# Creating Classes

## 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 {get; set;}
   public string Model {get; set;}
   // 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.**

Learn About Verified Certificates