**Understanding Constructors**

Given that objects have state (represented by the values of an object’s member variables), a programmer will typically want to assign relevant values to the object’s field data before use. Currently, the Car class demands that the petName and currSpeed fields be assigned on a field-by-field basis. For the current example, this is not too problematic, given that you have only two public data points. However, it is not uncommon for a class to have dozens of fields to contend with.

Clearly, it would be undesirable to author 20 initialization statements to set 20 points of data!

Thankfully, C# supports the use of constructors, which allow the state of an object to be established at the time of creation. A constructor is a special method of a class that is called indirectly when creating an object using the new keyword. However, unlike a “normal” method, constructors never have a return value (not even void) and are always named identically to the class they are constructing.

**The Role of the Default Constructor**

Every C# class is provided with a “freebie” default constructor that you can redefine if need be. By definition, a default constructor never takes arguments. After allocating the new object into memory, the default constructor ensures that all field data of the class is set to an appropriate default value (see Chapter 3 for information regarding the default values of C# data types).

If you are not satisfied with these default assignments, you may redefine the default constructor to suit your needs. To illustrate, update your C# Car class as follows:

class Car {

// The 'state' of the Car. public string petName; public int currSpeed;

// A custom default constructor. public Car()

{

petName = "Chuck";

currSpeed = 10; }

...

}

In this case, you are forcing all Car objects to begin life named Chuck at a rate of 10 mph. With this, you are able to create a Car object set to these default values as follows:

static void Main(string[] args)

{

Console.WriteLine("\*\*\*\*\* Fun with Class Types \*\*\*\*\*\n");

// Invoking the default constructor. Car chuck = new Car();

// Prints "Chuck is going 10 MPH."

chuck.PrintState(); ...

}

**Defining Custom Constructors**

Typically, classes define additional constructors beyond the default. In doing so, you provide the object user with a simple and consistent way to initialize the state of an object directly at the time of creation. Ponder the following update to the Car class, which now supports a total of three constructors:

class Car {

// The 'state' of the Car. public string petName; public int currSpeed;

// A custom default constructor. public Car()

{

petName = "Chuck";

currSpeed = 10; }

// Here, currSpeed will receive the // default value of an int (zero). public Car(string pn)

{

petName = pn; }

// Let caller set the full state of the Car. public Car(string pn, int cs)

{

petName = pn;

currSpeed = cs; }

...

}

Keep in mind that what makes one constructor different from another (in the eyes of the C# compiler) is the number of and/or type of constructor arguments. Recall from Chapter 4, when you define a method of the same name that differs by the number or type of arguments, you have overloaded the method. Thus, the Car class has overloaded the constructor to provide a number of ways to create an object at the time of declaration. In any case, you are now able to create Car objects using any of the public constructors. Here’s an example:

static void Main(string[] args)

{

Console.WriteLine("\*\*\*\*\* Fun with Class Types \*\*\*\*\*\n");

// Make a Car called Chuck going 10 MPH. Car chuck = new Car(); chuck.PrintState();

// Make a Car called Mary going 0 MPH. Car mary = new Car("Mary"); mary.PrintState();

// Make a Car called Daisy going 75 MPH. Car daisy = new Car("Daisy", 75); daisy.PrintState();

...

}

**The Default Constructor Revisited**

As you have just learned, all classes are provided with a free default constructor. Thus, if you insert a new class into your current project named Motorcycle, defined like so:

class Motorcycle

{

  public void PopAWheely()

  {

Console.WriteLine("Yeeeeeee Haaaaaeewww!"); }

}

you are able to create an instance of the Motorcycle type via the default constructor out of the box.

static void Main(string[] args)

{

Console.WriteLine("\*\*\*\*\* Fun with Class Types \*\*\*\*\*\n"); Motorcycle mc = new Motorcycle();

mc.PopAWheely();

...

}

However, as soon as you define a custom constructor with any number of parameters, the default constructor is silently removed from the class and is no longer available. Think of it this way: if you do not define a custom constructor, the C# compiler grants you a default in order to allow the object user to allocate an instance of your type with field data set to the correct default values. However, when you define a unique constructor, the compiler assumes you have taken matters into your own hands.

Therefore, if you want to allow the object user to create an instance of your type with the default constructor, as well as your custom constructor, you must explicitly redefine the default. To this end, understand that in a vast majority of cases, the implementation of the default constructor of a class is intentionally empty, as all you require is the ability to create an object with default values. Consider the following update to the Motorcycle class:

class Motorcycle

{

  public int driverIntensity;

  public void PopAWheely()

  {

for (int i = 0; i <= driverIntensity; i++) {

Console.WriteLine("Yeeeeeee Haaaaaeewww!"); }

}

// Put back the default constructor, which will // set all data members to default vaules. public Motorcycle() {}

// Our custom constructor. public Motorcycle(int intensity) {

driverIntensity = intensity; }

}