**Creating Classes and Members**  
  
In Visual C#, you can define your own custom types by creating classes. As a programming construct, the class is central to object-oriented programming in Visual C#. It enables you to encapsulate the behaviors and characteristics of any logical entity in a reusable and extensible way.  In this lesson, you will learn how to create, use, and test classes in your own applications.  
  
In Visual C#, a class is a programming construct that you can use to define your own custom types. When you create a class, you are effectively creating a blueprint for the type. The class defines the behaviors and characteristics, or class members, which are shared by all instances of the class. You represent these behaviors and characteristics by defining methods, fields, properties, and events within your class.  
  
Suppose you create a class named DrinksMachine.

You use the class keyword to declare a class, as shown in the following example:  
  
//Declaring a Class  
public class DrinksMachine  
{  
    // Methods, fields, properties, and events go here.  
}  
  
The class keyword is preceded by an access modifier, such as public in the above example, will be described in the Encapsulation section.

**Adding Members to a Class**  
   
You would use fields and properties to define the characteristics of a drinks machine, such as the make, model, age, and service interval of the machine. You would create methods to represent the things that a drinks machine can do, such as make an espresso or make a cappuccino. Finally, you would define events to represent actions that might require your attention, such as replacing coffee beans when the machine has run out of coffee beans.

Within your class, you can add methods, fields, properties, and events to define the behaviors and characteristics of your type, as shown in the following example:

// Defining Class Members  
public class DrinksMachine  
{  
   // The following statements define a property with a private field.  
   private string \_location;  
   public string Location  
   {  
      get  
      {  
         return \_location;  
      }  
      set  
      {  
         if (value != null)   
            \_location = value;  
      }  
   }  
   // The following statements define properties.  
   public string Make {};  
   public string Model {};  
   // The following statements define methods.  
   public void MakeCappuccino()  
   {  
      // Method logic goes here.  
   }  
   public void MakeEspresso()  
   {  
      // Method logic goes here.  
   }  
   // The following statement defines an event. The delegate definition is not shown.  
   public event OutOfBeansHandler OutOfBeans;  
}

**Partial Classes**

C# can also implement partial classes.  Partial classes allow you to split the definition of the class across multiple source files.  Then you compile your application, all of the parts are combined into a single file.

Partial classes are useful when:

* When working on large projects, spreading a class over separate files enables multiple programmers to work on the same class at the same time.
* When working with automatically generated source. Visual Studio uses this approach when your application uses Windows Forms, Web service wrapper code, etc. Microsoft recommends that you do not modify the auto-generated code for these components as it could be overwritten when the application is compiled or the project files changed.  Instead, you can create another portion of the class, as a partial class with the same name, and make your additions and edits there.

An example of using partial classes follows:

public partial class DrinksMachine  
{  
  
   public void MakeCappuccino()  
   {  
      // Method logic goes here.  
   }  
}  
  
public partial class DrinksMachine  
{  
  
   public void MakeEspresso()  
   {  
      // Method logic goes here.  
   }  
}

Note: you can also split structs and interfaces across multiple source files as well.

**Instantiating Classes**  
  
A class is just a blueprint for a type. To use the behaviors and characteristics that you define within a class, you need to create instances of the class. An instance of a class is called an object.

To create a new instance of a class, you use the new keyword, as shown in the following example:

// Instantiating a Class  
DrinksMachine dm = new DrinksMachine();

When you instantiate a class in this way, you are actually doing two things:

* You are creating a new object in memory based on the DrinksMachine type.
* You are creating an object reference named dm that refers to the new DrinksMachine object.

When you create your object reference, instead of explicitly specifying the DrinksMachine type, you can allow the compiler to deduce the type of the object at compile time. This is known as type inference. To use type inference, you create your object reference by using the var keyword, as shown in the following example:

// Instantiating a Class by Using Type Inference  
var dm = new DrinksMachine();

In this case, the compiler does not know in advance the type of the dm variable. When the dm variable is initialized as a reference to a *DrinksMachine* object, the compiler deduces that the type of dm isDrinksMachine. Using type inference in this way causes no change in how your application runs, it is simply a shortcut for you to avoid typing the class name twice. In some circumstances, type inference can make your code easier to read, while in other circumstances it may make your code more confusing. As a general rule, consider using type inference when the type of variable is absolutely clear.

After you have instantiated your object, you can use any of the members—methods, fields, properties, and events—that you defined within the class, as shown in the following example:

// Using Object Members  
var dm = new DrinksMachine();  
dm.Make = "Fourth Coffee";  
dm.Model = "Beancrusher 3000";  
dm.Age = 2;  
dm.MakeEspresso();

This approach to calling members on an instance variable is known as dot notation. You type the variable name, followed by a period, followed by the member name. The IntelliSense feature in Visual Studio will prompt you with member names when you type a period after a variable.