Chapter 7

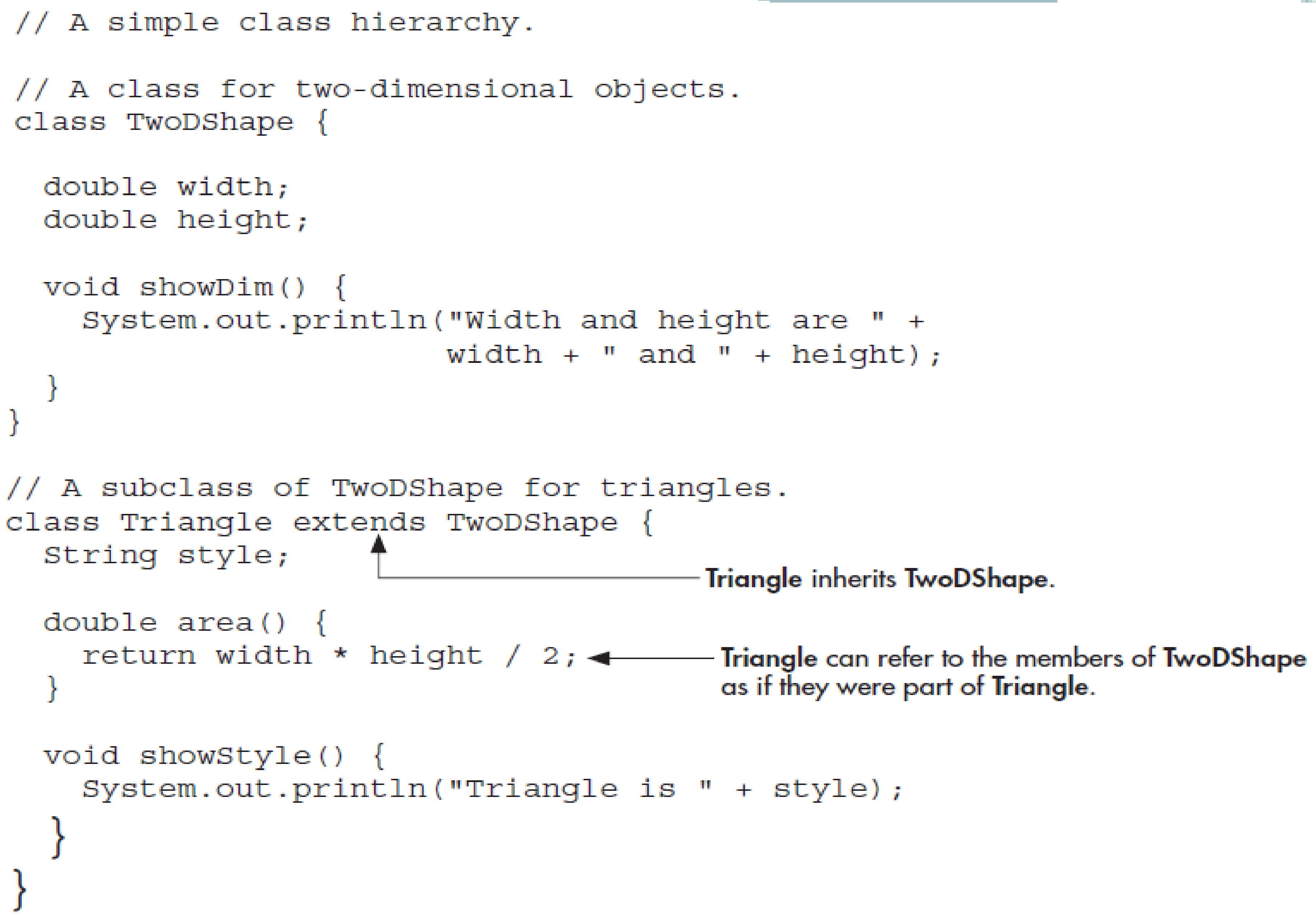
Inheritance

Inheritance

- **Inheritance** allows the creation of hierarchical classifications.
- Using inheritance, you can create a **general class** that defines traits common to a set of related items. This class can then be inherited by other, **more specific classes**, each adding those things that are unique to it.
- In the language of Java, a class that is inherited is called a ***superclass***. The class that does the inheriting is called a ***subclass***.

- A subclass is a specialized version of a superclass. It inherits all the variables and methods defined by the superclass and adds its own, unique elements.
- A class (a "subclass") can inherit all the methods and variables of another class (a "superclass").
- Use the keyword **extends**.
- General form:

