Previous Lecture:
- Array of objects

Today’s Lecture:
- Introduction to inheritance
- extends a class
- Visibility modifier protected
- Accessing parent class: super

Reading (JV):
- Sec 7.0 - 7.2

Object Oriented Programming
- OOP = Encapsulation + Extensibility
- Encapsulation: permit code to be used without knowing implementation details
  ⇒ Organize classes and control access
- Extensibility: permit the behavior of classes to be extended incrementally without involving original class implementor
  ⇒ Code reuse—change behavior of a class without having to rewrite the code of the class

Inheritance
- Allows programmer to derive a class from an existing one
- Existing class is called the parent class, or superclass
- Derived class is called the child class or subclass
- The child class inherits the methods and data defined for the parent class
- Inherited trait can be accessed as though it were locally declared (defined)

Inheritance, cont’d
Inheritance relationships shown in a class diagram, with the arrow pointing to the parent class

Vehicle
  └ Car

An is-a relationship: the child is a more specific version of the parent

Single inheritance: one parent only

Deriving a subclass
Reserved word extends establishes an inheritance relationship:

```java
class Vehicle {
    // class contents relating to Vehicle
}

class Car extends Vehicle {
    // class contents specific to Car
}
```

Different hierarchy for different case

Vehicle
  └ CargoType
      └ CargoPlane
  └ PassengerType
      └ PassPlane

Vehicle
  └ Car
     └ Plane

Car
  └ Plane

Plane
Single inheritance

Vehicle

CargoType

PassengerType

NO!

Single inheritance—one parent only

Extending a class

class Vehicle {
    private id;
    public void move() {
        ...
    }
    public String toString(){
        ...
    }
}

class Car extends Vehicle {
    private plateNum;
    public void drive() {
        ...
    }
    public String toString(){
        ...
    }
}

Object of class Car can access method move as though move is locally defined!

protected visibility

Visibility modifiers control which members get inherited

- private
  - Not inherited, can be accessed by local class only
- public
  - Inherited, can be accessed by all classes
- protected
  - Inherited, can be accessed by subclasses

Access: access as though declared locally
- All variables from a superclass exist in the subclass, but some (private) cannot be accessed directly

public class Vehicle {
    protected int plateNum;  //license plate #
    private int numWheels;   //# of wheels
    public String toString() {
        return "License plate " + plateNum;
    }
}  //class Vehicle

public class Plane extends Vehicle {
    protected double wingSpan;
    private boolean hasPropeller;
    public void writeProperties() {
        System.out.println(plateNum);  //valid?
        System.out.println(numWheels); //valid?
        if (hasPropeller)              //valid?
            System.out.println("Prop plane");
    }
}  //class Plane

Reserved word super

- Invoke constructor of superclass

    super(parameter-list);

- parameter-list must match that in superclass' constructor

- Access methods and variables from superclass

plateNum  0
numWheels  0
toString()

plateNum  0
numWheels  0
w wingSpan  0
hasPropeller  false
writeProperties()
}  //class Plane

Object of class Car can access method move as though move is locally defined!
// New definition of class Vehicle
public class Vehicle {
    protected int plateNum;  //plate #
    private int numWheels;  // # of wheels

    public Vehicle(int plate, int wheels) {
        plateNum = plate;
        numWheels = wheels;
    }

    public String toString() {
        return "License plate " + plateNum;
    }
}

// New definition of class Plane
public class Plane extends Vehicle {
    protected double wingSpan;
    private boolean hasPropeller;

    public Plane(int plate, int wheels, double span, boolean prop) {
        // must call superclass' constructor
        super(plate, wheels);
        wingSpan = span;
        hasPropeller = prop;
    }

    public void writeProperties() {
        System.out.println(plateNum);
        //System.out.println(numWheels);  // invalid
        if (hasPropeller) System.out.println("Prop plane");
    }
}

Important ideas in inheritance

- Use different hierarchies for different problems
- Single inheritance
- Keep common features as high in the hierarchy as reasonably possible
- Inherited features are continually passed down the line
- Use the superclass’ features as much as possible
- “Inherited” ⇒ “can be accessed as though declared locally”
  (private variables in superclass exists in subclasses; they just cannot be accessed directly)