原文地址：http://java.sun.com/docs/books/tutorial/java/concepts/index.html

翻译整理：Daniel<nevernet@msn.com>

What Is an Object?

什么是一个 Object(对象)

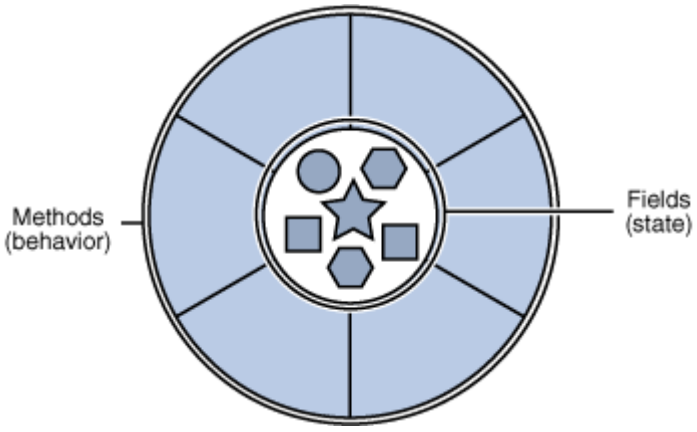
Objects are key to understanding *object-oriented* technology. Look around right now and you'll find many examples of real-world objects: your dog, your desk, your television set, your bicycle.
Objects 是理解面向对象技术的一个关键点。现在就看看你的周围，你会发现很多真实的例子：比如你的狗，桌子，电视机，自行车等等。

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.
真实世界的对象有两个相同的特征：状态(state)和行为(behavior)，狗有状态（名字，颜色，品种，饥饿）和行为（狗叫，抓，摆尾）；自行车也有状态（当前的齿轮，踏板，速度）和行为（更换齿轮，改变踏板节奏，刹车）。识别现实世界种对象的状态和行为是一个"理解和思考面向对象编程"很好的办法。

Take a minute right now to observe the real-world objects that are in your immediate area. 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?". Make sure to write down your observations. As you do, 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), but 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.

立刻花费几分钟观察你当前所在环境里面的对象，针对你看到的每个对象，对自己问两个问题："这个对象可能有哪些状态？"和"这个对象可能会执行哪些动作？"。请务必记录下你观察到的结果。就象你所做的那样，你会注意到现实世界的对象是非常复杂的，你的台灯或许只有两个状态（开和关）和两个动作（打开和关闭），但是你的收音机或许会有额外的状态

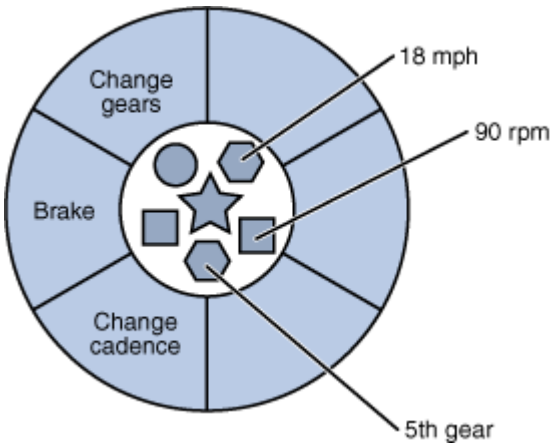
（开，关，当前音量，当前频率）和额外的动作（打开，关闭，调高音量，调低音量，搜索，扫描，调频）。你或许也注意到了，某些对象也会包含其他的对象。这就是把现实世界的观察结果转变到面对对象编程的世界。



A software 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.
软件里面对象的概率和现实世界的对象是相似的：他们也是由状态和相关的动作组成。一个对象（软件里面的对象）用字段（fields）来保存它们的状态（某些语言是用变量-variables），通过方法(methods)来表现它们的动作（某些语言是用函数-function），方法操作对象内部的状态和服务于原始对象和对象之间的通信。通过对象的方法，隐藏了内部的状态和执行了相关的操作，这就是传说中的数据封装 --------- 一个面向对象的基本原理。

Consider a bicycle, for example:



A bicycle modeled as a software 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.
通过总结分析状态（当前速度，当前踏板节奏--感觉是档位，和当前齿轮），提供一些方法来改变状态，对象控制着外部世界如何来使用它。
举例：加入自行车只有6个齿轮，那么改变齿轮的方法能够接受的值只能是大于等于和小于等于6.

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

1 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.
2 Information-hiding: By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world.
3 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.
4 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.

把代码写到一个单独的软件对象里面，我们得到的好处如下：
1. 模块化：这个对象的源代码可以独立的进行编写和维护。一旦创建了，这个对象可以很容易地在系统内部传递。
2. 信息隐藏：仅仅通过对象之间的方法来互相操作，而他们内部的详细信息依然对外部世界进行隐藏了。
3. 代码可复用：如果一个对象已经存在（可能是其他开发人员写的），你也可以在你的程序里面使用它。这样允许特殊的运行、测试、调试复杂的，特殊任务的对象，之后你可以信任这些代码并在你的代码中使用他们。
4. 可插行和容易调试：如果某个对象出了问题，你可以移除这个对象，并用一个新的不同的对象来代替它。这点类似修复现实世界的机械问题。如果某个螺钉坏了，你可以替换这个螺钉，而不用替换整个机器。

What Is a Class?