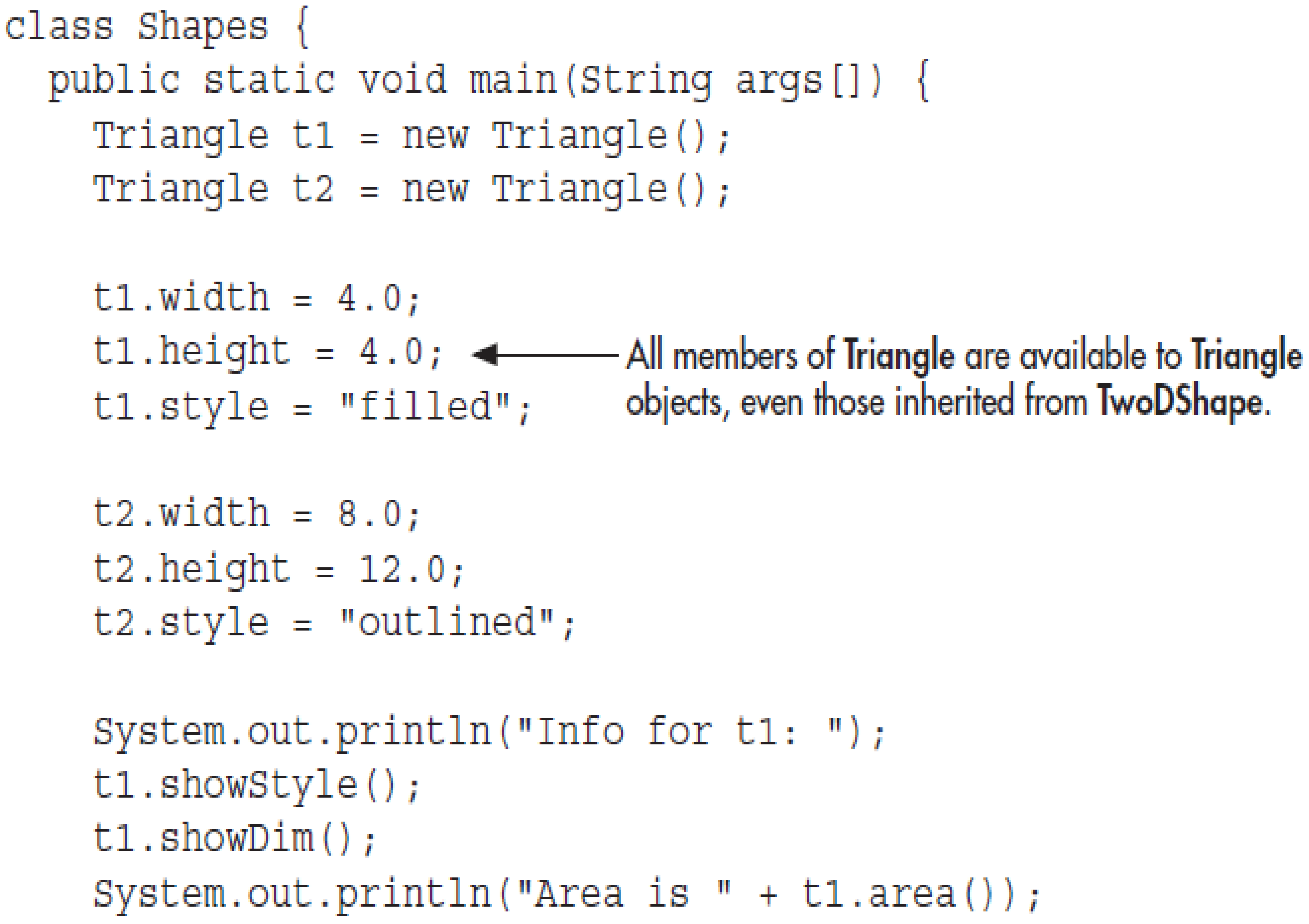
  class *subclass* extends *superclass* { … }
- The subclass extends the superclass by adding behavior and data to the behavior and data provided by the superclass.



```
// A simple class hierarchy.

// A class for two-dimensional objects.
class TwoDShape {

  double width;
  double height;

  void showDim() {
    System.out.println("Width and height are " +
                       width + " and " + height);
  }
}

// A subclass of TwoDShape for triangles.
class Triangle extends TwoDShape {
  String style;

  double area() {
    return width * height / 2;
  }

  void showStyle() {
    System.out.println("Triangle is " + style);
  }
}
```

