Previous Lecture:
- Array of objects

Today's Lecture:
- Inheritance

Reading:
- Sec 8.1, 8.2

A fair die is...

class Dice {
    private int top;
    private int sides;
    public Dice(…) {…}
    public void roll() {…}
    public String toString(){…}
    public int getTop() {…}
    public int getSides() {…}
}

What about a trick die?

class TrickDice {
    private int weightedSide;
    private int weight;
    public TrickDice(…) {…}
    public void roll() {…}
    public String toString(){…}
    public int getWSide() {…}
    public int getWeight() {…}
}

Separate classes—each has its own members

class Dice {
    private int top;
    private int sides;
    public Dice(…) {…}
    public void roll() {…}
    public String toString(){…}
    public int getTop() {…}
    public int getSides() {…}
}

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 getWSide() {…}
    public int getWeight() {…}
}

Separate classes—each has its own members

class Dice {
    private int top;
    private int sides;
    public Dice(…) {…}
    public void roll() {…}
    public String toString(){…}
    public int getTop() {…}
    public int getSides() {…}
}

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() {…}
}

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

class Dice {
    private int top;
    private int sides;
    public Dice(…) {…}
    public void roll() {…}
    public String toString(){…}
    public int getTop() {…}
    public int getSides() {…}
}

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 getWSide() {…}
    public int getWeight() {…}
}

Yes! Make TrickDice a subclass of Dice.

class Dice {
    private int top;
    private int sides;
    public Dice(…) {…}
    public void roll() {…}
    public String toString(){…}
    public int getTop() {…}
    public int getSides() {…}
}

class TrickDice extends Dice {
    private int weightedSide;
    private int weight;
    public TrickDice(…) {…}
    public void roll() {…}
    public String toString(){…}
    public int getTop() {…}
    public int getSides() {…}
    public int getWeight() {…}
}
Inheritance relationships are shown in a class diagram, with the arrow pointing to the class that is the parent.

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

Single inheritance: one parent only.

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).

Reserved word **super**

Invoke constructor of superclass

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

**parameter-list** must match that in superclass’ constructor.

**TrickDice** is a subclass of **Dice**

```
class Dice {
    private int top;
    private int sides;
    public Dice(…) {…}
    public void roll() {…}
    public String toString() {…}
    public int getTop() {…}
    public int getSides() {…}
}
```

```
class TrickDice extends Dice {
    private int weightedSide;
    private int weight;
    public TrickDice(…) {…}
    public void roll() {…}
    public String toString() {…}
    public int getWSide() {…}
    public int getWeight() {…}
}
```

April 21, 2005 Lecture 24
Which components get inherited?

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

**protected visibility** (see Sec 4.4)

- 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

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**

Accessing members in superclass

- **super**
- From constructor in subclass, call superclass' constructor
- Access superclass' version of a overridden method. E.g.:
  
  ```java
  super.toString()
  ```

**static** methods & variables

- **Do not re-declare static components!**
- Same rules for inheritance (accessibility) with respect to visibility modifiers
- **Static method**: implicitly **final**
- **Static variable**: same memory space as superclass

**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" ⇒ "can be accessed as though declared locally"
  
  (**private** variables in superclass **exist** in subclasses; they just cannot be accessed directly)
- Inherited features are continually passed down the line
- Use different hierarchies for different problems