Defining a Class

To illustrate classes, we will develop a class that encapsulates information about vehicles, such as cars, vans, and trucks. This class is called **Vehicle**, and it will store three items of information about a vehicle: the number of passengers that it can carry, its fuel capacity, and its average fuel consumption (in miles per gallon).

The first version of **Vehicle** is shown next. It defines three instance variables: **passengers**, **fuelcap**, and **mpg**. Notice that **Vehicle** does not contain any methods. Thus, it is currently a data-only class. (Subsequent sections will add methods to it.)

```
class Vehicle {
  int passengers; // number of passengers
  int fuelcap;    // fuel capacity in gallons
  int mpg;        // fuel consumption in miles per gallon
}
```

A **class** definition creates a new data type. In this case, the new data type is called **Vehicle**. You will use this name to declare objects of type **Vehicle**. Remember that a **class** declaration is only a type description; it does not create an actual object. Thus, the preceding code does not cause any objects of type **Vehicle** to come into existence.

To actually create a **Vehicle** object, you will use a statement like the following:

```
Vehicle minivan = new Vehicle(); // create a Vehicle object called minivan
```

After this statement executes, **minivan** refers to an instance of **Vehicle**. Thus, it will have "physical" reality. For the moment, don't worry about the details of this statement.

Each time you create an instance of a class, you are creating an object that contains its own copy of each instance variable defined by the class. Thus, every **Vehicle** object will contain its own copies of the instance variables **passengers**, **fuelcap**, and **mpg**. To access these variables, you will use the dot (.) operator. The *dot operator* links the name of an object with the name of a member. The general form of the dot operator is shown here:

object.member

Thus, the object is specified on the left, and the member is put on the right. For example, to assign the **fuelcap** variable of **minivan** the value 16, use the following statement:

```
minivan.fuelcap = 16;
```

In general, you can use the dot operator to access both instance variables and methods.

Here is a complete program that uses the **Vehicle** class:

```
// A program that uses the Vehicle class.

class Vehicle {
  int passengers; // number of passengers
  int fuelcap;    // fuel capacity in gallons
  int mpg;        // fuel consumption in miles per gallon
}

// This class declares an object of type Vehicle.
class VehicleDemo {
```

```
  public static void main(String args[]) {
    Vehicle minivan = new Vehicle();
    int range;

    // assign values to fields in minivan
    minivan.passengers = 7;
    minivan.fuelcap = 16; <-- Notice the use of the dot operator to access a member.
    minivan.mpg = 21;

    // compute the range assuming a full tank of gas
    range = minivan.fuelcap * minivan.mpg;
    System.out.println("Minivan can carry " + minivan.passengers +
                       " with a range of " + range);
  }
}
```

To try this program, you can put both the **Vehicle** and **VehicleDemo** classes in the same source file. For example, you could call the file that contains this program **VehicleDemo.java**. This name makes sense because the **main( )** method is in the class called **VehicleDemo**, not the class called **Vehicle**. Either class can be the first one in the file. When you compile this program using **javac**, you will find that two **.class** files have been created, one for **Vehicle** and one for **VehicleDemo**. The Java compiler automatically puts each class into its own **.class** file. It is important to understand that it is not necessary for both the **Vehicle** and the **VehicleDemo** class to be in the same source file. You could put each class in its own file, called **Vehicle.java** and **VehicleDemo.java**, respectively. If you do this, you can still compile the program by compiling **VehicleDemo.java**.

To run this program, you must execute **VehicleDemo.class**. The following output is displayed:

```
Minivan can carry 7 with a range of 336
```

Before moving on, let's review a fundamental principle: each object has its own copies of the instance variables defined by its class. Thus, the contents of the variables in one object can differ from the contents of the variables in another. There is no connection between the two objects except for the fact that they are both objects of the same type. For example, if you have two **Vehicle** objects, each has its own copy of **passengers**, **fuelcap**, and **mpg**, and the contents of these can differ between the two objects. The following program demonstrates this fact. (Notice that the class with **main( )** is now called **TwoVehicles**.)

```
// This program creates two Vehicle objects.

class Vehicle {
  int passengers; // number of passengers
  int fuelcap;    // fuel capacity in gallons
  int mpg;        // fuel consumption in miles per gallon
}

// This class declares an object of type Vehicle.
class TwoVehicles {
  public static void main(String args[]) {
```

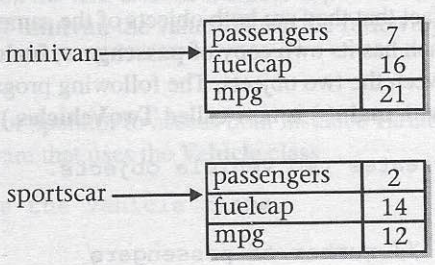
```
    Vehicle minivan = new Vehicle();
    Vehicle sportscar = new Vehicle();

    int range1, range2;

    // assign values to fields in minivan
    minivan.passengers = 7;
    minivan.fuelcap = 16;
    minivan.mpg = 21;

    // assign values to fields in sportscar
    sportscar.passengers = 2;
    sportscar.fuelcap = 14;
    sportscar.mpg = 12;

    // compute the ranges assuming a full tank of gas
    range1 = minivan.fuelcap * minivan.mpg;
    range2 = sportscar.fuelcap * sportscar.mpg;

    System.out.println("Minivan can carry " + minivan.passengers +
                       " with a range of " + range1);

    System.out.println("Sportscar can carry " + sportscar.passengers +
                       " with a range of " + range2);
  }
}
```

Remember, **minivan** and **sportscar** refer to separate objects.

The output produced by this program is shown here:

```
Minivan can carry 7 with a range of 336
Sportscar can carry 2 with a range of 168
```

As you can see, **minivan**'s data is completely separate from the data contained in **sportscar**. The following illustration depicts this situation.



How Objects Are Created

In the preceding programs, the following line was used to declare an object of type **Vehicle**:

```
Vehicle minivan = new Vehicle();
```