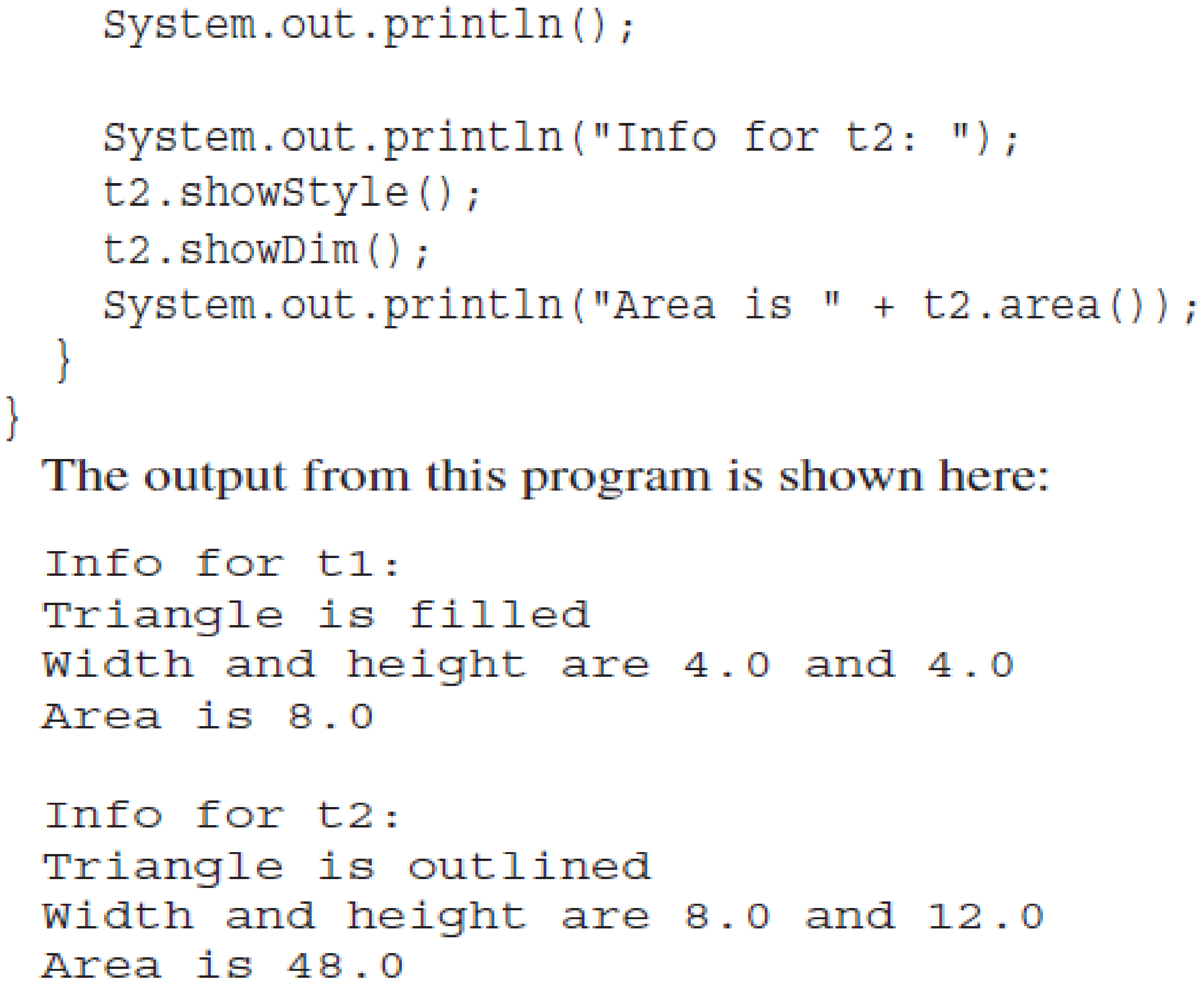
Triangle inherits **TwoDShape**.

Triangle can refer to the members of **TwoDShape** as if they were part of **Triangle**.



```
class Shapes {
  public static void main(String args[]) {
    Triangle t1 = new Triangle();
    Triangle t2 = new Triangle();

    t1.width = 4.0;
    t1.height = 4.0;
    t1.style = "filled";

    t2.width = 8.0;
    t2.height = 12.0;
    t2.style = "outlined";

    System.out.println("Info for t1: ");
    t1.showStyle();
    t1.showDim();
    System.out.println("Area is " + t1.area());
```

All members of **Triangle** are available to **Triangle** objects, even those inherited from **TwoDShape**.



```
    System.out.println();

    System.out.println("Info for t2: ");
    t2.showStyle();
    t2.showDim();
    System.out.println("Area is " + t2.area());
  }
}
```

The output from this program is shown here:

```
Info for t1:
Triangle is filled
Width and height are 4.0 and 4.0
Area is 8.0

Info for t2:
Triangle is outlined
Width and height are 8.0 and 12.0
Area is 48.0
```

Even though **TwoDShape** is a superclass for **Triangle**, it is also a completely independent, stand-alone class. Being a superclass for a subclass does not mean that the superclass cannot be used by itself. For example, the following is perfectly valid:

```
TwoDShape shape = new TwoDShape();

shape.width = 10;
shape.height = 20;

shape.showDim();
```

Of course, an object of **TwoDShape** has no knowledge of or access to any subclasses of **TwoDShape**.

