Reserved word super

• Invoke constructor of superclass
  
  ```java
  super(parameter-list);
  ```

  • parameter-list must match that in superclass’ constructor

• Access methods and variables from superclass

Previous Lecture:
• Introduction to inheritance
• `extends` a class
• Visibility modifier `protected`
• Using superclass’ constructor

Today’s Lecture:
• Accessing members in parent class: `super`
• `Overriding` methods
• Polymorphism

Reading (JV):
• Sec 7.1 – 7.3

```java
// 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;
    }

    private void voo() {
        System.out.println(“Boo”);
    }
}
```
Overriding methods

- Subclass can override definition of inherited method in favor of its own
- New method in subclass must have same signature as superclass (but has different method body)
- Which method gets used? 
  The object that is used to invoke a method determines which version is used
- Method declared to be final cannot be overridden
- Do not confuse overriding with overloading!

```java
// Better 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) {
        super(plate, wheels);
        wingSpan = span;
        hasPropeller = prop;
    }
    // Override toString
    public String toString() {
        // use method from superclass
        String s = super.toString();
        if (hasPropeller)
            s += "Prop plane";
        return s;
    }
    // Some other method
    public void poo() { System.out.println(); }
}
```

```java
public class Client {
    public static void main(String[] args) {
        Vehicle v1 = new Vehicle(7443, 4);
        Plane p1 = new Plane(333, 6, 35, true);
        System.out.println(v1); //?
        v1.voo(); //?
        v1.poo(); //?
        System.out.println(p1); //?
        p1.voo(); //?
        p1.poo(); //?
    }
}
```

A special Room

- Consider class Room:
  - Each Room has an ID
  - Each Room has a messiness level
  - A Room's messiness level decreases after cleaning
  - A Room's condition can be described
- Write a class Bathroom that has all the functionalities of a Room and ...
  - A Bathroom may or may not have a shower
  - A Bathroom may need some major cleanup

```java
// Better 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) {
        super(plate, wheels);
        wingSpan = span;
        hasPropeller = prop;
    }
    // Override toString
    public String toString() {
        // use method from superclass
        String s = super.toString();
        if (hasPropeller)
            s += "Prop plane";
        return s;
    }
    // Some other method
    public void poo() { System.out.println(); }
}
```

```java
public class Client {
    public static void main(String[] args) {
        Vehicle v1 = new Vehicle(7443, 4);
        Plane p1 = new Plane(333, 6, 35, true);
        System.out.println(v1); //?
        v1.voo(); //?
        v1.poo(); //?
        System.out.println(p1); //?
        p1.voo(); //?
        p1.poo(); //?
    }
}
```
class Room {
    private static int nextID = 1;
    // id of next room to be created

    protected int id;
    private int mess; // mess level

    public Room(int mess) {
        this.mess = mess;
        id = nextID; nextID++;
    }

    public String toString() {
        return "Room " + id;
    }

    public void clean() {
        mess--;
        if (mess<0) mess=0;
    }

    public void report() {
        System.out.println(toString()+
        ", has mess level +"+mess);
    }
}

Polymorphism

- "Have many forms"
- A **polymorphic** reference can refer to different objects (related through inheritance) at different times

- Example:
  
  Vehicle mover; // a Vehicle reference
  Plane flyer;   // a Plane reference
  mover = new Vehicle(...);
  flyer = new Plane(...);
  // A plane is a vehicle
  mover = new Plane(...);
  mover = flyer;
  // A vehicle is not a plane
  flyer = new Vehicle(...); // invalid