什么是一个类?

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.

在现实世界里面，你经常会发现同一个类型的很多个体。这里可能会有成千上万的自行车存在，它们有同样的牌子和型号。每个自行车是根据同一个图纸做出来的，所以它们包含同样的零件。用面向对象的术语来讲：每个自行车是所有**自行车对象**的一个**实例**，所有的自行车，称为一个类（class）,这个类就是那个图纸。

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);
        }
}
```

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.

Here's a BicycleDemo class that creates two separate Bicycle objects and invokes their methods:

```
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();
    }
}
```

The output of this test prints the ending pedal cadence, speed, and gear for the two bicycles:

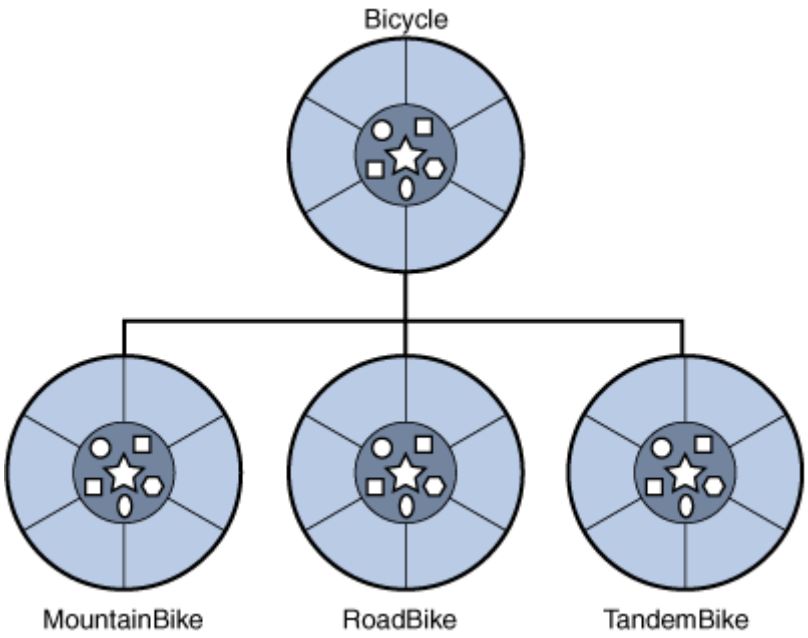
```
cadence:50 speed:10 gear:2

cadence:40 speed:20 gear:3
```

What Is Inheritance?

什么是继承？

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.
不同类型的对象之间往往会有一定的共同点 ，如山地车，公路和双人车，它们共享自行车的特点（当前的速度，踏板节奏，齿轮），同时它们也有不同的功能，双人车有两个座位和两副把手;某些山地车有额外的链条，使其可以有低速的齿轮转速。

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*:
面向对象编程允许类从其他类里面继承公用的状态和动作。在这个例子里面，"Bicycle"变成了"Mountain Bike,RoadBike,TandermBike"的父类。在 Java 编程语言里面，每个 class 允许有一个直接的父类，每个超类可以有无限的子类。



A hierarchy of bicycle classes.

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
}
```

This gives MountainBike all the same fields and methods as Bicycle, yet allows its code to focus exclusively on the features that make it unique. This makes code for your subclasses easy to read. However, you must take care to properly document the state and behavior that each superclass defines, since that code will not appear in the source file of each subclass.

通过集成，可以使 MountainBike 拥有和 Bicycle 一样的字段和方法，也使它的代码集中在不同点上。这样是你的子类代码易读，但是，当那些字段和动作不在子类中出现的时候，你必须要给每个子类的字段和动作加以注释，便于其它人的理解。

What Is an Interface?

什么是接口？

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.
就想你已经学习过的，对象用它们表现出来的方法定义了它们跟外部世界的相互作用。方法来自于对象与外部世界的分界面，你电视机前面的按钮，例如，在你和拖线板之间，你可以按下“按钮”来关闭和打开电视机。

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:
在大部分相同的情况下接口是一组相关的没有内容的方法。比如 bicycle 的动作，提取成接口后就可能是下面的样子：

```
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
}
```

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.
通过实现一个接口，可以让类的动作变得更正式，它可以保证跟提供动作的一致。接口建立一个类和外部之间的约束，并且这个约束在编译器编译的时候是强制的。如果你的类声明要实现一个接口，那么这个接口的所有方法都必须出现在你的类中，这样你的类才会编译成功。

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.

Questions and Exercises: Object-Oriented Programming Concepts

Questions

1 Real-world objects contain _State__ and _behavior__.
2 A software object's state is stored in __fields_.
3 A software object's behavior is exposed through _Methods__.
4 Hiding internal data from the outside world, and accessing it only through publicly exposed methods is known as data _encapsulations__.
5 A blueprint for a software object is called a __Class_.
6 Common behavior can be defined in a _superclass__ and inherited into a _subclass__ using the _extends__ keyword.
7 A collection of methods with no implementation is called an _interfaces__.
8 A namespace that organizes classes and interfaces by functionality is called a package.

9 The term API stands for Application Programming Interface

Exercises

10 Create new classes for each real-world object that you observed at the beginning of this trail. Refer to the Bicycle class if you forget the required syntax.

11 For each new class that you've created above, create an interface that defines its behavior, then require your class to implement it. Omit one or two methods and try compiling. What does the error look like?