STATIC METHODS AND STATIC VARIABLES

STATIC VARIABLES

A **static variable** is a variable that belongs to the class as a whole and not just to one object. Each object has its own copies of the instance variables. However, with a static variable there is only one copy of the variable, and all the objects can use this one variable. Thus, a static variable can be used by objects to communicate between the objects. One object can change the static variable, and another object can read that change. To make a variable static, you declare it like an instance variable but add the modifier **static** as follows:

```
public static int turn;
```

Static variables are created and initialized when the class is first loaded – that is, when a static member of the class is referred to or when an instance of the class is created.

Let's say, for example, I want to create a static variable in the **Dice** class that we created in the previous lesson, to keep count of the number of times the **Dice** object is rolled. The variable would be declared as follows:

```
public static int totalRolls;
```

The variable would then be initialized in the constructor as follows:

```
public Dice() {

   totalRolls = 0;
   ...
}
```

You can access the static variable without creating an instance of the class as follows:

```
System.out.println(Dice.totalRolls);
```

The above program will output the value of the static variable **totalRolls** to the system console.

STATIC CONSTANTS

Constants are variables with values that can't be changed. The value is assigned to a constant when it is declared. Many classes, including many in the Java core libraries, contain static constants that are used to give names to certain values so that their meaning is more obvious. For example, the **javax.swing.JOptionPane** class contains numerous constants to represent different options that are available to a user when interacting with a message or input dialog box (e.g. **JOptionPane.YES_OPTION**, **JOptionPane.NO_OPTION**, etc.).

If we were to look at the **JOptionPane** class, we would see each of these variables declared as follows:

```
public static final int YES_OPTION = 0;
public static final int NO_OPTION = 1;
public static final int CANCEL_OPTION = 2;
```

Within the class you would also find a method called **showConfirmDialog()** that returns an integer representing the user's selection. The programmer uses the constants included in the class to help determine the user's selection, as follows:

```
if (choice == JOptionPane.YES_OPTION) {
   ...
}
```

The meaning of the number 0 within the **JOptionPane** class is not obvious, but if it was given the name **YES_OPTION** then you know immediately what it means.

STATIC METHODS

A **static method**, like a static variable, is a method that isn't associated with an instance of a class; rather, the method belongs to the class itself. A static method, therefore, can be called without having to first create an instance of the class to which the method belongs. This means that you don't have to create an instance of the class to access a static variable or method. You access a static variable or method by specifying the class name, not a variable that references an object.

If, for example, we wanted to create a static method called **getTotalRolls()** for our **Dice** class that returns the number of total rolls, we would create the method as follows:

```
public static int getTotalRolls() {

   return totalRolls;
}
```

We could then use this static method to get the total number of flips as follows:

```
System.out.println("TOTAL FLIPS: " + Dice.getTotalRolls());
```

One of the basic rules of working with static methods is that you can't access a non-static method or instance variable from a static method because the static method does not have an instance of the class to use to reference instance methods or variables.

The following program, for example, will generate a compile-time error because it is trying to access an instance variable (i.e. **mark**) within a static method **calculateAverage()**:

```
public class Test {

   private int mark = 30;

   public static double calculateAverage() {
         return mark / 35;
      }
   }
```

You can, however, access static methods and variables from non-static methods:

```
   public class Test {

      private static int mark = 30;

      public double calculateAverage() {

         return mark / 35;
      }
   }
```

WHEN TO USE INSTANCE OR STATIC

Now that we've looked at the difference between static and non-static (or instance) variables and methods, the question that remains is when to use one over the other. The following is a useful guideline that you can use when making such a decision:

- A variable or method that is dependent on a specific instance of the class should be an **instance** variable or method.
- A variable or method that is not dependent on a specific instance of the class should be a **static** variable or method.