Object-Oriented Programming Concepts

CCVT-2018

Ravi Tomar

Assistant Professor(Senior Scale)
Department of Analytics
School of Computer Science
University of Petroleum & Energy Studies

January 16, 2018

What Is an Object?

An object is a software bundle of related state and behavior. Software objects are often used to model the real-world objects that you find in everyday life.We will learn how state and behavior are represented within an object, introduces the concept of data encapsulation, and explains the benefits of designing your software in this manner

What Is an Object?

Real-world objects share two characteristics: They all have state and behavior.

- Dogs have state (name, color, breed, hungry) and behavior (barking, fetching, wagging tail).
- Bicycles also have state (current gear, current pedal cadence, current speed) and behavior (changing gear, changing pedal cadence, applying brakes).

Identifying the state and behavior for real-world objects is a great way to begin thinking in terms of object-oriented programming.

What Is an Object?

For each object that you see, ask yourself two questions:

- "What possible states can this object be in?"
- and "What possible behavior can this object perform?"

What Is an Object?

you'll notice that real-world objects vary in complexity:

- your desktop lamp may have only two possible states (on and off) and two possible behaviors (turn on, turn off).
- your desktop radio might have additional states (on, off, current volume, current station) and behavior (turn on, turn off, increase volume, decrease volume, seek, scan, and tune).
- You may also notice that some objects, in turn, will also contain other objects. These real-world observations all translate into the world of object-oriented programming.

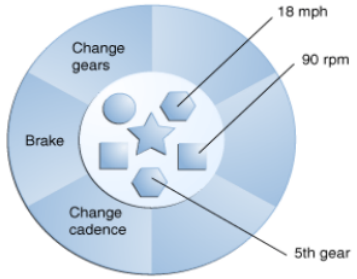
What Is an Object?

Software objects are conceptually similar to real-world objects: they too consist of state and related behavior.

- An object stores its state in fields (variables in some programming languages) and exposes its behavior through methods (functions in some programming languages).
- Methods operate on an object's internal state and serve as the primary mechanism for object-to-object communication.
- Hiding internal state and requiring all interaction to be performed through an object's methods is known as ***data encapsulation*** — a fundamental principle of object-oriented programming.

What Is an Object?

Consider a bicycle, for example:



A bicycle modeled as a software object.

Figure: Bicycle Object

By attributing state (current speed, current pedal cadence, and current gear) and providing methods for changing that state, the object remains in control of how the outside world is allowed to use it. For example, if the bicycle only has 6 gears, a method to change gears could reject any value that is less than 1 or greater than 6.

What Is an Object?

Bundling code into individual software objects provides a number of benefits, including:

- Modularity: The source code for an object can be written and maintained independently of the source code for other objects. Once created, an object can be easily passed around inside the system.
- Information-hiding: By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world.
- Code re-use: If an object already exists (perhaps written by another software developer), you can use that object in your program. This allows specialists to implement/test/debug complex, task-specific objects, which you can then trust to run in your own code.
- Pluggability and debugging ease: If a particular object turns out to be problematic, you can simply remove it from your application and plug in a different object as its replacement. This is analogous to fixing mechanical problems in the real world. If a bolt breaks, you replace it, not the entire machine.

What Is a Class?

In the real world, you'll often find many individual objects all of the same kind. There may be thousands of other bicycles in existence, all of the same make and model. Each bicycle was built from the same set of blueprints and therefore contains the same components. In object-oriented terms, we say that your bicycle is an instance of the class of objects known as bicycles. A class is the blueprint from which individual objects are created

What Is a Class?

```
class Bicycle {

    int cadence = 0;
    int speed = 0;
    int gear = 1;

    void changeCadence(int newValue) {
         cadence = newValue;
    }

    void changeGear(int newValue) {
         gear = newValue;
    }

    void speedUp(int increment) {
         speed = speed + increment;
    }

    void applyBrakes(int decrement) {
         speed = speed - decrement;
    }

    void printStates() {
         System.out.println("cadence:" +
             cadence + " speed:" +
             speed + " gear:" + gear);
    }
```

What Is a Class?

The syntax of the Java programming language will look new to you, but the design of this class is based on the previous discussion of bicycle objects.

- The fields cadence, speed, and gear represent the object's state, and
- the methods (changeCadence, changeGear, speedUp etc.) define its interaction with the outside world.

You may have noticed that the Bicycle class does not contain a main method. That's because it's not a complete application; it's just the blueprint for bicycles that might be used in an application. The responsibility of creating and using new Bicycle objects belongs to some other class in your application.

