Ch 3- Classes & Objects

A Simple First Example

Here we shall look at a simple example of the Class and how it is used. The example is trivial but is used to explain some important concepts of OOP

C++ Syntax

The 'class' keyword	defining a class
The 'private:' keyword	defines the 'hidden' part of the class. The data attributes.
The 'public:' keyword	defines the visible interface of the class

The class

In OOP we model an abstract data type by using the ‘class'. The class has two parts;

'private' - Allow access via class subroutines & functions only.
'public' - The variable can be used by any code file in the project and defines the 'behaviour' (methods) of any object of the class.

It is normal practice to put attributes into the private part of the class, where they can only be accessed via methods. Methods appear in the public part of the class so they can be accessed externally and provide the interface to the class.

In the following examples, attributes are private and methods are public.
The syntax of a class (here called 'ExampleClass') in C++ is as follows:

```
Public Class ExampleClass
   ' Attributes
   Private i As Integer

   'Methods
   Public Sub ExampleClass()
      i = 0
   End Sub

   Public Sub setValue(ByVal value_in)
      i = value_in
   End Sub

   Public Function getValue() As Integer
      Return i
   End Function

End Class
```

Class declaration.
The class is called ExampleClass

Private attributes.
Accessed only by class functions(below)

This is called the **constructor**.
When an object of the class is created the constructor is used to set initial values.

a public method called setValue
which sets the value of the attribute ‘i' using an integer parameter

a public method called getValue
which returns the attribute ‘i'

- The integer variable 'i' is private and therefore **inaccessible** directly from outside the class. Only the 'setValue' and 'getValue' methods may change its value.
- Methods are known as 'member functions'. The member functions 'setValue' and 'getValue' are defined 'inline' (inside the class definition) in this case.

A More Useful Example

The 'ExampleClass' shows the essential features of the syntax, but does not provide a useful abstract data type. All we could do with it would be to set and return the value of an integer. As a more realistic example, we will create a class called 'BankAccount' which' will have the following attributes:

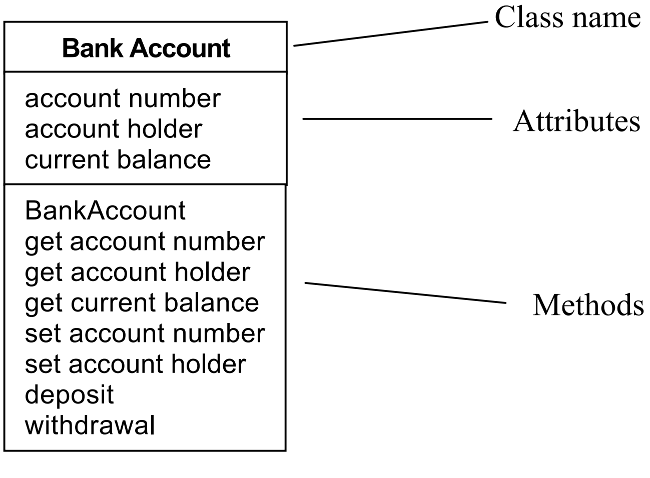
account number
account holder
current balance

It will have the following methods:

Constructor
BankAccount
Selector methods:
get account number
get account holder
get balance

Modifier methods:
set account number
set account holder
deposit
withdrawal

A simple diagram is used to show the class, followed by its attributes and its methods, using a divided rectangle:



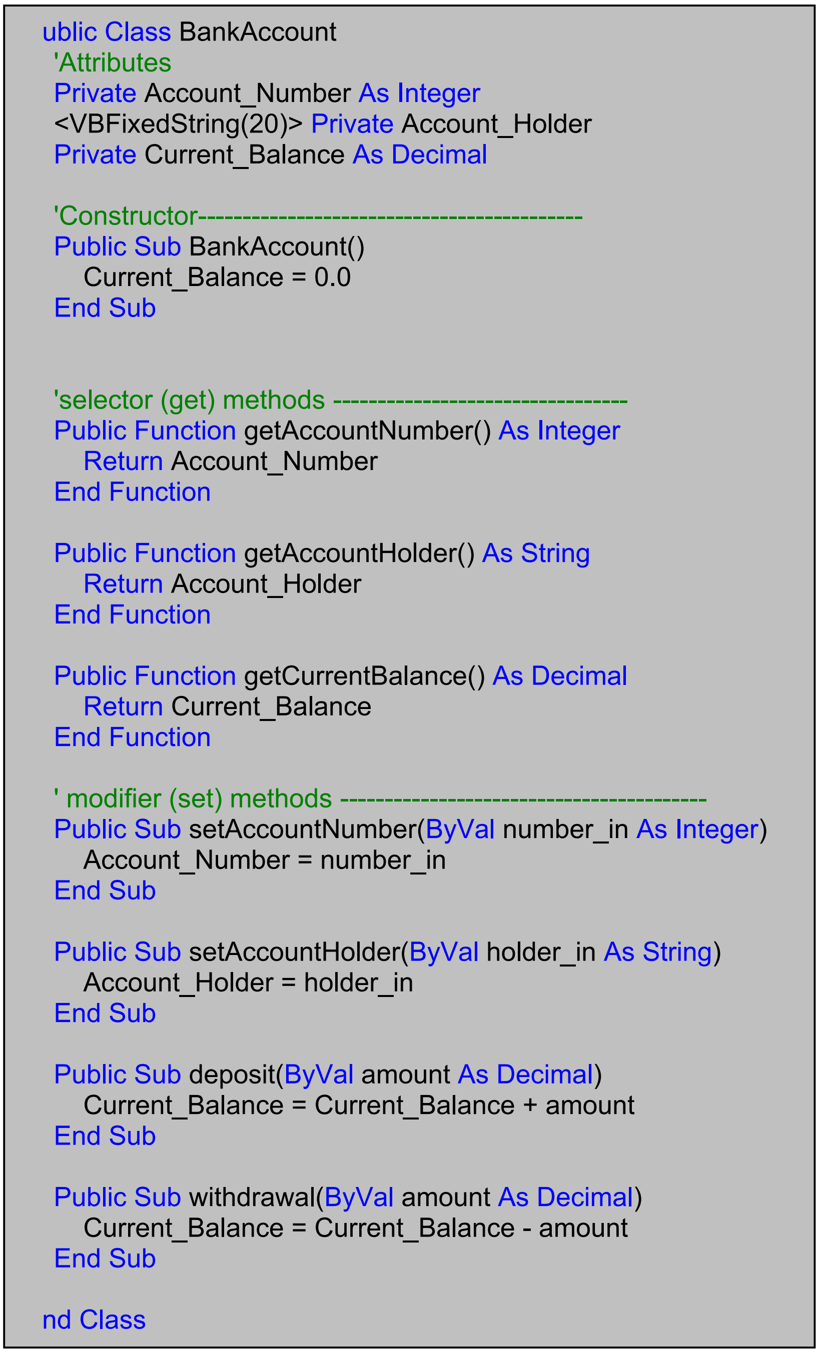
The class might be implemented in C++ as shown below(over page).

Note the following:

The **attributes** must have types (integer, String and decimal in this case)
The **methods** must also have return types.
Selector methods usually have a specific return type, whereas **modifier** methods are often Subroutines.

It is good practice (though it makes no actual difference to the compiler) to group selector operations, followed by the modifier operations, as shown here.

Example: BankAccount



```
ublic Class BankAccount
 'Attributes
 Private Account_Number As Integer
 <VBFixedString(20)> Private Account_Holder
 Private Current_Balance As Decimal

 'Constructor-----------------------------------------
 Public Sub BankAccount()
    Current_Balance = 0.0
 End Sub


 'selector (get) methods --------------------------------
 Public Function getAccountNumber() As Integer
    Return Account_Number
 End Function

 Public Function getAccountHolder() As String
    Return Account_Holder
 End Function

 Public Function getCurrentBalance() As Decimal
    Return Current_Balance
 End Function

 ' modifier (set) methods ----------------------------------------
 Public Sub setAccountNumber(ByVal number_in As Integer)
    Account_Number = number_in
 End Sub

 Public Sub setAccountHolder(ByVal holder_in As String)
    Account_Holder = holder_in
 End Sub

 Public Sub deposit(ByVal amount As Decimal)
    Current_Balance = Current_Balance + amount
 End Sub

 Public Sub withdrawal(ByVal amount As Decimal)
    Current_Balance = Current_Balance - amount
 End Sub

nd Class
```

the **private** attributes come first

followed by the **public** methods

Object Constructor

BankAccount set CurrentBalance when the object is first created.

selector (get) methods ---------------

getAccountNumber returns the integer account number

getAccountHolder returns the name of the account holder

getCurrentBalance returns the current account balance (a decimal)

modifier (set) methods ----------

setAccountNumber sets the account number from the integer parameter

setAccountHolder sets the name of the holder from the string parameter

deposit allows a parameter value to be added to the current balance

withdrawal allows a parameter value to be subtracted from the balance

Exercise

1) Create a new project. Add the above file as a new class called **BankAccountClass**.