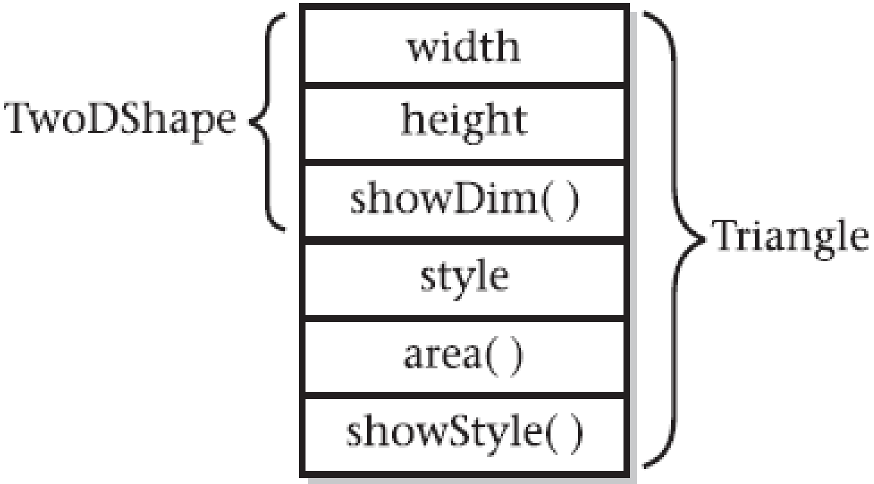


Figure 7-1 A conceptual depiction of the **Triangle** class

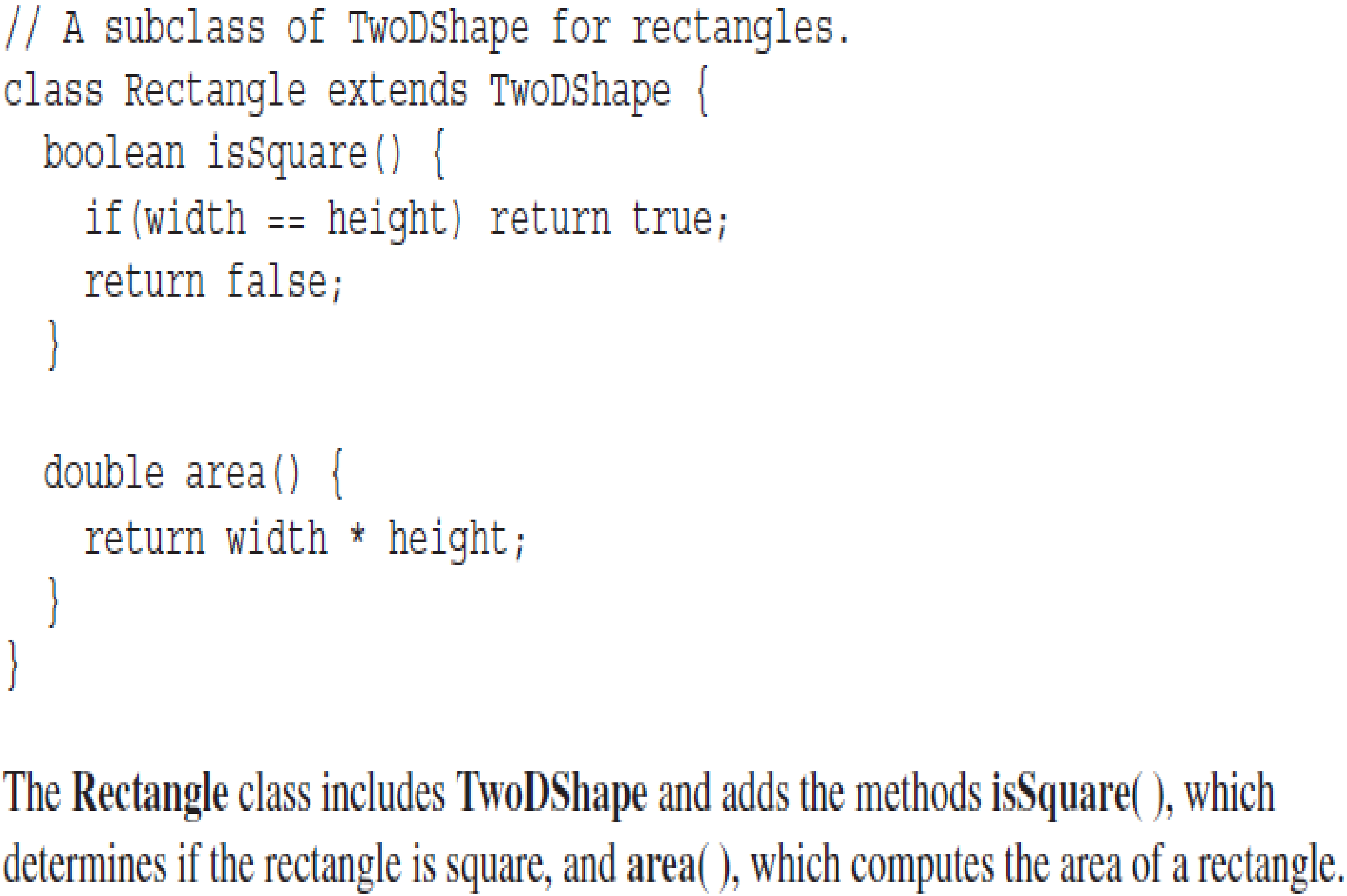
Another Example

```
class OneDimPoint {
  int x = 3;
  int getX() { return x; }
}
class TwoDimPoint extends OneDimPoint {
  int y = 4;
  int getY() { return y; }
}
class TestInherit {
  public static void main(String[] args) {
    TwoDimPoint pt = new twoDimPoint();
    System.out.println(pt.getX() + "," + pt.getY());
  }
}
```

Properties of Inheritance

- A subclass cannot access the private members of its superclass.
- Each class can have at most one superclass, but each superclass can have many subclasses.
- A subclass constructor can call a superclass constructor by use of **super( )**, before doing anything else.
- If you do not call a superclass constructor, the no-argument constructor is automatically called.

- Java does not support the inheritance of multiple superclasses into a single subclass
- You can create a hierarchy of inheritance in which a subclass becomes a superclass of another subclass
- A major advantage of inheritance is that once you have created a superclass that defines the attributes common to a set of objects, it can be used to create any number of more specific subclasses
- Each subclass can precisely tailor its own classification



```
// A subclass of TwoDShape for rectangles.
class Rectangle extends TwoDShape {
  boolean isSquare() {
    if(width == height) return true;
    return false;
  }

  double area() {
    return width * height;
  }
}
```

The **Rectangle** class includes **TwoDShape** and adds the methods **isSquare( )**, which determines if the rectangle is square, and **area( )**, which computes the area of a rectangle.

