Partial extract/modification from **The Java™ Tutorials**

**Understanding Class Members**

In this section, we discuss the use of the static keyword to create fields and methods that belong to the class, rather than to an instance of the class.

**Class Variables**

When a number of objects are created from the same class blueprint, they each have their own distinct copies of *instance variables*. In the case of the Bicycle class, the instance variables are cadence, gear, and speed. Each Bicycle object has its own values for these variables, stored in different memory locations.

Sometimes, you want to have variables that are common to all objects. This is accomplished with the static modifier. Fields that have the static modifier in their declaration are called *static fields* or *class variables*. They are associated with the class, rather than with any object. Every instance of the class shares a class variable, which is in one fixed location in memory. Any object can change the value of a class variable, but class variables can also be manipulated without creating an instance of the class.

For example, suppose you want to create a number of Bicycle objects and assign each a serial number, beginning with 1 for the first object. This ID number is unique to each object and is therefore an instance variable. At the same time, you need a field to keep track of how many Bicycle objects have been created so that you know what ID to assign to the next one. Such a field is not related to any individual object, but to the class as a whole. For this you need a class variable, numberOfBicycles, as follows:

public class Bicycle {

private int cadence;

private int gear;

private int speed;

// **add an instance variable for the object ID**

private int id;

// **add a class variable for the**

// **number of Bicycle objects instantiated**

private **static** int numberOfBicycles = 0;

...

}

Class variables are referenced by the class name itself, as in

Bicycle.numberOfBicycles

This makes it clear that they are class variables.

**Note:** You can also refer to static fields with an object reference like

myBike.numberOfBicycles

but this is discouraged because it does not make it clear that they are class variables.

You can use the Bicycle constructor to set the id instance variable and increment the numberOfBicycles class variable:

public class Bicycle {

private int cadence;

private int gear;

private int speed;

private int id;

private static int numberOfBicycles = 0;

public Bicycle(int startCadence, int startSpeed, int startGear){

gear = startGear;

cadence = startCadence;

speed = startSpeed;

// **increment number of Bicycles**

// **and assign ID number**

**id = ++numberOfBicycles;**

}

// **new instance method to return the ID instance variable**

public int getID() {

return id;

}

...

}

**Class Methods**

The Java programming language supports static methods as well as static variables. Static methods, which have the static modifier in their declarations, should be invoked with the class name, without the need for creating an instance of the class, as in

ClassName.methodName(args)

**Note:** You can also refer to static methods with an object reference like

instanceName.methodName(args)

but this is discouraged because it does not make it clear that they are class methods.

A common use for static methods is to access static fields. For example, we could add a static method to the Bicycle class to access the numberOfBicycles static field:

public **static** int getNumberOfBicycles() {

return numberOfBicycles;

}

**Instance Methods**

Instance methods, which do not have the static modifier in their declarations, are invoked with the object identifier.

objectName.methodName(args)

int myBikeId = myBike.getID()