2) In the main module add the code below and save it all as **BankAccount**

```
Imports System
Imports BankAccount
Module Module1

   Sub Main()
      Dim account1 As BankAccount
      account1 = New BankAccount
      Dim account2 As BankAccount
      account2 = New BankAccount
      Dim account3 As BankAccount
      account3 = New BankAccount

      account1.setAccountNumber(100.0)
      account2.setAccountNumber(110.0)
      account3.setAccountNumber(120.0)

      Console.WriteLine("Account Numbers Are :")
      Console.WriteLine("{0}", account1.getAccountNumber( ) )
      Console.WriteLine("{0}", account2.getAccountNumber( ) )
      Console.WriteLine("{0}", account3.getAccountNumber( ) )

      Console.ReadKey()

   End Sub
End Module
```

Import the system functions
Import the BankAccount class

Instantiate 3 accounts

Set the 3 account numbers

Use getAccountNumber() to display the three Account numbers

Expected output

```
Account Numbers are :
100
110
120
```

3) Add further code to achieve the following:

a) set the name of the account holders as specified below.
Use setAccountHolder()

b) set the balance of the accounts as specified below.
Use deposit()

	Account1	**Account2**	**Account3**
Name	Yourself	Harry Hill	Bob Zurunkle
Deposit	1000	500	Remain 0

4) Display each account name & current balance (check that it matches the table above)
use getAccountHolder() & getCurrentBalance()

5) Make a withdrawal of 500.50 from your account
use withdrawal()

6) Display details of all three accounts. Check that it matches the table below.

	Number	Name	Balance
Account1	100	Yourself	499.50
Account2	110	Harry Hill	500
Account3	120	Bob Zurunkle	0

Summary

With the BankAccount class we have modelled a simple bank account as an abstract data type; providing the attributes and methods necessary to hold and update data for any bank account created using this data type.

In order to use this effectively, we will need to store it in a file (i.e. BankAccountClass) which may be included in subsequent programs which process bank accounts.

Whenever we create new abstract data types and store them in such files, we are adding to a potential library, allowing us to create objects of any predefined class.

Key concepts from this chapter

1. **Encapsulation** is the combination of state (attributes) and behaviour (methods) into a software 'object'.

2, **Information hiding** is the division of an object into private and public parts. The public part provides the external interface. The private part (and all internal processes) are 'hidden' inside the object.

3. An **abstract data type** defines the **attributes** (state) and **methods** (behaviour) of all objects belonging to a particular 'class’.

4. We may make a distinction between '**selector**' methods (which access attributes but cannot change them) and '**modifier**' methods (which are able to change the state of attributes).

5. In OOP, we use the 'class' to model abstract data types. The keywords `**public**' and '**private**' are used to define the class interface and hidden part respectively.

Exercises

1. Consider the attributes and methods required for a coffee cup. Using the notation previously shown (the divided rectangle), list the name, attributes and methods which would apply to a ‘coffee cup' abstract data type.

2. Suggest attributes and methods for a “wallet/purse" abstract data type. Think about what you need a wallet or purse for (what its behaviour is) and what internal attributes relate to these behaviours. Bear in mind that there is no single 'right answer' for this kind of abstract exercise.

3. Create a OOP class to represent a person with attributes of name, year of birth and height in metres. Define methods to get and set these three attributes. Add a method which will return a person's (approximate) age when given a year as a parameter. Add another method which will return their height in centimetres. These values may be derived from the given attributes. Do not add extra attributes to the class.

4. Create a OOP class for a 'StockItem' abstract data type. It should have the attributes of stock level (an integer) and unit price (a float). Define methods to return the values of these two attributes and to set them using parameters. Add two more methods to allow stock receipts, and stock issues, updating the stock level as appropriate.