The ABCs of Classes and Objects

A Tutorial and Walkthrough of coding Classes and Objects

The purpose of this tutorial is to give the student an opportunity to write code and to explore the fundamentals of Classes and Objects. Specifically, this tutorial demonstrates fields, properties, constructors, and methods in class declarations along with object instantiation and the use of the member access operator.

*Understanding Classes and Objects is easier if you’ve already taken a look at the distinction between* ***variables****,* ***values*** *and* ***data types****.*

# About Classes and Objects

A computer program is a ***set of instructions*** for manipulating ***information***.

***Information*** is central to the whole purpose of computer programs, and is the “thing” that has value in the eyes of the end-user. Information is often complex, being made up of many smaller pieces of related information. For example, the information could be the details of a bank account, the marks earned by a student, or information about a company or an employee. A key aspect of all of these is that the *details* of the information (First and Last name, for example) are *bundled together* and treated as a single unit – a composition of related information. In programming, those bundled sets of detailed information are known as objects.

***Objects*** embody more than just information, however. A person can *do* things with objects. For example, if you imagine a Bank Account as an object, it can be made up of a bunch of information, such as Account Number, Balance and Overdraft Limit. Now imagine the things you can *do* with a bank account: you can Deposit or Withdraw money. These actions are tightly related to bank accounts, largely because they directly affect the information in the bank account. These actions are carried out as a ***set of instructions*** – known as ***methods*** – which are carried out in a prescribed way through a set of one or more ***program statements***.

Two objects can have the same set of characteristics, but different information. For example, consider two Bank Account objects where one account has a balance of $4,500.00 and another has a balance of $125.00. The actual information is different, but the characteristics (what Bank Accounts “look like”) remain the same. The things you can do with the two bank accounts also remain the same: you can Deposit and Withdraw money. In programs, the characteristics of an object are defined in something called a ***Class***. A class defines

* What an object “looks like” (information)
* How an object “behaves” (instructions to manipulate information)

For a computer program to be able to work with objects, it must first know the ***class definition*** of that object. Coding a class definition is like telling the computer about a new ***data type*** that you are defining. The name of the class is the name of the new data type. Once the data type is defined, then it can be used to create objects. A class definition (or “class” for short) acts as a blueprint for creating objects; from the blueprint, the computer is told what the object “looks like” and how it “behaves”. Every object is said to be an “instance” of a class, in the same way that a physical car would be an instance of the car’s blueprint. The term ***instantiation*** basically means “to instantiate”, and it refers to the act of creating objects in code. An object is said to be “based on” a class when that class is the data type from which an object has been instantiated (just as a physical car would be “based on” the car’s blueprint).

So, objects and classes are closely related, but not exactly the same. The relationship between objects and classes is pretty much the same as the relationship between ordinary values and data types.

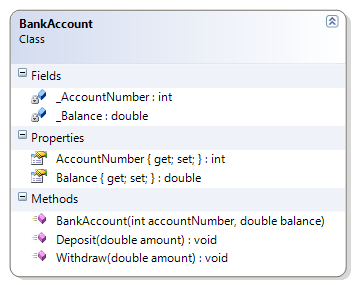
## Coding the Class Definition

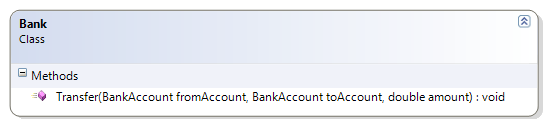
A class definition can contain four parts: ***Fields***, ***Properties***, ***Constructors***, and ***Methods***. Fields and properties work together to describe what an object based on the class “looks like” (the object’s information). Constructors and Methods describe how the object “behaves” (what instructions an object performs to manipulate information).

