Object-Oriented Programming Concepts

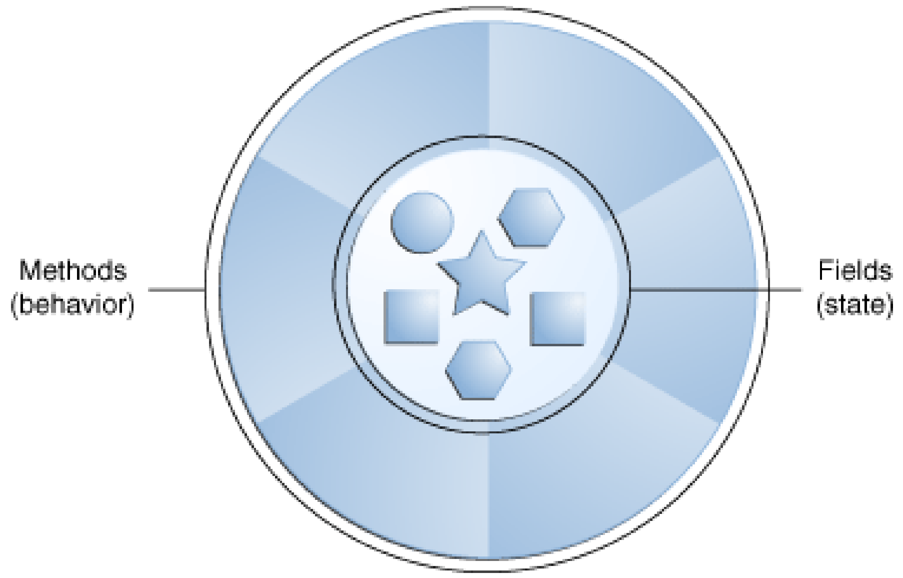
What Is an Object?

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

- Bicycles have state (current gear, current pedal cadence, current speed) and behavior (changing gear, changing pedal cadence, applying brakes).
- Desktop lamp may have only two possible states (on and off) and two possible behaviors (turn on, turn off) .

What Is an Object?

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

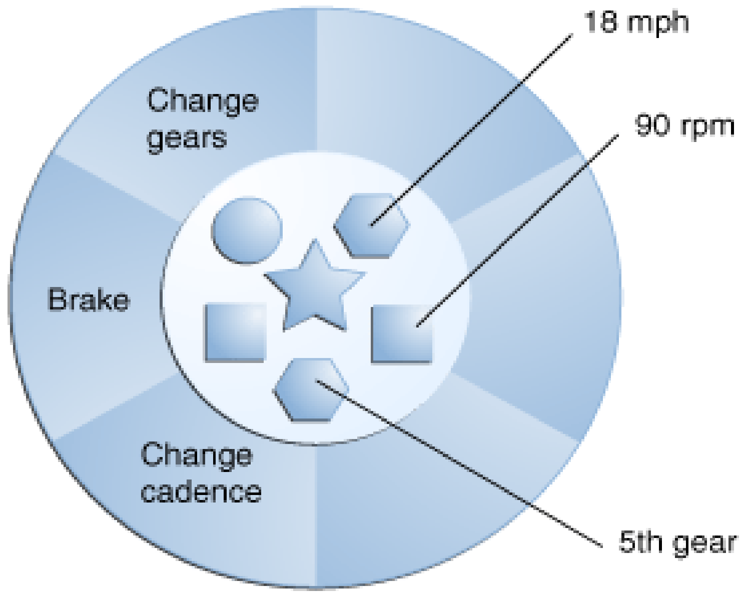


A software object

What Is an Object?

- An object stores its state in *fields* (variables) .
- An object exposes its behavior through *methods* (functions).
- 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.

Example – A Bicycle



State：
current speed
current pedal cadence
current gear

Benefits of Using Objects

- 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.

Benefits of Using Objects

- 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.

What Is a Class?

