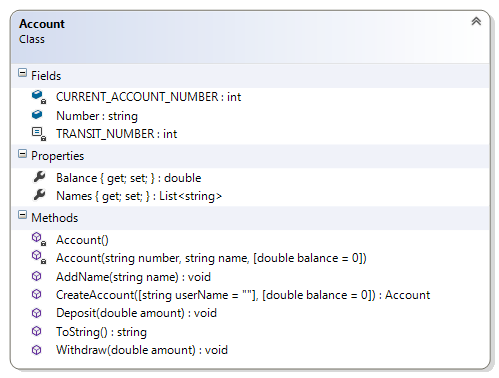
This exercise you will be exploring more usage of the static keyword.

# You must following the specifications exactly

# The Account class

This is the front end of the application. There are 12 members.

#### Description of class members

##### Fields:

**CURRENT\_ACCOUNT\_NUMBER** – this private class variable is an int representing the unique number to be used when creating a new account. This is incremented in the instance constructor

**Number** – this public instance variable is a string indicating the current account number of this object reference. This must be decorated with the readonly keyword. This is set in the **CreateAccount()** method.

A **readonly** field is like a **consts** but it is normally assigned when the program is running

A **const** is a value that cannot be modified. Its value is set at declaration. i.e. When the program is compiled

**TRANSIT\_NUMBER** – this private int is a constant representing the branch number of all the accounts. That is initialized to 314. It is used in the **CreateAccount()** method to build the account number of this account.

##### Properties:

**Balance** – this property is a double that represents the current balance of this object. . This getter is public and the setter is private.

**Names** – this property is a list of string representing the name associated with this object. This getter is public and the setter is private.

##### Constructor:

There are two constructors for this class: a static and a private one

**Account()** – This is the static constructor. It will be used to initialize the class variable **CURRENT\_ACCOUNT\_NUMBER** to a suitable value (10000?)

A static constructor is used to initialise the static fields and properties. It is invoked once in the life time of a program, before ANY member is accessed. There is no access modifier i.e. public, protected or private

**Account(string number, string name, double balance)** – This is a private constructor that does the following:

* The Number field and the Balance property are initialized to the values of the arguments
* The Names property is initialised to an empty list (use the new operator). And the argument is added to this collection

##### Methods

**AddName(string name)** – This is a public method adds the argument int0 the Names collection. It is possible to have multiple names associated with this account. This method does not return a value.

**Deposit(double amount)** – This is a public method that increases the property Balance by the amount specified by the argument. This method does not return a value.

**Withdraw(double amount)** – This is a public method that decreases the property Balance by the amount specified by the argument. This method does not return a value.

**ToString()** – This is a public method overrides the corresponding method in the object class to return a stringify form of the object. For this exercise omit the Names field because it is a list of strings

**CreateAccount(string name, double balance = 0)** – This is a public static method is used to create accounts. It does the following:

* Builds an account number from the **TRANSIT\_NUMBER** and **CURRENT\_ACCOUNT\_NUMBER** according to the template “AC-[transit number]-[current account number]. e.g. if the value of **TRANSIT\_NUMBER** and **CURRENT\_ACCOUNT\_NUMBER** are 314 and 10005 respectively then the account number will be “AC-314-10005”  
  It also increments the **CURRENT\_ACCOUNT\_NUMBER** field so the next object will have a unique number.
* Instantiate a new account object with the appropriate arguments. And returns this object

### Test Harness

Insert the following code statements in your Program.cs file:

List<Account> accounts = new List<Account>();

Random rand = new Random();

for (int i = 0; i < 6; i++)

{

accounts.Add(Account.CreateAccount("#" + i, rand.Next(10, 100)));

}

Console.WriteLine("\n\nAll accounts");

foreach (Account account in accounts)

{

Console.WriteLine(account);

}

foreach (Account account in accounts)

{

account.Deposit(1.11);

}

Console.WriteLine("\n\nAfter $1.11 deposit ");

foreach (Account account in accounts)

{

Console.WriteLine(account);

}

foreach (Account account in accounts)

{

account.Withdraw(0.05);

}

Console.WriteLine("\n\nAfter $0.05 withdrawal.");

foreach (Account account in accounts)

{

Console.WriteLine(account);

}