters, only provide them if you think you need them. Don’t just write getters and setters for the sake of it; never write code just for the sake of it (unless the sake of it is out of interest, but then I guess there is a reason after all). Think about what is useful now and write that code. Unless you are working from a detailed design, an effective way of writing code is to let the code evolve over time.
* You will need to write a constructor that takes parameters and update the object’s constructor to (notice that it doesn’t take a balance parameter):

BankAccount account(101, "Mike Meredith");

* The viewSummary method receives the account object as a constant. This means it cannot be changed. Any methods called within this function must have the const post-fix keyword, e.g.

const string getCustomerName() const;

* What on earth… this line of code has two const keywords – what manner of voodoo magic is this? The first const is connected to the return (i.e. the returned string value cannot be changed) whereas the trailing const means this method doesn’t alter the object state. We will discuss this in more detail later.

5a) Having the main body of code check to see if the account object can process a withdrawal is an utterly stupid thing to do (completely stupid… it is the same as walking around with your purse or wallet open for anyone who passes to take whatever they want).

Object Oriented Programming is about encapsulating and assigning responsibility correctly. A much better design is to allow the bank account object to decide whether it can make the withdrawal request or not. A positive refinement is therefore to update the code (perhaps writing a new method) that means when a withdrawal is attempted the account is responsible for checking (and validating) and updating its balance. It is also convenient to know whether the withdrawal was successfully so perhaps use a Boolean return for that method.

5b) It is arguable that a deposit method should act in a comparable way to the withdraw method in that you request the account object to make a deposit and if it can, it does so, updating the balance, and then returns a Boolean indicator of success. Add this change into the system… now we are writing code for the sake of it, but we will come back to this.

5c) If you haven’t already, change the name of the account … I mean geez, how many bank accounts do I (Mike Meredith) need?

6) Write constructor and destructor code that allows the balance to persist between executions of the program. i.e. when the program loads, code within the constructor opens a file and reads in a balance and when the program terminates, the destructor opens the same file and writes the balance. You may need to go back and look at your FoP notes for file IO stuff, but the key thing to remember is that the constructor is automatically called when the object is created and the destructor is automatically called when the object is destroyed so you don't need to call these "methods" implicitly.

**Extras**

7) Modify the code so that you can enter the account name (or number) and it uses this information to create the account object. You should also update your load and save mechanism to store balances in different files depending on the account name (or number).

8) Can you figure out at what point in the code the account object constructor and destructor are being called?

9) Experiment. Do stuff with the code and see what happens. One of the greatest assets you can have is inquisition. How could this be done? What happens if …? Cause and effect… you’ll hear me say that a lot this year.

10) If you have got this far, I would like to admit that this example is a little dry, but things will get better as we progress… what, you were expecting to write a tower-defence style game using object oriented principles in your first week back? No of course not. That we start next week now we can all programming in an OO sense in C++.

**Basic Bank Account Example Code**

#include <iostream>

#include <string>

using namespace std;

#define MENU\_EXIT 0

#define MENU\_VIEW\_SUMMARY 1

#define MENU\_DEPOSIT 2

#define MENU\_WITHDRAW 3

#define FIRST\_MENU\_ITEM MENU\_EXIT

#define LAST\_MENU\_ITEM MENU\_WITHDRAW

struct BANK\_ACCOUNT

{

unsigned long accountNumber;

string customerName;

int balance; // in pence

};

void displayMenu();

int getMenuChoice();

void viewSummary(const BANK\_ACCOUNT& account);

void deposit(BANK\_ACCOUNT& account);

void withdraw(BANK\_ACCOUNT& account);

int main()

{

BANK\_ACCOUNT account={101, "Mike Meredith", 0};

int choice;

do

{

displayMenu();

choice=getMenuChoice();

switch (choice)

{

case MENU\_VIEW\_SUMMARY: viewSummary(account); break;

case MENU\_DEPOSIT: deposit(account); break;

case MENU\_WITHDRAW: withdraw(account); break;

}

} while (choice!=MENU\_EXIT);

return 0;

}

void displayMenu()

{

cout << "\nBank Account Manager\n";

cout << "====================\n";

cout << MENU\_VIEW\_SUMMARY << ") View Account Summary\n";

cout << MENU\_DEPOSIT << ") Make deposit\n";

cout << MENU\_WITHDRAW << ") Make withdraw\n";

cout << MENU\_EXIT << ") Exit\n";

}

int getMenuChoice()

{

int choice;

cout << "Choose option: ";

cin >> choice;

while (choice<FIRST\_MENU\_ITEM || choice>LAST\_MENU\_ITEM)

{

cout << "Invalid option. Choose option: ";

cin >> choice;

}

return choice;

}

void viewSummary(const BANK\_ACCOUNT& account)

{

const int pounds=account.balance/100;

const int pence=account.balance-(pounds\*100);

cout << "\nAccount Summary\n";

cout << "---------------\n";

cout << account.customerName << "\n";

cout << "----\n";

cout << "Balance: \x9C" << pounds << ".";

if (pence<10)

cout << "0";

cout << pence << "\n\n";

}

void deposit(BANK\_ACCOUNT& account)

{

double amount;

cout << "\nMake a deposit\n";

cout << "--------------\n";

cout << "Enter amount to deposit \x9C";

cin >> amount;

account.balance+=(int) (amount\*100);

}

void withdraw(BANK\_ACCOUNT& account)

{

double amount;

cout << "\nWithdraw funds\n";

cout << "--------------\n";

cout << "Enter amount to withdraw \x9C";

cin >> amount;

int tmp=(int) (amount\*100);

if (tmp<=account.balance)

account.balance-=tmp;

else

cout << "\n\*\* Not enough funds to withdraw requested amount \*\*\n";

}