Object-Oriented Programming
Lecture 1: OOP Concepts

Dr. Lê Hồng Phương -- Department of Mathematics, Mechanics and Informatics, VNUH

July 2012
OOP Concepts

✦ Object
✦ Class
✦ Inheritance
✦ Interface
✦ Package
Object

✦ **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.

✦ Objects are key to understand *object-oriented technology*. 
Many 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**. For example:

- Dogs have state (**name**, **color**, **breed**) and behavior (**barking**, **fetching**, **wagging tale**).

- Bicycles also have state (**current gear**, **current pedal cadence**, **current speed**) and behavior (**changing gear**, **changing pedal cadence**, **applying brakes**).
Object

- Identify the state and behavior for real-world objects is a good way to begin thinking in terms of OOP.

**Exercise 1:**

- 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?
  - What possible behaviors can this object perform?

- Write down your observations
Object

- Real-world objects vary in complexity:
  - Your desktop lamp has only two possible states (on, off) and two possible behaviors (turn on, turn off).
  - Your desktop radio might have additional states (on, off, current volume, current station) and behaviors (turn on, turn off, increase volume, decrease volume, seek, scan, tune).
  - Some objects will also contain other objects.
Object

- These real-world observations all translate into the world of OOP.

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

- An object stores its state in **fields** (variables in some programming language) and exposes its behavior through **methods** (function in some programming languages).
Object

✦ Methods operate on an object’s internal state and serve as the primary mechanism for object-oriented communication.

✦ Hiding internal state and requiring all interaction to be performed though an object’s methods is known as data encapsulation.

✦ Data encapsulation is a fundamental principle of OOP.
A bicycle modeled as a software object:

- state (speed, cadence, gear)
- methods for changing that state

The object remains in control of how the outside world is allow to use it.

If the bicycle has only 6 gears, a method to change gear could reject any value that is less than 1 or greater than 6.
Bundling code into individual software objects provides a number of benefits:

1. Modularity
2. Information hiding
3. Code re-use
4. Pluggability and debugging ease
Object

✦ Modularity:
  ✦ The source code for an object can be written and maintained independently of the source code for another objects.

✦ Information hiding:
  ✦ By interacting only with object’s methods, the details of its internal implementation remain hidden from the outside world.
Object

✦ Code re-use:

✦ If an object already exists (written by another software developer) you can use that object in your program.

✦ This allows specialists to implement, test, debug complex task-specific objects.

✦ 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. No need to remove the entire system.
OOP Concepts

- Object
- Class
- Inheritance
- Interface
- Package
Class

✦ In the real-world, many individual objects are of the same kind.

✦ There may be thousands of bicycles in existence, all of the same make and model.

✦ Each bicycle was built from the same set of blueprints and contains the same components.

✦ Your bicycle is an instance of the class of objects known as bicycles.

✦ What is a class?

✦ A class is a blueprint from which individual objects are created.
```java
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 fields represent the object’s state.

The methods define its interaction with the outside world.
The Bicycle class does not contain a main method.

- It’s not a complete application.
- It’s just the blue print 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.

BicycleTester class that creates two separate Bicycle objects and invokes their methods.
class BicycleTester {
    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();
    }
}
OOP Concepts

- Object
- Class
- Inheritance
- Interface
- Package
Inheritance

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

✦ Mountain bikes, road bikes and tandem bikes all share characteristics of bicycles.

✦ Each kind also defines additional features that make them different:

✦ Tandem bicycles have two seats and two sets of handlebars.

✦ Road bikes have drop handlebars.

✦ Mountain bikes have an additional chain ring.
Inheritance

✦ Object-oriented programming allows class to \textit{inherit} commonly used state and behavior from other classes.

✦ Bicycle now becomes the \textit{superclass} of MountainBike, RoadBike and TandemBike.

✦ In the Java programming language, each class is allowed to have one direct superclass, each superclass has the potential for an unlimited number of \textit{subclasses}. 
Inheritance

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

MountainBike has the same fields and methods as Bicycle.
OOP Concepts

- Object
- Class
- Inheritance
- Interface
- Package
Interface

✦ Recall: objects define their interaction with the outside world through their methods.

✦ Methods form the object’s interface with the outside world.

✦ The buttons on the front of your television set 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.
An interface is a group of related methods with empty bodies.

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

Implement an interface:

```
class MyBicycle implements IBicycle {
    // remainder of this class
    // implemented as before
}
```

* Note that MyBicycle must implement all methods defined by the interface IBicycle (as in Bicycle).
OOP Concepts

- Object
- Class
- Inheritance
- Interface
- Package
A package is a namespace that organizes a set of related classes and interfaces.

Packages are similar to different folders on your computer.

The Java platform provides an enormous class library (a set of packages) suitable for use in your own applications.

This library is called Application Programming Interface (API)

Example: String, File, Socket...

http://docs.oracle.com/javase/7/docs/api/index.html
Exercise

✦ **Exercise 2**: Create new classes for each real-world object that you observed.

✦ **Exercise 3**: For each new class that you have 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?