A class is the template or blueprint from which objects are made.

```
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);
    }
 }
```

BicycleDemo

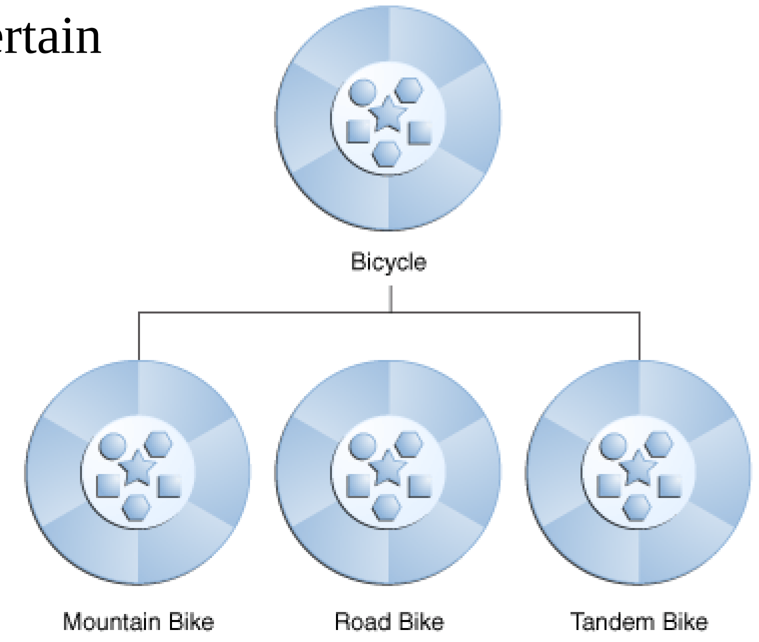
```
class BicycleDemo {
    public static void main(String[] args) {

        // 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();
    }
}
```

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:
  - current speed,
  - current pedal cadence,
  - current gear
- Each also defines additional features that make them different



- 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.

What is Inheritance?

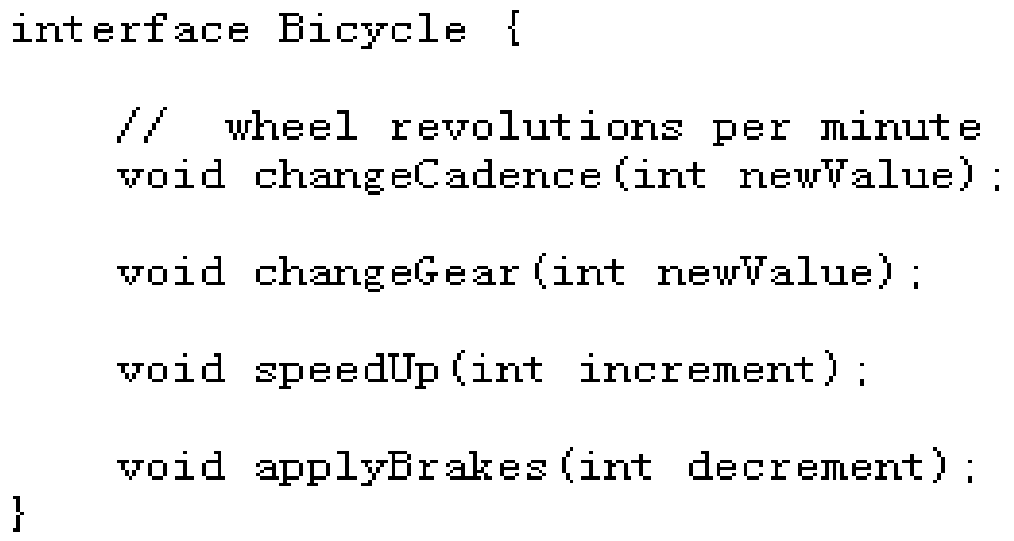
- In the Java programming language, each class is allowed to have one direct superclass, and each superclass has the potential for an unlimited number of *subclasses*.
- Subclass syntax:

```
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.
- 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.

What is an Interface?

- Methods form the object's *interface* with the outside world.
- In its most common form, an interface is a group of related methods with empty bodies.



```
interface Bicycle {

    //  wheel revolutions per minute
    void changeCadence(int newValue);

    void changeGear(int newValue);

    void speedUp(int increment);

    void applyBrakes(int decrement);
}
```

Implement the Interface?

```
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;
    }

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

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.
- 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:
    - 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;
  - The Java Platform API Specification contains the complete listing for all packages, interfaces, classes, fields, and methods supplied by the Java SE platform.