**Classes**

The introduction to object-oriented concepts in the lesson titled Object-oriented Programming Concepts used a bicycle class as an example, with racing bikes, mountain bikes, and tandem bikes as subclasses. Here is sample code for a possible implementation of a Bicycle class, to give you an overview of a class declaration. Subsequent sections of this lesson will back up and explain class declarations step by step. For the moment, don't concern yourself with the details.

public class Bicycle {

// the Bicycle class has

// three fields

public int cadence;

public int gear;

public int speed;

// the Bicycle class has

// one constructor

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

gear = startGear;

cadence = startCadence;

speed = startSpeed;

}

// the Bicycle class has

// four methods

public void setCadence(int newValue) {

cadence = newValue;

}

public void setGear(int newValue) {

gear = newValue;

}

public void applyBrake(int decrement) {

speed -= decrement;

}

public void speedUp(int increment) {

speed += increment;

}

}

A class declaration for a MountainBike class that is a subclass of Bicycle might look like this:

public class MountainBike extends Bicycle {

// the MountainBike subclass has

// one field

public int seatHeight;

// the MountainBike subclass has

// one constructor

public MountainBike(int startHeight, int startCadence,

int startSpeed, int startGear) {

super(startCadence, startSpeed, startGear);

seatHeight = startHeight;

}

// the MountainBike subclass has

// one method

public void setHeight(int newValue) {

seatHeight = newValue;

}

}

MountainBike inherits all the fields and methods of Bicycle and adds the field seatHeight and a method to set it (mountain bikes have seats that can be moved up and down as the terrain demands).

# Declaring Classes

You've seen classes defined in the following way:

class *MyClass* {

// field, constructor, and

// method declarations

}

This is a *class declaration*. The class body (the area between the braces) contains all the code that provides for the life cycle of the objects created from the class: constructors for initializing new objects, declarations for the fields that provide the state of the class and its objects, and methods to implement the behavior of the class and its objects.

The preceding class declaration is a minimal one. It contains only those components of a class declaration that are required. You can provide more information about the class, such as the name of its superclass, whether it implements any interfaces, and so on, at the start of the class declaration. For example,

class *MyClass extends MySuperClass implements YourInterface* {

// field, constructor, and

// method declarations

}

means that MyClass is a subclass of MySuperClass and that it implements the YourInterface interface.

You can also add modifiers like *public* or *private* at the very beginning—so you can see that the opening line of a class declaration can become quite complicated. The modifiers *public* and *private*, which determine what other classes can access MyClass, are discussed later in this lesson. The lesson on interfaces and inheritance will explain how and why you would use the *extends* and *implements* keywords in a class declaration. For the moment you do not need to worry about these extra complications.

In general, class declarations can include these components, in order:

1. Modifiers such as *public*, *private*, and a number of others that you will encounter later. (However, note that the *private* modifier can only be applied to [Nested Classes](https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html" \t "https://docs.oracle.com/javase/tutorial/java/javaOO/_top).)
2. The class name, with the initial letter capitalized by convention.
3. The name of the class's parent (superclass), if any, preceded by the keyword *extends*. A class can only *extend* (subclass) one parent.
4. A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword *implements*. A class can *implement* more than one interface.
5. The class body, surrounded by braces, {}.

Declaring Classes

You've seen classes defined in the following way:

class MyClass {

// field, constructor, and

// method declarations

}

This is a class declaration. The class body (the area between the braces) contains all the code that provides for the life cycle of the objects created from the class: constructors for initializing new objects, declarations for the fields that provide the state of the class and its objects, and methods to implement the behavior of the class and its objects.

The preceding class declaration is a minimal one. It contains only those components of a class declaration that are required. You can provide more information about the class, such as the name of its superclass, whether it implements any interfaces, and so on, at the start of the class declaration. For example,

class MyClass extends MySuperClass implements YourInterface {

// field, constructor, and

// method declarations

}

means that MyClass is a subclass of MySuperClass and that it implements the YourInterface interface.

You can also add modifiers like public or private at the very beginning—so you can see that the opening line of a class declaration can become quite complicated. The modifiers public and private, which determine what other classes can access MyClass, are discussed later in this lesson. The lesson on interfaces and inheritance will explain how and why you would use the extends and implements keywords in a class declaration. For the moment you do not need to worry about these extra complications.

In general, class declarations can include these components, in order:

Modifiers such as public, private, and a number of others that you will encounter later. (However, note that the private modifier can only be applied to Nested Classes.)

The class name, with the initial letter capitalized by convention.

The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.

A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.

The class body, surrounded by braces, {}.

# Declaring Member Variables

There are several kinds of variables:

* Member variables in a class—these are called *fields*.
* Variables in a method or block of code—these are called *local variables*.
* Variables in method declarations—these are called *parameters*.

The Bicycle class uses the following lines of code to define its fields:

public int cadence;

public int gear;

public int speed;

Field declarations are composed of three components, in order:

1. Zero or more modifiers, such as public or private.
2. The field's type.
3. The field's name.

The fields of Bicycle are named cadence, gear, and speed and are all of data type integer (int). The public keyword identifies these fields as public members, accessible by any object that can access the class.

## Access Modifiers

The first (left-most) modifier used lets you control what other classes have access to a member field. For the moment, consider only public and private. Other access modifiers will be discussed later.

* public modifier—the field is accessible from all classes.
* private modifier—the field is accessible only within its own class.

In the spirit of encapsulation, it is common to make fields private. This means that they can only be *directly* accessed from the Bicycle class. We still need access to these values, however. This can be done *indirectly* by adding public methods that obtain the field values for us:

public class Bicycle {

private int cadence;

private int gear;

private int speed;

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

gear = startGear;

cadence = startCadence;

speed = startSpeed;

}

public int getCadence() {

return cadence;

}

public void setCadence(int newValue) {

cadence = newValue;

}

public int getGear() {

return gear;

}

public void setGear(int newValue) {

gear = newValue;

}

public int getSpeed() {

return speed;

}

public void applyBrake(int decrement) {

speed -= decrement;

}

public void speedUp(int increment) {

speed += increment;

}

}

## Types

All variables must have a type. You can use primitive types such as int, float, boolean, etc. Or you can use reference types, such as strings, arrays, or objects.

## Variable Names

All variables, whether they are fields, local variables, or parameters, follow the same naming rules and conventions that were covered in the Language Basics lesson, [Variables—Naming](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html" \l "naming" \t "https://docs.oracle.com/javase/tutorial/java/javaOO/_top).

In this lesson, be aware that the same naming rules and conventions are used for method and class names, except that

* the first letter of a class name should be capitalized, and
* the first (or only) word in a method name should be a verb.