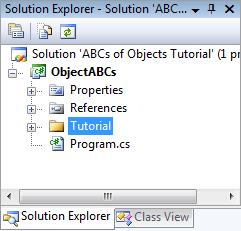
* ***Fields***
  + Fields are variables that are **shared by**/**accessible to** all the other members of a class (properties, constructors, and methods).
  + Fields are almost always declared as ***private*** (meaning that they are not directly accessible from outside the class).
  + Although not commonly done, a field can be assigned an initial default value when it is declared.
* ***Properties***
  + Properties are used like ordinary fields or variables, but work internally as methods.
  + Properties can have a Get method and/or a Set method.
    - The Get method is used when attempting to retrieve a value from the property.
    - The Set method is used when attempting to store a value in the property.
  + Properties often (but not always) have a corresponding field that is used to hold or store the property's value in the "background"; the field associated with a property is often referred to as the "backing store".
  + Properties that do not have a corresponding backing store will only have a Get method that calculates a value based on other values; there is no Set method because there is no single field in which to store a value for the property.
  + Properties are usually declared as ***public*** (meaning that they can be accessed directly from outside the class).
* Constructors
  + Constructors are called automatically whenever an object is created. Objects are created by the new keyword, whose general syntax is as follows:  
    **new** ***ClassName***(***ArgList***)
  + ArgList (or Argument List) is a comma-separated list of values that are sent to the constructor. The decision of which constructor can be called depends on the arguments (values) sent in when the new keyword is used; the compiler looks for a constructor with a parameter list that matches the argument list.
  + Constructors have one purpose: to make sure that the fields in the object have meaningful values. This usually means that a constructor will need some parameters to get those values from the code that creates the object.
  + Usually, constructors are made ***public***.
  + A class can define more than one constructor. If no constructor is defined for a class, then a "default constructor" is used, which simply sets all the fields to their default values.
* Methods
  + A method is a set of instructions that are typically focused on manipulating or working with the fields and properties of an object.
  + Methods are identified by a name (called the Method Name) and a parameter list; together the method name and parameter list make up what is called the ***Method Signature***.
  + If a method has a return type of **void**, then it is known as a ***Subroutine***; subroutines do not return any information to the code that called it.
  + If a method has any return type except **void**, then it is called a ***Function***; functions can only return one item, and must do so using a ***return statement***.

# A Tutorial - Part 1

This tutorial will walk you through the steps to create a class and an object based on the following design. This program will demonstrate a **BankAccount** class and the transfer of money from one bank account to another.





1. Open up Visual Studio and create a new C# Console Application project. In that project, create a folder called “Tutorial”.  
   
2. Right-click on the Tutorial folder, and create a class called “BankAccount”. Edit the BankAccount class so that it has the following code.

namespace ObjectABCs.Tutorial

{

public class BankAccount

{

#region Fields (always private)

private int \_AccountNumber;

private double \_Balance;

#endregion

#region Properties

public int AccountNumber

{

get { return \_AccountNumber; }

private set { \_AccountNumber = value; }

}

public double Balance

{

get { return \_Balance; }

set { \_Balance = value; }

}

#endregion

public BankAccount(int accountNumber, double balance)

{

AccountNumber = accountNumber;

Balance = balance;

}

public void Deposit(double amount)

{

Balance = Balance + amount;

}

public void Withdraw(double amount)

{

Balance = Balance - amount;

}

}

}

1. Create another class inside the Tutorial folder, and name it “Bank”. Edit the class so that it has the following code.

namespace ObjectABCs.Tutorial

{

public class Bank

{

public static void Transfer(BankAccount fromAccount, BankAccount toAccount, double amount)

{

fromAccount.Withdraw(amount);

toAccount.Deposit(amount);

}

}

}

1. Create a driver class named “DemoDriver” and place it in the Tutorial folder. Edit the driver so that it has the following code.

using System;

namespace ObjectABCs.Tutorial

{

public class DemoDriver

{

public static void Start()

{

BankAccount myAccount = new BankAccount(123546, 250.00);

BankAccount storeAccount = new BankAccount(987654, 5473.28);

DisplayAccount(myAccount, "My Account");

DisplayAccount(storeAccount, "Store Account");

Console.WriteLine("I am purchasing an item for $12.95");