What Is a Class?

```
class BicycleDemo {
    public static void main(String[] args) {

        // Create two different
        // Bicycle objects
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();

        // Invoke methods on
        // those objects
        bike1.changeCadence(50);
        bike1.speedUp(10);
        bike1.changeGear(2);
        bike1.printStates();

        bike2.changeCadence(50);
        bike2.speedUp(10);
        bike2.changeGear(2);
        bike2.changeCadence(40);
        bike2.speedUp(10);
        bike2.changeGear(3);
        bike2.printStates();
    }
}
```

What Is a Class?

OUTPUT

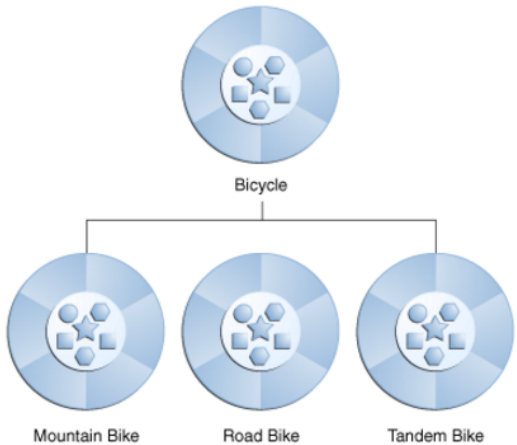
cadence:50 speed:10 gear:2
cadence:40 speed:20 gear:3

What Is Inheritance?

Different kinds of objects often have a certain amount in common with each other. Mountain bikes, road bikes, and tandem bikes, for example, all share the characteristics of bicycles (current speed, current pedal cadence, current gear). Yet each also defines additional features that make them different: tandem bicycles have two seats and two sets of handlebars; road bikes have drop handlebars; some mountain bikes have an additional chain ring, giving them a lower gear ratio.

What Is Inheritance?

Object-oriented programming allows classes to inherit commonly used state and behavior from other classes. In this example, Bicycle now becomes the superclass of MountainBike, RoadBike, and TandemBike. In the Java programming language, each class is allowed to have one direct superclass, and each superclass has the potential for an unlimited number of subclasses:



What Is Inheritance?

The syntax for creating a subclass is simple. At the beginning of your class declaration, use the extends keyword, followed by the name of the class to inherit from:

```
class MountainBike extends Bicycle {
// new fields and methods defining
// a mountain bike would go here
}
```

What Is an Interface?

As you've already learned, objects define their interaction with the outside world through the methods that they expose. Methods form the object's interface with the outside world; the buttons on the front of your television set, for example, are the interface between you and the electrical wiring on the other side of its plastic casing. You press the "power" button to turn the television on and off.

What Is an Interface?

In its most common form, an interface is a group of related methods with empty bodies. A bicycle's behavior, if specified as an interface, might appear as follows:

```
public interface Bicycle {
// wheel revolutions per minute
void changeCadence(int newValue);
void changeGear(int newValue);
void speedUp(int increment);
void applyBrakes(int decrement);
}
```

What Is an Interface?

To implement this interface, the name of your class would change (to a particular brand of bicycle, for example, such as ATLASBicycle), and you'd use the **implements** keyword in the class declaration:

```
class ACMEBicycle implements Bicycle {

int cadence = 0;
int speed = 0;
int gear = 1;

// The compiler will now require that methods
// changeCadence, changeGear, speedUp, and applyBrakes
// all be implemented. Compilation will fail if those
// methods are missing from this class.

void changeCadence(int newValue) {
cadence = newValue;
}

void changeGear(int newValue) {
gear = newValue;
}

void speedUp(int increment) {
speed = speed + increment;

}
void applyBrakes(int decrement) {
speed = speed - decrement;
```

What Is an Interface?

Implementing an interface allows a class to become more formal about the behavior it promises to provide. Interfaces form a contract between the class and the outside world, and this contract is enforced at build time by the compiler. If your class claims to implement an interface, all methods defined by that interface must appear in its source code before the class will successfully compile.

What Is a Package?

A package is a namespace that organizes a set of related classes and interfaces. Conceptually you can think of packages as being similar to different folders on your computer. For e.g.:

- You might keep HTML pages in one folder, images in another, and scripts or applications in yet another.

Because software written in the Java programming language can be composed of hundreds or thousands of individual classes, it makes sense to keep things organized by placing related classes and interfaces into packages.

What Is a Package?

- The Java platform provides an enormous class library (a set of packages) suitable for use in your own applications.
- This library is known as the "Application Programming Interface", or "API" for short.
- packages represent the tasks most commonly associated with general-purpose programming. For example:

» a String object contains state and behavior for character strings;
» a File object allows a programmer to easily create, delete, inspect, compare, or modify a file on the filesystem;
» a Socket object allows for the creation and use of network sockets;
This allows you, the programmer, to focus on the design of your particular application, rather than the infrastructure required to make it work.

We have learnt about:

- Objects
- Class
- Inheritence
- Interface
- Package

Thank You