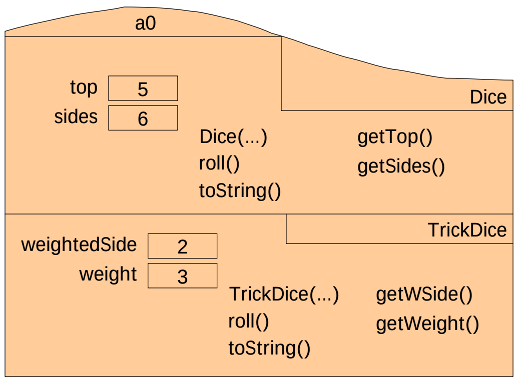
- Previous Lecture:
  - Array of objects
  - Inheritance—extending a class
- Today's Lecture:
  - Constructor in the subclass
  - Overriding methods
  - Using super to access members from the superclass
- Reading:
  - Sec 11.3, 11.6, 11.7

Make TrickDice a subclass of Dice.

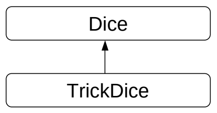
```
class Dice {

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



Inheritance

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



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

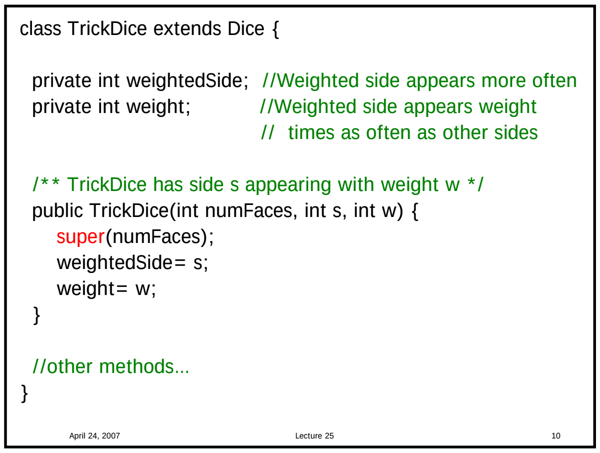
Single inheritance: one parent only

Inheritance

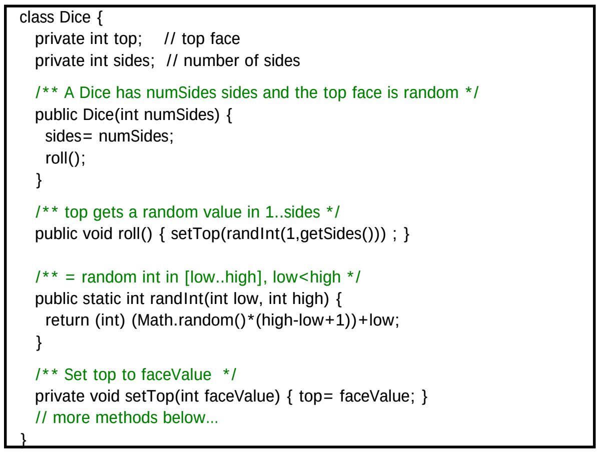
- Allows programmer to *derive* a class from an existing one
- Existing class is called the *parent class,* or *superclass*
- Derived class is called the *child class* or *subclass*
- The child class *inherits* the (public) members defined for the parent class
- Inherited trait can be *accessed as though it was **locally** declared (defined)*

Calling one constructor from another

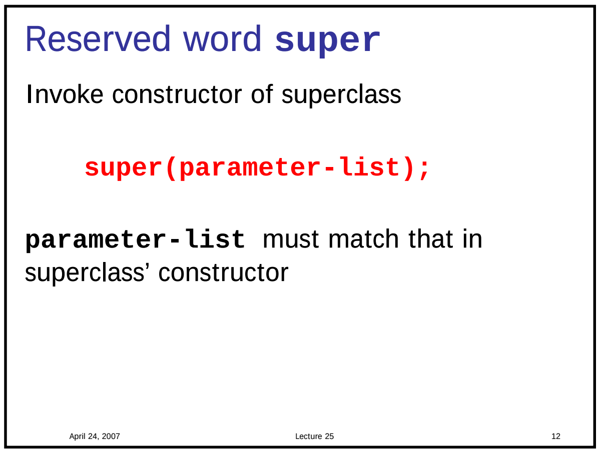
- In a subclass' constructor, call the superclass' constructor with the keyword super instead of the superclass' (constructor's) name
- Always make a call to the superclass' constructor as the 1st statement in a constructor in a subclass!



