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 initialized in the static constructor. 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

**static Account()** – This is the static constructor. It will be used to initialize the class variable **CURRENT\_ACCOUNT\_NUMBER** to a suitable value e.g. **100000**

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

**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

**public void 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.

**public void 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.

**public void 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.

The static string method Join enables you to stringify a collection. Use the following code to get string from a collection:
**string.Join(", ", Names);**

**public override string ToString()** – This is a public method overrides the corresponding method in the object class to return a stringify form of the object. To display the Names field, you will have to do some extra work because it is a list of strings []

**public static Account 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”. [Hint: use **string.Format()** method to do this.]
* 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.
	+ Returns the above object

### Test Harness

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

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

Random rand = new Random();

accounts.Add(Account.CreateAccount("Narendra", rand.Next(50, 500)));

accounts.Add(Account.CreateAccount("Ilia", rand.Next(50, 500)));

accounts.Add(Account.CreateAccount("Yin", rand.Next(50, 500)));

accounts.Add(Account.CreateAccount("Arben", rand.Next(50, 500)));

accounts.Add(Account.CreateAccount("Patrick", rand.Next(50, 500)));

accounts.Add(Account.CreateAccount("Joanne", rand.Next(50, 500)));

accounts.Add(Account.CreateAccount("Nicoleta", rand.Next(50, 500)));

accounts.Add(Account.CreateAccount("Mohammed", rand.Next(50, 500)));

Console.WriteLine("\nAll accounts");

foreach (Account account in accounts)

{

 Console.WriteLine(account);

}

foreach (Account account in accounts)

{

 account.Deposit(55.55);

}

Console.WriteLine("\nAfter $55.55 deposit ");

foreach (Account account in accounts)

{

 Console.WriteLine(account);

}

foreach (Account account in accounts)

{

 account.Withdraw(1.11);

 if(account.Balance > 300)

 account.AddName("(Hacked)");

}

Console.WriteLine("\nAfter $1.11 withdrawal");

foreach (Account account in accounts)

{

 Console.WriteLine(account);

}

### Test Harness

Output

All accounts

[AC-314-10000] Narendra $390.00

[AC-314-10001] Ilia $228.00

[AC-314-10002] Yin $210.00

[AC-314-10003] Arben $351.00

[AC-314-10004] Patrick $224.00

[AC-314-10005] Joanne $210.00

[AC-314-10006] Nicoleta $126.00

[AC-314-10007] Mohammed $194.00

After $55.55 deposit

[AC-314-10000] Narendra $445.55

[AC-314-10001] Ilia $283.55

[AC-314-10002] Yin $265.55

[AC-314-10003] Arben $406.55

[AC-314-10004] Patrick $279.55

[AC-314-10005] Joanne $265.55

[AC-314-10006] Nicoleta $181.55

[AC-314-10007] Mohammed $249.55

After $1.11 withdrawal

[AC-314-10000] Narendra, (Hacked) $444.44

[AC-314-10001] Ilia $282.44

[AC-314-10002] Yin $264.44

[AC-314-10003] Arben, (Hacked) $405.44

[AC-314-10004] Patrick $278.44

[AC-314-10005] Joanne $264.44

[AC-314-10006] Nicoleta $180.44

[AC-314-10007] Mohammed $248.44

Press any key to continue . . .