Member Access and Inheritance

- An instance variable of a class will be declared **private** to prevent its unauthorized use.
- Inheriting a class *does not* overrule the **private** access restriction.
- Thus, even though a subclass includes all of the members of its superclass, it cannot access those members of the superclass that have been declared **private**.
- For example, if, as shown here, **width** and **height** are made private in **TwoDShape**, then **Triangle** will not be able to access them:

```
// Private members are not inherited.    This example will not compile.
// A class for two-dimensional objects.
class TwoDShape {
     private double width;       // these are
     private double height;      // now private
     void showDim() {
             System.out.println("Width and height are " +
                                                   width + " and " + height);
      }
 }
// A subclass of TwoDShape for triangles.
class Triangle extends TwoDShape {
     String style;
     double area() {
             return width * height / 2;         // Error! can't access
       }                  // Can’t access a private member of a superclass.
     void showStyle() {
             System.out.println("Triangle is " + style);
      }
}
```

```
// Use accessor methods to set and get private members.
// A class for two-dimensional objects.
class TwoDShape {
     private double width;      // these are
     private double height; // now private
    // Accessor methods for width and height.
    double getWidth() { return width; }
    double getHeight() { return height; }
    void setWidth(double w) { width = w; }
    void setHeight(double h) { height = h; }
    void showDim() {
       System.out.println("Width and height are " +width + " and " + height);
     }
  }
// A subclass of TwoDShape for triangles.
class Triangle extends TwoDShape {
     String style;
     double area()   {   return getWidth() * getHeight() / 2;  }
    void showStyle() {  System.out.println("Triangle is " + style)  }
 }
```

```
class Shapes2 {
    public static void main(String args[]) {
        Triangle t1 = new Triangle();

        t1.setWidth(4.0);
        t1.setHeight(4.0);
        t1.style = "filled";

        System.out.println("Info for t1: ");
        t1.showStyle();
        t1.showDim();
        System.out.println("Area is " + t1.area());


   }
}
```

```
class Shapes2 {  //Two instances of Triangle
      public static void main(String args[]) {
            Triangle t1 = new Triangle();
            Triangle t2 = new Triangle();
            t1.setWidth(4.0);
            t1.setHeight(4.0);
            t1.style = "filled";
            t2.setWidth(8.0);
            t2.setHeight(12.0);
            t2.style = "outlined";
            System.out.println("Info for t1: ");
            t1.showStyle();
            t1.showDim();
            System.out.println("Area is " + t1.area());
            System.out.println();
            System.out.println("Info for t2: ");
            t2.showStyle();
             t2.showDim();
            System.out.println("Area is " + t2.area());
    }
}
```

Constructors and Inheritance

- It is possible for both superclasses and subclasses to have their own constructors.
- What constructor is responsible for building an object of the subclass—the one in the superclass, the one in the subclass, or both?
- The constructor for the superclass constructs the superclass portion of the object, and the constructor for the subclass constructs the subclass part.
- The superclass has no knowledge of or access to any element in a subclass. Thus, their construction must be separate.
- in practice, most classes will have explicit constructors.
- When only the subclass defines a constructor, it constructs the subclass object.
- The superclass portion of the object is constructed automatically using its default constructor.

```
// Add a constructor to Triangle.
// A class for two-dimensional objects.
class TwoDShape {
    private double width;      // these are
    private double height;     // now private
    // Accessor methods for width and  height.
    double getWidth() { return width; }
    double getHeight() { return height; }
    void setWidth(double w) { width = w; }
    void setHeight(double h) { height = h; }
    void showDim() {
        System.out.println("Width and height are " +
                                          width + " and " + height);
     }
  }
// A subclass of TwoDShape for triangles.
class Triangle extends TwoDShape {
    private String style;
```

```
 // Constructor
Triangle(String s, double w, double h) {
           setWidth(w);
           setHeight(h);
           style = s;
       }
     double area() {
           return getWidth() * getHeight() / 2;
      }
    void showStyle() {
           System.out.println("Triangle is " + style);
     }
 }
class Shapes3 {
     public static void main(String args[]) {
           Triangle t1 = new Triangle("filled", 4.0, 4.0);
           System.out.println("Info for t1: ");
           t1.showStyle();
           t1.showDim();
           System.out.println("Area is " + t1.area());
```

```
          Triangle t2 = new Triangle("outlined", 8.0, 12.0);
          System.out.println();
          System.out.println("Info for t2: ");
          t2.showStyle();
          t2.showDim();
          System.out.println("Area is " + t2.area());
      } }
```

- When both the superclass and the subclass define constructors, both the superclass and subclass constructors must be executed.
- We must use Java’s keyword, **super**, which has **two general forms**.
- **The first** calls a superclass constructor.
- **The second** is used to access a member of the superclass that has been hidden by a member of a subclass.
- A subclass can call a constructor defined by its superclass by use of the following form of super:

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

- Here, parameter-list specifies any parameters needed by the constructor in the superclass.
- super( ) must always be the first statement executed inside a subclass constructor.

```
// Add more constructors to TwoDShape.
class TwoDShape {
        private double width;
        private double height;
        // A default constructor.
        TwoDShape() {
                width = height = 0.0;
         }
         // Parameterized constructor.
        TwoDShape(double w, double h) {
                width = w;
                height = h;
          }
        // Construct object with equal width and height.
        TwoDShape(double x) {
                width = height = x;
          }
        // Accessor methods for width and height.
                double getWidth() { return width; }
                double getHeight() { return height; }
        void setWidth(double w) { width = w; }
        void setHeight(double h) { height = h; }
        void showDim() {
            System.out.println("Width and height are " + width + " and " + height);
        }
    }
    // A subclass of TwoDShape for triangles.
    class Triangle extends TwoDShape {
        private String style;
       // A default constructor.
       Triangle() {
               super();
               style = "none";
        }
        // Constructor
        Triangle(String s, double w, double h) {
               super(w, h); // call superclass constructor
               style = s;
         }
        // One argument constructor.
       Triangle(double x) {
               super(x); // call superclass constructor
```

```
style = "filled";
}
double area() {
return getWidth() * getHeight() / 2;
}
void showStyle() {
System.out.println("Triangle is " + style);
}
}
class Shapes5 {
public static void main(String args[]) {
Triangle t1 = new Triangle();
Triangle t2 = new Triangle("outlined", 8.0, 12.0);
Triangle t3 = new Triangle(4.0);
 t1=t2;
System.out.println("Info for t1: ");
t1.showStyle();
t1.showDim();
System.out.println("Area is " + t1.area());
System.out.println();
System.out.println("Info for t2: ");
t2.showStyle();
```

```
        t2.showDim();
        System.out.println("Area is " + t2.area());
        System.out.println();
        System.out.println("Info for t3: ");
        t3.showStyle();
        t3.showDim();
        System.out.println("Area is " + t3.area());
        }
    }
```