Bank.Transfer(myAccount, storeAccount, 12.95);

DisplayAccount(myAccount, "My Account");

DisplayAccount(storeAccount, "Store Account");

Console.WriteLine("Purchase complete");

} // end of Start() method

private static void DisplayAccount(BankAccount account, string message)

{

Console.WriteLine(message);

Console.WriteLine("-- Account Info --");

Console.WriteLine("Acct. #: " + account.AccountNumber);

Console.WriteLine("Balance: $ " + account.Balance);

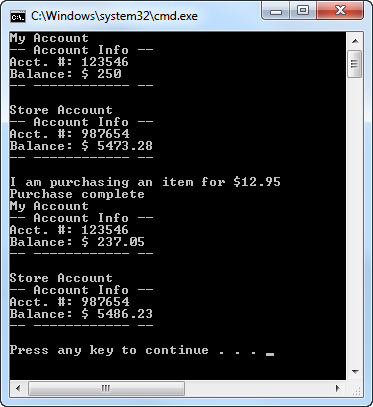
Console.WriteLine("-- ------------ --");

Console.WriteLine();

} // end of DisplayAccount() method

}

}

1. Open the Program.cs file and edit the Main() method to call the demo driver class.

using ObjectABCs.Tutorial;

namespace ObjectABCs

{

class Program

{

static void Main(string[] args)

{

DemoDriver.Start();

}

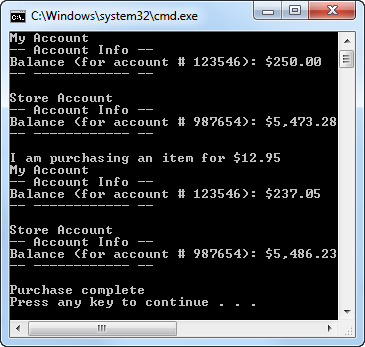
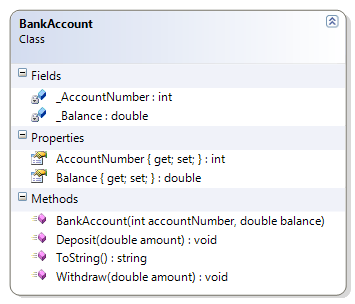
}

}

When you run the program, you should see the result as shown in the screen-shot on the right.

# A Tutorial - Part 2

A big aspect of developing computer programs is the work of modifying existing code that you or others have written. In this part of the tutorial, we'll revisit the BankAccount and DemoDriver classes to add and take advantage of a new method on the BankAccount class: ToString().



1. Open the BankAccount class and modify it by adding the following ToString() method in the body of the class.

public override string ToString()

{

string accountInfo = "Balance (for account # "

+ AccountNumber.ToString()

+ "): "

+ string.Format("{0:C}", Balance);

return accountInfo;

}

1. Open the DemoDriver class and modify the DisplayAccount() method to look like the following code.

private static void DisplayAccount(BankAccount account, string message)

{

Console.WriteLine(message);

Console.WriteLine("-- Account Info --");

Console.WriteLine(account);

Console.WriteLine("-- ------------ --");

Console.WriteLine();

} // end of DisplayAccount() method

# A Tutorial - Part 3

Many times, properties act to provide transparent access to fields. (All of the examples so far have used Properties in this simplest way.) In C# version 3.0, a new syntax was introduced called ***auto-implemented properties*** which reduces the amount of code required to get the same effect as these transparent properties.

