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

Classes form a hierarchy

Declaring a subclass:

```java
public class Shape {
    private Color color;
    public void setColor(c) { … }
    public void color() { return color; }
    …
 }
public void 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
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

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.

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

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();
    }
}
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

### Changing the return type of an inherited method
You can make the return type more restrictive:
- Subclass of the class it originally returned
- Any code written for the original class will work with this new class

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

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

### 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()`
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

```java
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;
    }
  }
}
```

The equals method

The following will work

```java
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;
    }
  }
}
```

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);
    }