Here is the output from this version:

```
Info for t1:
Triangle is outlined
Width and height are 8.0 and 12.0
Area is 48.0
Info for t2:
Triangle is outlined
Width and height are 8.0 and 12.0
 Area is 48.0
 Info for t3:
 Triangle is filled
 Width and height are 4.0 and 4.0
 Area is 8.0
```

Using super to Access Superclass Members

- There is a **second form of super** that refers to the superclass of the subclass in which it is used. This usage has the following general form:

  super.member

- Here, member can be either a method or an instance variable.
- This is applicable to situations in which member names of a subclass hide members by the same name in the superclass.

```
// Using super to overcome name hiding.
class A {
        int i;
  }
// Create a subclass by extending class A.
class B extends A {
      int i; // this i hides the i in A
      B(int a, int b) {
            super.i = a; // i in A
            i = b; // i in B
         }
      void show() {
            System.out.println("i in superclass: " + super.i);
            System.out.println("i in subclass: " + i);
        }
}
```

```
class UseSuper {
    public static void main(String args[]) {
          B subOb = new B(1, 2);
          subOb.show();
     }
}
```

This program displays the following:

```
i in superclass: 1
i in subclass: 2
```

Creating a Multilevel Hierarchy

- You can build hierarchies that contain as many layers of inheritance as you like.
- It is perfectly acceptable to use a subclass as a superclass of another.
- Given three classes called A, B, and C, C can be a subclass of B, which is a subclass of A. C inherits all aspects of B and A.

- In the following, the subclass Triangle is used as a superclass to create the subclass called ColorTriangle.
- ColorTriangle inherits all of the traits of Triangle and TwoDShape and adds a field called color, which holds the color of the triangle.

```
// A multilevel hierarchy.
class TwoDShape {
        private double width;
        private double height;
        // A default constructor.
        TwoDShape() {
             width = height = 0.0;
          }
        // Parameterized constructor.
       TwoDShape(double w, double h) {
             width = w;
             height = h;
        }
        // Construct object with equal width and height.
        TwoDShape(double x) {
             width = height = x;
         }
       // Accessor methods for width and height.
       double getWidth() { return width; }
       double getHeight() { return height; }
```

```
        void setWidth(double w) { width = w; }
        void setHeight(double h) { height = h; }
        void showDim() {
             System.out.println("Width and height are " +
                                                                width + " and " + height);
         }
  }
// Extend TwoDShape.
class Triangle extends TwoDShape {
         private String style;
         // A default constructor.
        Triangle() {
             super();
             style = "none";
         }
       Triangle(String s, double w, double h) {
             super(w, h); // call superclass constructor
             style = s;
        }
       // One argument constructor.
```

```
Triangle(double x) {
super(x); // call superclass constructor
style = "filled";
}
double area() {
return getWidth() * getHeight() / 2;
}
void showStyle() {
System.out.println("Triangle is " + style);
}
}
// Extend Triangle.
class ColorTriangle extends Triangle {
private String color;
ColorTriangle(String c, String s,
double w, double h) {
super(s, w, h);
color = c;
}
```

```
    String getColor() { return color; }
    void showColor() {
        System.out.println("Color is " + color);
    }
}
class Shapes6 {
    public static void main(String args[]) {
        ColorTriangle t1 =
        new ColorTriangle("Blue", "outlined", 8.0, 12.0);
        ColorTriangle t2 =
        new ColorTriangle("Red", "filled", 2.0, 2.0);
        System.out.println("Info for t1: ");
        t1.showStyle();
        t1.showDim();
        t1.showColor();
        System.out.println("Area is " + t1.area());
        System.out.println();
        System.out.println("Info for t2: ");
        t2.showStyle();
        t2.showDim();
        t2.showColor();
        System.out.println("Area is " + t2.area());
    }
}
```

The output of this program is shown here:

```
Info for t1:
Triangle is outlined
Width and height are 8.0 and 12.0
Color is Blue
Area is 48.0
Info for t2:
Triangle is filled
Width and height are 2.0 and 2.0
Color is Red
Area is 2.0
```

In a class hierarchy, constructors complete their execution in order of derivation, from superclass to subclass. Since super( ) must be the first statement executed in a subclass' constructor, this order is the same whether or not super( ) is used.

when constructors are executed?

```
// Create a super class.
class A {
        A() {
                System.out.println("Constructing A.");
        }
}
// Create a subclass by extending class A.
class B extends A {
         B() {
                System.out.println("Constructing B.");
         }
}
// Create another subclass by extending B.
class C extends B {
          C() {
                System.out.println("Constructing C.");
          }
}
```

```
class OrderOfConstruction {
public static void main(String args[]) {
C c = new C();
}
}
```

The output from this program is shown here:

```
Constructing A.
Constructing B.
Constructing C.
```

Superclass References and Subclass Objects

- A reference variable for one class type cannot normally refer to an object of another class type.

