
Java OOP

(SE Tutorials: Learning the Java Language Trail : Object-Oriented Programming Concepts Lesson)

Dongwon Jeong
djeong@kunsan.ac.kr; <http://ist.kunsan.ac.kr/>

Information Sciences and Technology Laboratory,
Dept. of Informatics & Statistics,
Kunsan National University

Contents

- ❖ **Scope**
- ❖ **OOP Concepts: Key Parts**
- ❖ **What Is an Object?**
- ❖ **What Is a Class?**
- ❖ **What Is Inheritance?**
- ❖ **What Is an Interface?**
- ❖ **What Is a Package?**
- ❖ **Q/A**

Scope

❖ Ref.

- ✓ The Java Tutorials
→ Trail: Learning the Java Language
- ✓ <http://java.sun.com/docs/books/tutorial/java/index.html>

❖ This trail covers the fundamentals of programming in the Java programming language

- ✓ **Object-Oriented Programming Concepts**
teaches you the core concepts behind object-oriented programming: objects, messages, classes, and inheritance. This lesson ends by showing you how these concepts translate into code. Feel free to skip this lesson if you are already familiar with object-oriented programming.
- ✓ **Language Basics**
describes the traditional features of the language, including variables, arrays, data types, operators, and control flow.
- ✓ **Classes and Objects**
describes how to write the classes from which objects are created, and how to create and use the objects.

Scope (cont.)

- ✓ **Interfaces and Inheritance**
describes interfaces—what they are, why you would want to write one, and how to write one. This section also describes the way in which you can derive one class from another. That is, how a *subclass* can inherit fields and methods from a *superclass*. You will learn that all classes are derived from the Object class, and how to modify the methods that a subclass inherits from superclasses.
- ✓ **Numbers and Strings**
This lesson describes how to use Number and String objects. The lesson also shows you how to format data for output.
- ✓ **Generics**
are a powerful feature of the Java programming language. They improve the type safety of your code, making more of your bugs detectable at compile time.
- ✓ **Packages**
are a feature of the Java programming language that help you to organize and structure your classes and their relationships to one another.

OOP Concepts: Key Parts

❖ Lesson: Object-Oriented Programming Concepts

- ✓ <http://java.sun.com/docs/books/tutorial/java/concepts/index.html>

❖ 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.
- ✓ This lesson explains 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 a Class?

- ✓ A class is a blueprint or prototype from which objects are created.
- ✓ This section defines a class that models the state and behavior of a real-world object.
- ✓ It intentionally focuses on the basics, showing how even a simple class can cleanly model state and behavior.

OOP Concepts: Key Parts (cont.)

❖ What Is Inheritance?

- ✓ Inheritance provides a powerful and natural mechanism for organizing and structuring your software.
- ✓ This section explains how classes inherit state and behavior from their superclasses, and explains how to derive one class from another using the simple syntax provided by the Java programming language.

❖ What Is an Interface?

- ✓ An interface is a contract between a class and the outside world.
- ✓ When a class implements an interface, it promises to provide the behavior published by that interface.
- ✓ This section defines a simple interface and explains the necessary changes for any class that implements it.

OOP Concepts: Key Parts (cont.)

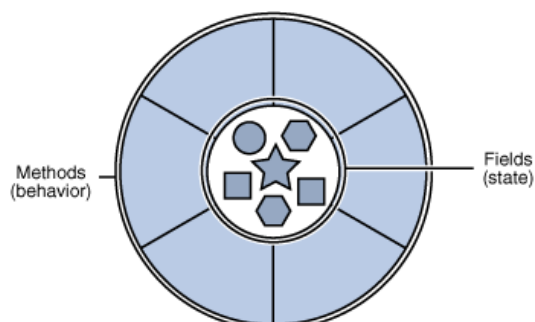
❖ What Is a Package?

- ✓ A package is a namespace for organizing classes and interfaces in a logical manner.
- ✓ Placing your code into packages makes large software projects easier to manage.
- ✓ This section explains why this is useful, and introduces you to the Application Programming Interface (API) provided by the Java platform.

What Is an Object?

❖ **Objects**

- ✓ key to understanding object-oriented technology.
- ✓ Examples of real-world objects: your dog, your desk, your television set, your bicycle.
- ✓ 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).



