

- Previous Lecture:
 - Array of objects
- Today's Lecture:
 - Inheritance—**extending** a class
- Reading:
 - Sec 7.1

Nov 15, 2005 Lecture 23 2

Separate classes—each has its own members

<pre>class Dice { private int top; private int sides; public Dice(...) {...} public void roll() {...} public String toString() {...} public int getTop() {...} public int getSides() {...} }</pre>	<pre>class TrickDice { private int top; private int sides; private int weightedSide; private int weight; public TrickDice(...) {...} public void roll() {...} public String toString() {...} public int getTop() {...} public int getSides() {...} public int getWeight() {...} }</pre>
---	--

Nov 15, 2005 Lecture 23 5

Can we get all the functionality of Dice in TrickDice without re-writing all the Dice components in class TrickDice?

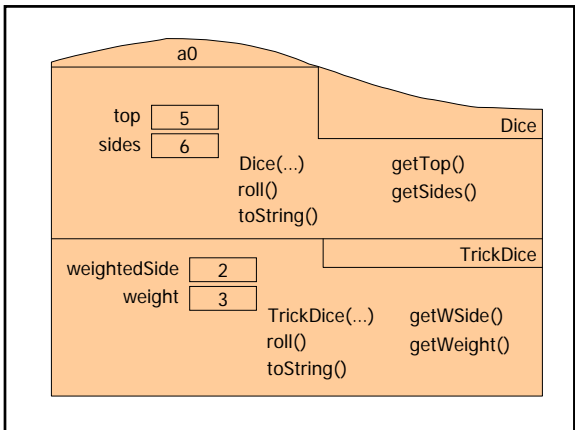
<pre>class Dice { private int top; private int sides; public Dice(...) {...} public void roll() {...} public String toString() {...} public int getTop() {...} public int getSides() {...} }</pre>	<pre>class TrickDice { //everything in class Dice //plus new/modified stuff //below private int weightedSide; private int weight; public TrickDice(...) {...} public void roll() {...} public String toString() {...} public int getWSide() {...} public int getWeight() {...} }</pre>
---	--

Nov 15, 2005 Lecture 23 6

Yes! Make TrickDice a **subclass** of Dice.

<pre>class Dice { private int top; private int sides; public Dice(...) {...} public void roll() {...} public String toString() {...} public int getTop() {...} public int getSides() {...} }</pre>	<pre>class TrickDice extends Dice { private int weightedSide; private int weight; public TrickDice(...) {...} public void roll() {...} public String toString() {...} public int getWSide() {...} public int getWeight() {...} }</pre>
---	---

Nov 15, 2005 Lecture 23 7



Inheritance

Inheritance relationships are shown in a *class diagram*, with the arrow pointing to the parent class

```

classDiagram
    class Dice
    class TrickDice
    Dice <|-- TrickDice
    
```

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

Single inheritance: one parent only

Nov 15, 2005 Lecture 23 9

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 (public) members defined for the parent class
- Inherited trait can be *accessed as though it was locally declared (defined)*

Nov 15, 2005

Lecture 23

12

Reserved word **super**

Invoke constructor of superclass

```
super(parameter-list);
```

parameter-list must match that in superclass' constructor

Nov 15, 2005

Lecture 23

14

Which components get inherited?

- public** components get inherited
- private** components exist in object of child class, but cannot be **directly** accessed in child class \Rightarrow we say they are **not inherited**
- Note the difference between inheritance and existence!

Nov 15, 2005

Lecture 23

16

protected visibility (see Sec 7.2 for detail)

- 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 the **private** ones cannot be *accessed* directly

Nov 15, 2005

Lecture 23

17

Overriding methods

- Subclass can *override* definition of inherited method
- 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*!

Nov 15, 2005

Lecture 23

18

Important ideas in inheritance

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

Nov 15, 2005

Lecture 23

20

```

/** A Dice (or Die) */
class Dice {

    private int top;    // top face
    private int sides; // number of sides

    /** A Dice has numSides sides and the top face is random */
    public Dice(int numSides) {
        sides= numSides;
        roll();
    }

    /** top gets a random value in 1..sides */
    public void roll() {
        setTop(randInt(1,getSides())) ;
    }

    /** = random int in low..high */
    public static int randInt(int low, int high) {
        return (int) (Math.random()*(high-low+1))+low;
    }

    /** = Get top face */
    public int getTop() { return top; }

    /** = Get number of sides */
    public int getSides() { return sides; }

    /** Set top to faceValue */
    public void setTop(int faceValue) { top= faceValue; }

    /** = String description of this Dice */
    public String toString() {
        return  getSides() + "-sided dice shows face " + getTop();
    }
} //class Dice

```

```

/** A TrickDice has one weightedSide such that the
 * weightedSide appears weight times as often as other sides
 */
class TrickDice extends Dice {

    private int weightedSide; //Weighted side appears more often
    private int weight;      //Weighted side appears weight times as often as other sides

    /** TrickDice has side s appearing with weight w */
    public TrickDice(int numFaces, int s, int w) {
        super(numFaces);
        weightedSide= s;
        weight= w;
    }

    /** = Get weighted side */
    public int getWSide() { return weightedSide; }

    /** = Get weight of weighted side */
    public int getWeight() { return weight; }

    /** top gets random value in 1..sides given trick property */
    public void roll() {
        int r= randInt(1,(getSides()+weight-1));
        if (r>getSides())
            setTop(weightedSide);
        else
            setTop(r);
    }

    /** = String description of this TrickDice */
    public String toString() { return "Tricky " + super.toString(); }
} //class TrickDice

```