Superclass References and Subclass Objects

- A reference variable for one class type cannot normally refer to an object of another class type.
- A reference variable of a superclass can be assigned a reference to an object of any subclass derived from that superclass.

```
// A superclass reference can refer to a subclass object.
class X {
int a;
X(int i) { a = i; }
}
class Y extends X {
        int b;
        Y(int i, int j) {
             super(j);
             b = i;
         }
}
```

```
class SupSubRef {
    public static void main(String args[]) {
        X x = new X(10);
        X x2;
        Y y = new Y(5, 6);

        x2 = x;             // OK, both of same type
        System.out.println("x2.a: " + x2.a);

        x2 = y;            // still Ok because Y is derived from X
        System.out.println("x2.a: " + x2.a);

        // X references know only about X members
        x2.a = 19;       // OK
        // x2.b = 27;    // Error, X doesn't have a b member
    }
}
```

```
class TwoDShape {    // a superclass reference can refer to a subclass object
      private double width;
      private double height;
      // Construct an object from an object.
      TwoDShape(TwoDShape ob) {   //TwoDShape reference is pointing to
              width = ob.width;                // Triangle class object
              height = ob.height;
       }
// A subclass of TwoDShape for triangles.
class Triangle extends TwoDShape {
       private String style;
       // Construct an object from an object.
       Triangle(Triangle ob) {
              super(ob);              // pass object to TwoDShape constructor
              style = ob.style;
         }     }
class Shapes7 {
       public static void main(String args[]) {
              Triangle t1 =  new Triangle("outlined", 8.0, 12.0);
              // make a copy of t1
              Triangle t2 = new Triangle(t1);
        }   }
```

Progress Check

- Can a subclass be used as a superclass for another superclass?
- In a class hierarchy, in what order are the constructors executed?
- Given that Jet extends Airplane, can an Airplance reference refer to a Jet Object?

Method Overriding

- In a class hierarchy, when a method in a subclass has the same return type and signature as a method in its superclass, then the method in the subclass is said to override the method in the superclass.
- When an overridden method is called from within a subclass, it will always refer to the version of that method defined by the subclass.
- The version of the method defined by the superclass will be hidden.
- If the signatures of the two methods are not identical then the two methods are simply overloaded.

```
// Method overriding.
class A {
    int i, j;
    A(int a, int b) {
        i = a;
        j = b;
    }
    // display i and j
    void show() {
        System.out.println("i and j: " + i + " " + j);
    }
}
```

```
class B extends A {
       int k;
       B(int a, int b, int c) {
             super(a, b);
             k = c;
        }
      void show() {          // display k – this overrides show() in A
             System.out.println(“overridden : " + k);
        }
       void show(String msg) {        // overload show()
             System.out.println(msg + k);  //since signature is different
        }
 }
class Override {
        public static void main(String args[]) {
             B subOb = new B(1, 2, 3);
             subOb.show();      // this calls overridden show() in B
             subOb.show(“overload: ”);     // this calls  overloaded show
         }
  }     //The output produced by this program is shown here:       overridden: 3
         overloaded : 3
```

Overridden Methods Support Polymorphism

- *Dynamic method dispatch* is the mechanism by which a call to an overridden method is resolved at run time rather than compile time.
- Through Dynamic method dispatch, Java implements run-time polymorphism.
- A superclass reference variable can refer to a subclass object.
- Java uses this fact to resolve calls to overridden methods at run time.
- When an overridden method is called through a superclass reference, Java determines which version of that method to execute based upon the type of the object being referred to at the time the call occurs.
- When different types of objects are referred to, different versions of an overridden method will be called.
- It is the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed.
- If a superclass contains a method that is overridden by a subclass, then when different types of objects are referred to through a superclass reference variable, different versions of the method are executed.

```
// Demonstrate dynamic method dispatch.
class Sup {
     void who() {
        System.out.println("who() in Sup");
      }
}
class Sub1 extends Sup {
    void who() {
       System.out.println("who() in Sub1");
     }
}
class Sub2 extends Sup {
     void who() {
        System.out.println("who() in Sub2");
     }
}
class DynDispDemo {
     public static void main(String args[]) {
          Sup superOb = new Sup();
          Sub1 subOb1 = new Sub1();
          Sub2 subOb2 = new Sub2();
          Sup supRef;
          supRef = superOb;
          supRef.who();
          supRef = subOb1;
          supRef.who();
          supRef = subOb2;
          supRef.who();
     }
}
```

The output from the program is shown here:

```
who() in Sup
who() in Sub1
who() in Sub2
```

In each case, the version of **who( )** to call is determined at run time by the type of object being referred to.

Why Overridden Methods?

- Overridden methods allow Java to support run-time polymorphism.
- Polymorphism allows a general class to specify methods that will be common to all of its derivatives, while allowing subclasses to define the specific implementation of some or all of those methods.
- Overridden methods allow Java implements the “one interface, multiple methods” aspect of polymorphism.
- The superclasses and subclasses form a hierarchy that moves from lesser to greater specialization.
- The superclass provides all elements that a subclass can use directly.
- It also defines those methods that the derived class must implement on its own.
- By combining inheritance with overridden methods, a superclass can define the general form of the methods that will be used by all of its subclasses.