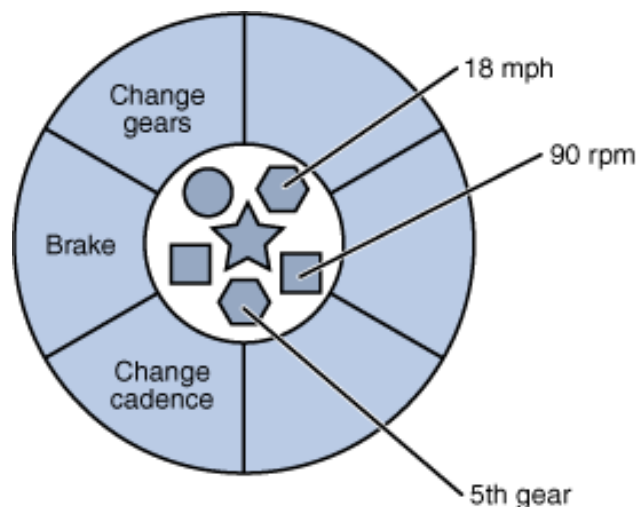
What Is an Object? (cont.)

❖ 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? (cont.)

❖ Consider a bicycle, for example:



What Is an Object? (cont.)

❖ 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*

- ✓ the blueprint from which individual objects are created.

What Is a Class? (cont.)

The following `Bicycle` class is one possible implementation of a bicycle:

```
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? (cont.)

- ❖ **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.

- ❖ **Here's a `BicycleDemo` class that creates two separate `Bicycle` objects and invokes their methods:**
 - ✓ Next Slide

What Is a Class? (cont.)

```
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? (cont.)

- ❖ **The output of this test prints the ending pedal cadence, speed, and gear for the two bicycles:**
 - ✓ cadence:50 speed:10 gear:2
 - ✓ cadence:40 speed:20 gear:3

What Is Inheritance?

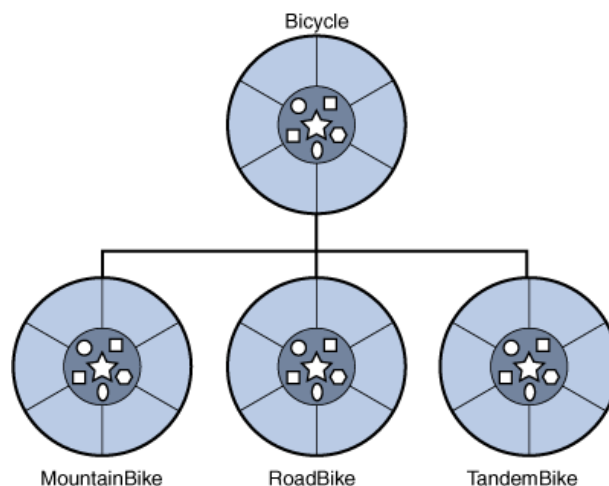
❖ Different kinds of objects often have a certain amount in common with each other.

- ✓ 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? (cont.)

❖ 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? (cont.)

❖ 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? (cont.)

- ❖ 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:

```
interface Bicycle {  
    void changeCadence(int newValue);  
    void changeGear(int newValue);  
    void speedUp(int increment);  
    void applyBrakes(int decrement);  
}
```

- ❖ To implement this interface, the name of your class would change (to `ACMEBicycle`, for example), and you'd use the `implements` keyword in the class declaration:

```
class ACMEBicycle implements Bicycle {  
    // remainder of this class implemented as before  
}
```

What Is an Interface? (cont.)

- ❖ **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 an Interface? (cont.)

❖ Note:

- ✓ To actually compile the ACMEBicycle class,
- ✓ you'll need to add the public keyword to the beginning of the implemented interface methods.
- ✓ You'll learn the reasons for this later in the lessons on [Classes and Objects](#) and [Interfaces and Inheritance](#).

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.
- ✓ 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? (cont.)

- ❖ **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.
- ❖ **Its 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;
 - ✓ various GUI objects control buttons and checkboxes and anything else related to graphical user interfaces.

What Is a Package? (cont.)

- ❖ **There are literally thousands of classes to choose from.**
 - ✓ This allows you, the programmer, to focus on the design of your particular application, rather than the infrastructure required to make it work.
- ❖ **The Java Platform API Specification**
 - ✓ contains the complete listing for all packages, interfaces, classes, fields, and methods supplied by the Java Platform 6, Standard Edition.
 - ✓ Load the page in your browser and bookmark it.
 - ✓ As a programmer, it will become your single most important piece of reference documentation.



Dongwon Jeong

djeong@kunsan.ac.kr; <http://ist.kunsan.ac.kr>

Information Sciences & Technology Laboratory,
Informatics & Statistics Department,
Kunsan National University