```
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;
  }

  //other methods...
}
```



```
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], low<high */
   public static int randInt(int low, int high) {
     return (int) (Math.random()*(high-low+1))+low;
   }
   /** Set top to faceValue  */
   private void setTop(int faceValue) { top= faceValue; }
   // more methods below...
}
```

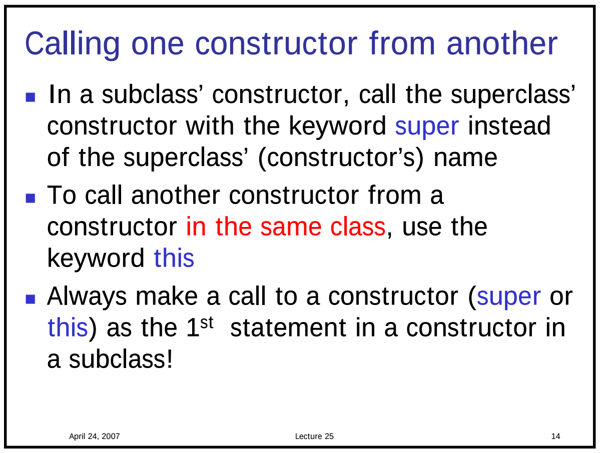


Reserved word **super**

Invoke constructor of superclass

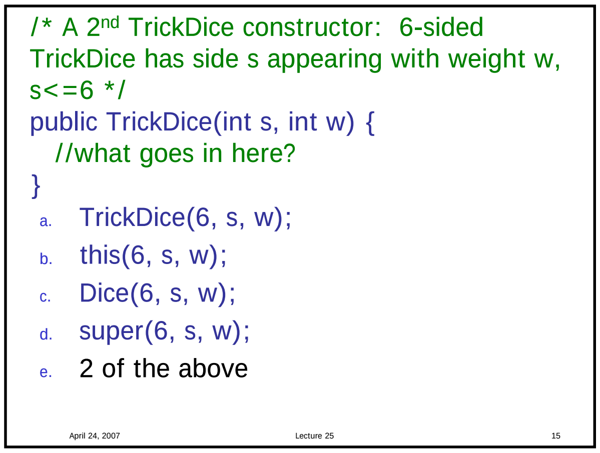
`super(parameter-list);`

`parameter-list` must match that in superclass' constructor



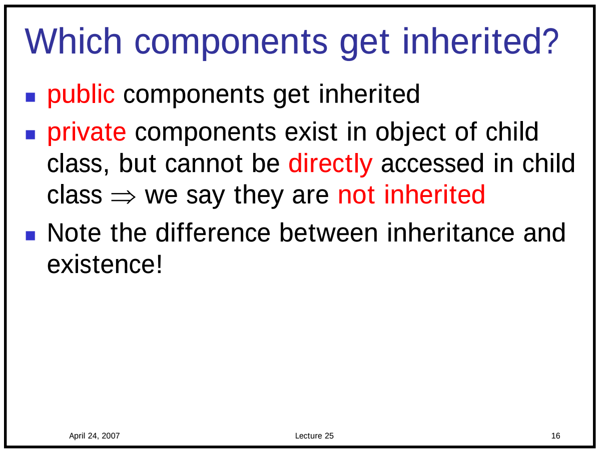
Calling one constructor from another

- In a subclass' constructor, call the superclass' constructor with the keyword super instead of the superclass' (constructor's) name
- To call another constructor from a constructor in the same class, use the keyword this
- Always make a call to a constructor (super or this) as the 1st statement in a constructor in a subclass!



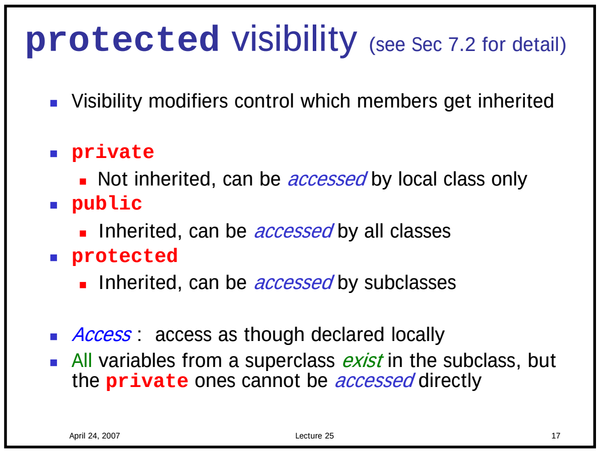
```
/* A 2nd TrickDice constructor:  6-sided
TrickDice has side s appearing with weight w,
s<=6 */
public TrickDice(int s, int w) {
   //what goes in here?
}
```

a. TrickDice(6, s, w);
b. this(6, s, w);
c. Dice(6, s, w);
d. super(6, s, w);
e. 2 of the above



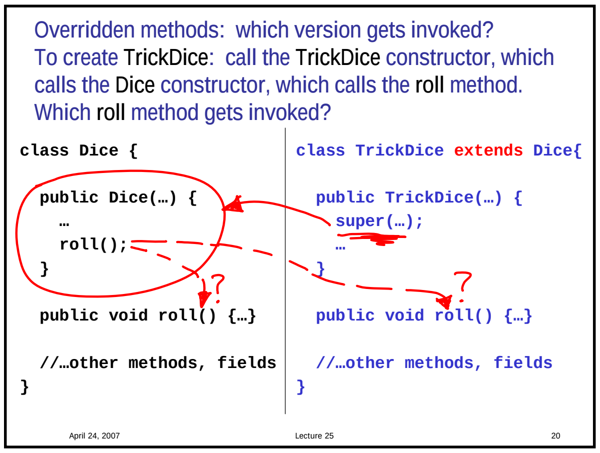
Which components get inherited?

- 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!



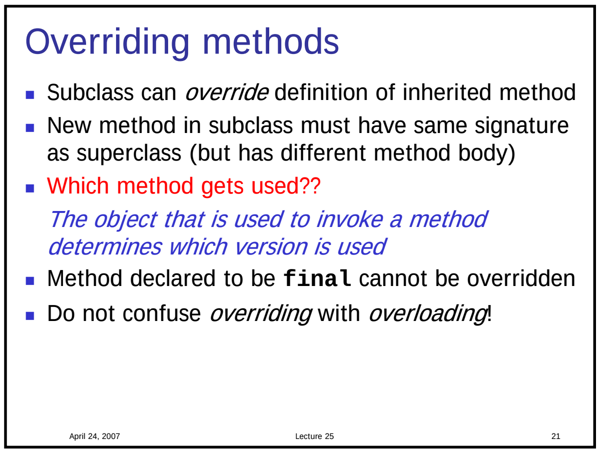
`protected` visibility (see Sec 7.2 for detail)

- 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



Overridden methods: which version gets invoked?
To create TrickDice: call the TrickDice constructor, which calls the Dice constructor, which calls the roll method. Which roll method gets invoked?

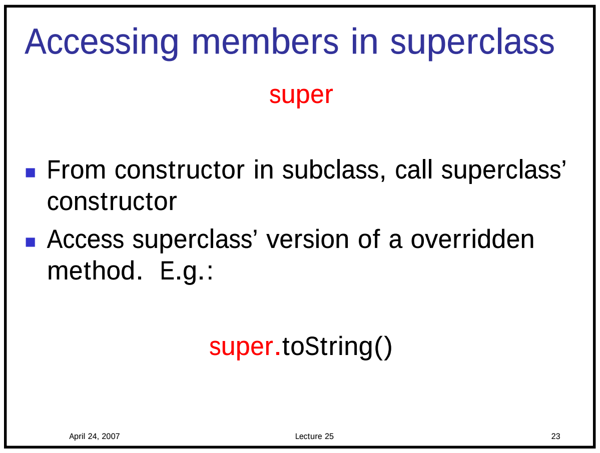
```
class Dice {                          class TrickDice extends Dice{