Progress Check

- What is method overriding?
- Why is method overriding is important?
- When an overridden method is called through a superclass reference, which version of method is executed

Abstract Classes

- Sometimes you want a class that is only partially implemented and you want to leave it to the subclasses to complete the implementation.
- In that case, use an *abstract* class with *abstract* methods.
- To declare a class or method as abstract, just add the keyword **abstract** in front of the class or method declaration.
- In the case of an abstract method, you must also leave off the body of the method.

Abstract Classes

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

Example of an Abstract Class

```
abstract class Super {
  int x;

  int getX() { return x; }

  abstract void setX(int newX); // no body
}

class Sub extends Super {

  void setX(int newX) { x = newX; }
}
```

Progress Check

- What is an abstract method? How is one created?
- When must a class be declared abstract?
- Can an object of abstract class can be instantiated?

The Keyword final

- If you do not want a class to be subclassed, precede the class declaration with the keyword **final**.
- If you do not want a method to be overridden by a subclass, precede the method declaration with the keyword **final**.
- If you want a variable to be read-only (that is, a constant), precede it with the keyword **final**.

Example Using final

```
//class final, prevents inheritancee
final class MyClass {
  //variable final, read only
  final int x = 3;
  public static final double PI = 3.14159;
  //method final, can not be overridden
  final double getPI() { return PI; }
}
```

Progress Check

- How do you prevent a method being overriding?
- If a class is declared as final, can it be inherited?

The Object Class

- Java defines a special class called **Object** that is an **implicit superclass** of all other classes.
- Therefore, all classes inherit the methods in the **Object** class.
- A variable of type **Object** can refer to an object of any other class, including an array.

Some Methods in the **Object** Class

Method	**Purpose**
Object clone( )	Creates a copy of this object
boolean equals(Object *obj*)	Tests whether two objects are equal
void finalize( )	Called before recycling the object
Class<?> getClass( )	Returns the class of the object
int hashCode( )	Returns the hash code of the object
void notify( )	Resumes execution of a thread waiting on the object
void notifyAll( )	Resumes execution of all threads waiting on the object
String toString( )	Returns a string describing the object
void wait( )	Waits on another thread of execution

Which of the following statements is correct?

a. An object of a super class can access any of the members of its subclass
b. An object of a sub class can access any of the members of its superclass
c. An object of a super class can access only data members of its subclass
d. An object of a super class can access only data members of its subclass

Which of the following statements is correct?

a. An object of a super class can access any of the members of its subclass

b. An object of a sub class can access any of the members of its superclass

c. An object of a super class can access only data members of its subclass

d. An object of a super class can access only data members of its subclass

Which of the following statements is correct?

a. A superclass specifies specialization and a subclass specifies generalization
b. A subclass specifies specialization and a superclass specifies generalization
c. A superclass specifies containership and a subclass specifies generalization
d. A subclass specifies containership and a superclass specifies generalization

Which of the following statements is correct?

a. A superclass specifies specialization and a subclass specifies generalization

b. A subclass specifies specialization and a superclass specifies generalization

c. A superclass specifies containership and a subclass specifies generalization

d. A subclass specifies containership and a superclass specifies generalization

‘super’can be used to

a. pass the parameters to super class constructor
b. Refer to the hidden member of the super class
c. Both a and b above are true
d. None of the above

‘super’can be used to

a. pass the parameters to super class constructor
b. Refer to the hidden member of the super class
c. Both a and b above are true
d. None of the above

In Java

a. A super class reference can refer a sub class object
b. A sub class reference can refer a super class object
c. Both a and b above are rue
d. None of the above

In Java

a. A super class reference can refer a sub class object
b. A sub class reference can refer a super class object
c. Both a and b above are rue
d. None of the above

Which of the following is a superclass of every class in Java?

a) ArrayList
b) Abstract class
c) Object class
d) String

Which of the following is a superclass of every class in Java?

a) ArrayList

b) Abstract class

c) Object class

d) String

‘final’ can be used to

a. Prevent a method from being overridden
b. Create named constants
c. Prevent a class being inherited
d. All the above

‘final’ can be used to

a. Prevent a method from being overridden
b. Create named constants
c. Prevent a class being inherited
d. All the above

```java
class A
{
public int i;
private int j;
}
class B extends A
{
void display()
{
super.j = super.i + 1;
System.out.println(super.i + " " + super.j);
}
}
class inheritance
{
public static void main(String args[])
{
B obj = new B();
obj.i=1;
obj.j=2;
obj.display();
}
}
```

- Compilation Error

Class contains a private member variable j, this cannot be inherited by subclass B and does not have access to it.

```
class A
{
public int i;
protected int j;
}
class B extends A
{
int j;
void display()
{
super.j = 3;
System.out.println(i + " " + j);
}
}
class Output
{
public static void main(String args[])
{
B obj = new B();
obj.i=1;
obj.j=2;
obj.display();
}
}
```

- 1 2

Both class A & B have member with same name that is j, member of class B will be called by default if no specifier is used. I contains 1 & j contains 2, printing 1 2.

```java
final class A
{
int i;
}
class B extends A
{
int j;
System.out.println(j + " " + i);
}
class inheritance
{
public static void main(String args[])
{
B obj = new B();
obj.display();
}
}
```

- Compilation Error

class A has been declared final hence it cannot be inherited by any other class. Hence class B does not have member i, giving compilation error.