Auto-implemented properties do not have a field as a "backing store" and do not have bodies for the get and set methods. For example, the following code demonstrates an auto-implemented property.

public string ClientName { get; private set; }

This property has the exact same effect as the following, longer approach of using a property with a field in the background.

private string \_ClientName;

public string ClientName

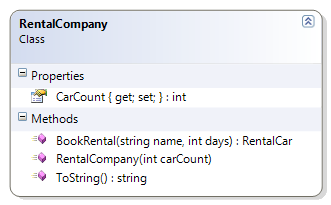
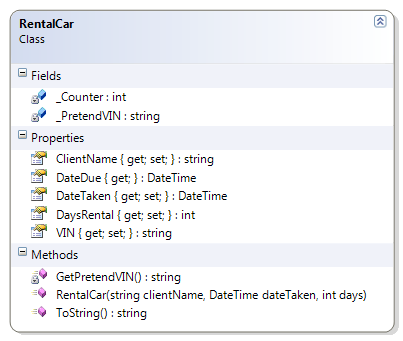
{

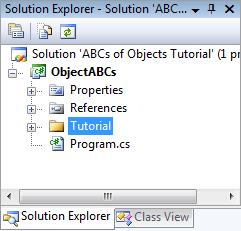
get { return \_ClientName; }

private set { \_ClientName = value; }

}

In this tutorial, you will create classes that use auto-implemented properties, as well as regular properties.



1. Open up Visual Studio and create a new C# Console Application project. In that project, create a folder called “Tutorial”.  
   
2. Create a new class in the Tutorial folder called RentalCar. Edit the class so that it matches the following code.

using System; // needed for DateTime

namespace ObjectABCs.Tutorial

{

public class RentalCar

{

public string VIN { get; private set; }

public string ClientName { get; private set; }

public DateTime DateTaken { get; private set; }

public int DaysRental { get; set; }

public DateTime DateDue

{

get

{

return DateTaken.AddDays(DaysRental);

}

}

public RentalCar(string clientName, DateTime dateTaken, int days)

{

ClientName = clientName;

DateTaken = dateTaken;

DaysRental = days;

// figure out the remaining values

VIN = GetPretendVIN();

}

private static string \_PretendVIN = "1ABC234DE567";

private static int \_Counter = 10;

private static string GetPretendVIN()

{

string tempVIN = \_PretendVIN + \_Counter.ToString();

\_Counter = \_Counter + 1;

return tempVIN;

}

public override string ToString()

{

return VIN;

}

}

}

1. Create another class named RentalCompany. Edit the code to match the following.

using System; // needed for DateTime

namespace ObjectABCs.Tutorial

{

public class RentalCompany

{

public int CarCount { get; private set; }

public RentalCompany(int carCount)

{

CarCount = carCount;

}

public RentalCar BookRental(string name, int days)

{

RentalCar car;

car = new RentalCar(name, DateTime.Today, days);

CarCount = CarCount - 1;

return car;

}

public override string ToString()

{

return CarCount.ToString() + " cars available to rent.";

}

}

}

1. Create a final class named DemoRentals. This will be the driver to use RentalCompany and RentalCar objects. Edit DemoRentals to match the following.

using System;

namespace ObjectABCs.Tutorial

{

public class DemoRentals

{

public static void Start()

{

string name;

int daysToRent;

name = PromptName();

daysToRent = PromptRentalTime();

RentalCar rental;

RentalCompany budget = new RentalCompany(5);

rental = budget.BookRental(name, daysToRent);

DisplayRentalInfo(rental);

}

private static string PromptName()

{

Console.Write("Enter name of person renting vehicle: ");

string name = Console.ReadLine();

return name;

}

private static int PromptRentalTime()

{

Console.Write("Enter number of days to rent: ");

string daysAsText = Console.ReadLine();

int days = Convert.ToInt32(daysAsText);

return days;

}

private static void DisplayRentalInfo(RentalCar rental)

{

Console.WriteLine("Rental Info");

Console.WriteLine("-----------");

Console.WriteLine(rental);

Console.WriteLine("Customer: " + rental.ClientName);

Console.WriteLine("Due Date: " + rental.DateDue.ToShortDateString());

Console.WriteLine();

}

}

}

1. Lastly, modify the Main method in Program.cs to call the DemoRentals.Start() method.

using ObjectABCs.Tutorial;

namespace ObjectABCs

{

class Program

{

static void Main(string[] args)

{

DemoRentals.Start();

}

}

}

1. Run the program. You should get results similar to the following screenshot.  
   