  public Dice(…) {                      public TrickDice(…) {
    …                                     super(…);
    roll();                               …
  }                                     }

  public void roll() {…}                public void roll() {…}

  //…other methods, fields              //…other methods, fields
}                                     }
```



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*!

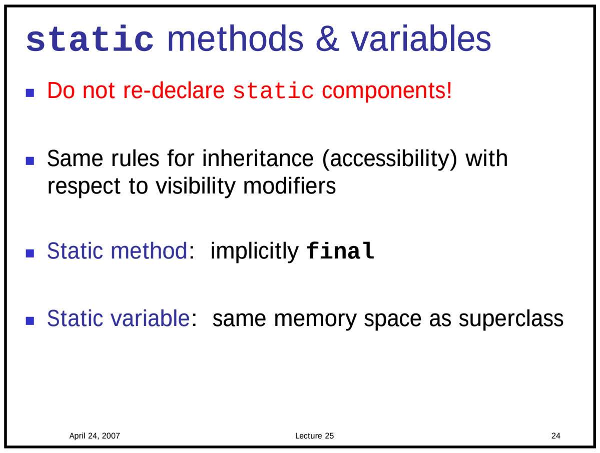


Accessing members in superclass

super

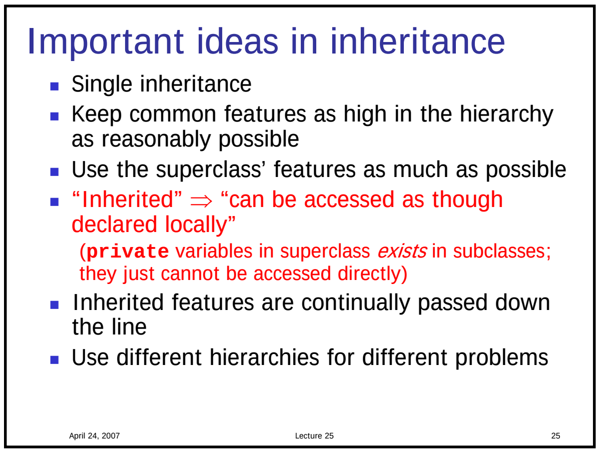
- From constructor in subclass, call superclass' constructor
- Access superclass' version of a overridden method. E.g.:

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 *exists* in subclasses; they just cannot be accessed directly)
- Inherited features are continually passed down the line
- Use different hierarchies for different problems

```java
/** 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], low<high */
   public static int randInt(int low, int high) {
     return (int) (Math.random()*(high-low+1))+low;
   }

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

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

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

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

```java
/** 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
```