

CSE 8B—Intro to CS: Java

Winter, 2006

January 19 (Day 4)
Inheritance

Instructor: Neil Rhodes

Inheritance

Hierarchy

Subclasses inherit from their superclasses

inherited: a new access permission

Overriding methods

Final methods/final classes

Changing the return type of an overridden method

Testing membership in a class

All classes inherit from Object

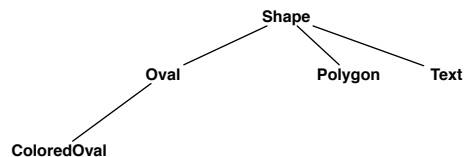
The equals method

Using super and this in constructors

2

Hierarchy

Classes form a hierarchy



3

Subclasses inherit from their superclasses

Declaring a subclass:

```
public class Shape {  
    private Color color;  
    public void setColor(c) { ... };  
    public void color() { return color; }  
    ...  
}  
public class Polygon extends Shape {  
    private int[] points;  
    public void draw();  
}
```

Subclasses inherit:

- Data
- Methods

Private data and methods can't be directly accessed from subclass methods

4

The is-a relationship

A subclass is-a superclass

- A polygon is-a shape
- An hourly employee is-a employee
- A car is-a vehicle

For instance variables, we use the has-a relationship

- A car has-a wheel

Any operations you can do to a superclass, you can do to the subclass as well.

- The operations may work differently, but they're still meaningful

5

inherited: a new access permission

Access permissions

- **private**
 - can only be accessed by methods in this class
- **public**
 - can be accessed by methods from any class
- **protected**
 - Can only be accessed by methods in this class, or a subclass
- **package** (no access permission explicitly specified)
 - Can only be accessed by methods in this package.

6

Overriding methods

Inherit:

- State
- Behavior

Add new:

- State
- Behavior

Modify:

- Behavior
- Override the method by redeclaring it

Overridden methods:

- May want to completely replace the old method
- May want to augment the old method's behavior

7

Overriding methods

```
public class Shape {  
    ...  
    public void Draw() {  
        // do nothing. A raw shape  
        doesn't know how to draw  
    }  
}
```

```
public class Oval extends Shape {  
    ...  
    public void Draw() {  
        do oval drawing stuff  
    }  
}
```

```
public class ColoredOval extends  
Oval {  
    public void setColor() { ...}  
    public void Draw() {  
        setColor();  
        super.Draw();  
    }  
}
```

8

Final methods/Final classes

The `final` keyword modifying a:

- method
 - specifies that the method cannot be overridden
- class
 - Specifies that the class cannot be subclassed
- variable
 - Specifies that the variable can't be changed

9

Changing the return type of an inherited method

You can make the return type more restrictive

New in Java 1.5
Covariant return type

- Subclass of the class it originally returned
- Any code written for the original class will work with this new class

```
public class ShapeList {  
    public void addShape(Shape s){ ...}  
    public Shape nthShape(int n) {  
        ...  
    }  
}
```

```
ShapeList sl;  
...  
sl.addShape(new Shape());  
...  
Shape s = sl.nthShape(2);
```

```
public class OvalList {  
    public void addShape(Shape s){ ...}  
    public Oval nthShape(int n) {  
        ...  
    }  
}
```

```
OvalList ol;  
...  
ol.addShape(new Oval());  
...  
Oval s = ol.nthShape(2);
```

10

Testing membership in a class

You can test using the `instanceof` operator:

- `object instanceof Class`
 - Returns true if `object` is an instance of `Class` (or some subclass)

You can also find the class of an object:

- `object.getClass()`

11

All classes inherit from Object

Either

- directly (no extends)
- indirectly

Object defines several methods

- `String toString()`
 - By default, returns ""
- `Boolean equals(Object obj)`
 - By default, only if `obj == this`
- `Class getClass()`
- `int hashCode()`

12

The equals method

Must be an equivalence relation

- reflexive
 - `a.equals(a)` must be true
- transitive
 - if `a.equals(b)` and `b.equals(c)`, then `a.equals(c)`
- symmetric
 - if `a.equals(b)` then `b.equals(a)`
- doesn't equal null
 - except that `null.equals(null)` is true

The following *won't* work

```
public class Oval {
    private int width, height;

    public boolean equals(Object o) {
        Shape s = (Shape) o;
        return s.width == width && s.height == height;
    }
}
```

13

The equals method

The following *won't* work

```
public class Oval {
    private int width, height;

    public boolean equals(Object o) {
        if (o == null)
            return false;
        else if (!o instanceof Oval))
            return false;
        else {
            Shape s = (Shape) o;
            return s.width == width && s.height == height;
        }
    }
}
```

14

The equals method

The following *will* work

```
public class Oval {
    private int width, height;

    public boolean equals(Object o) {
        if (o == null)
            return false;
        else if (getClass() != o.getClass())
            return false;
        else {
            Shape s = (Shape) o;
            return s.width == width && s.height == height;
        }
    }
}
```

15

Using super and this in constructors

When you subclass a class, the constructor for the subclass must make sure *all* instance variables are initialized.

- If you don't specify otherwise, the default constructor will be called automatically as first line of the constructor.
- You can specify explicitly:

```
- public Oval(int w, int h) {
    super();
}

- public ColoredOval(int w, int h, Color c) {
    super(w, h);
    setColor(c);
}
```

- The call to the `super()` constructor must be the first line of the constructor
- You can also call another constructor in the same class with `this`:

```
- public Oval() {
    this(DEFAULT_WIDTH, DEFAULT_HEIGHT);
}
```

16