OVERLOADING METHODS AND CONSTRUCTORS

METHOD OVERLOADING

Method overloading is when you create more than one version of the same method, but with different **parameter lists**. The compiler determines which method to call based on the name of the method and the number, order, and type of parameters. This combination of name and parameter list is called the **method signature**.

The parameters may vary by the number or by the types. In other words, you can have two versions with one parameter but different types (e.g. a double and a String) or you can have on version that takes a single parameter and a second version that takes two parameters.

You can also overload a method by changing the order of the parameters, if the parameters are of different types:

```
public void myMethod(int age, String name)
public void myMethod(String name, int age)
```

Keep in mind that if the two parameters are of the same type, you will get a compile-time error.

```
public void myMethod(int age, int size)
public void myMethod(int size, int age)
```

These last two methods are indistinguishable to the compiler. The compiler looks only at the types, not at the parameter names.

APPLYING OVERLOADED METHODS TO THE *CAR* CLASS

We can apply the ability to overload methods to our **Car** class. For example, we can add a second method called **move()** that take a parameter that will determine how many pixels to move the car:

```
public void move(int pixels) {

   xPos += pixels;

   if (xPos >= panelWidth)
   {
      xPos = 0 – getCarWidth();
   }
}
```

Now we have two **move()** methods: one that does not take a parameter and moves the car 5 pixels, and one that takes an **int** parameter which represents how many pixels the car will move.

OVERLOADING THE CONSTRUCTOR

We can overload a constructor just like we can overload a method. Constructor overloading gives us a great deal of flexibility as it allows us to create our objects with a variety of different starting data.

For example, we can add constructor overloading to our **Car** class to enable the programmer to use any image file to create their car. Previously we assigned a default image file to the car:

```
public Car() {

   imgCar = new ImageIcon("images\\car.png").getImage();
}
```

One way to overload the constructor and provide more flexibility in your class is to allow the programmer to specify an image file. For example, we can provide an implementation of the constructor that takes a **String** parameter which will represent the location and name of the picture file:

```
public Car(String filename) {

   imgCar = new ImageIcon(filename).getImage();
}
```

We will continue to allow users to initialize a **Car** object with no parameters, but now we will also allow users to pass in a **String** value for the name of the file.