Topics: Polymorphism, static variables and methods, method invocation

Reading (JV): Sec 7.3, 7.4

Polymorphism

A polymorphic reference can refer to different objects (related through inheritance) at different times.

Vehicle mover; //a Vehicle reference
Plane flyer; //a Plane reference
mover = new Vehicle(...);
flyer = new Plane(...);
// A plane is a vehicle
mover = new Plane(...); // valid statement
mover = flyer; // valid statement
// A vehicle is not a plane
flyer = new Vehicle(...); // invalid statement

Accessing methods and fields through polymorphic references

Ask two questions:
1. What determines whether a method can be accessed?
2. What determines which version of an overridden method gets invoked?

1. The type of the reference determines the methods and fields that can be accessed

   ```java
class V {
      int num1;
      void vmethod() { num1++; }
   }

class W extends V {
      int num2;
      void wmethod() { num2++; }
   }
```

   **Client code:**
   ```java
   V x = new W();
   System.out.println(x.num1); //?
   System.out.println(x.num2); //?
   x.vmethod(); //?
   x.wmethod(); //?
   ```

2. The type of the object determines which version of the overridden method gets invoked. Class Vehicle has method toString which class Plane overrides:

   ```java
   Vehicle v1 = new Vehicle(...);
   Vehicle v2 = new Plane(...);
   System.out.println(v1); //the Vehicle’s version
   System.out.println(v2); //the Plane’s version
   ```

static methods and variables

- Same rules for inheritance (accessibility) with respect to visibility modifiers
- Method: implicitly `final`
- Variable: same memory space as super class

Example: House.java

Change the methods or add your own and try to call them. If something doesn’t work, make sure you understand why. The error messages from the compiler usually explains what the errors are.
class Room {
    private static int nextID = 1;
    //id of next room to be created
    protected int id;
    private int mess; //messiness index
    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 messiness index "+mess);
    }
    public static void countRooms() {
        System.out.println((nextID-1)+
        " rooms in total");
    }
} //class Room

class Bathroom extends Room {
    private boolean hasShower;
    public Bathroom(int mess,
    boolean hasShower) {
        super(mess);
        this.hasShower = hasShower;
    }
    public String toString() {
        String line = super.toString();
        line += ", a bathroom";
        if (hasShower)
            line += " with shower";
        return line;
    }
    public void majorCleanUp() {
        clean(); clean(); clean(); clean();
    }
} //class Bathroom

class House {
    public static void main(
        String[]args) {
        Room r1 = new Room(5);
        Bathroom r2 = new
        Bathroom(10,true);
        // Method invocation
        // Access non-inherited fields
        System.out.println(r1);
        System.out.println(r2);
        r1.report();
        r2.report();
        r1.clean(); r1.report();
        r2.clean(); r2.report();
        r2.majorCleanUp();
        r2.report();
        //r1.majorCleanUp();
        System.out.println();
        // Polymorphism
        Room r3 = new
        Bathroom(20,false);
        System.out.println(r3);
        r3.clean(); r3.report();
        //r3.majorCleanUp();
        ((Bathroom)r3).majorCleanUp();
        r3.report();
        System.out.println();
        // Static methods and variables
        Room.countRooms();
        Bathroom.countRooms();
    }
} //class House