JAVA PROGRAMMING

Week 3: Inheritance

Lecturer:

- Ho Tuan Thanh, M.Sc.

Plan

1. Understand inheritance basics
2. Member access and inheritance
3. Constructors and inheritance
4. Using super
5. Create a multilevel class hierarchy
6. Override methods
7. Use abstract classes

Plan

1. Understand inheritance basics
2. Member access and inheritance
3. Constructors and inheritance
4. Using super
5. Create a multilevel class hierarchy
6. Override methods
7. Use abstract classes

Inheritance

- One of the three foundation principles of object-oriented programming
  - It 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 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.

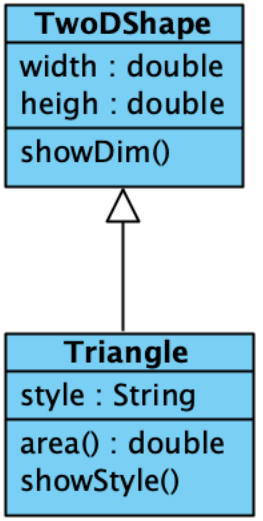
→It inherits all of the variables and methods defined by the superclass and adds its own, unique elements.

INHERITANCE BASICS

- Java supports inheritance by allowing one class to incorporate another class into its declaration.
  - Keyword: extends
  - The subclass adds to (extends) the superclass.

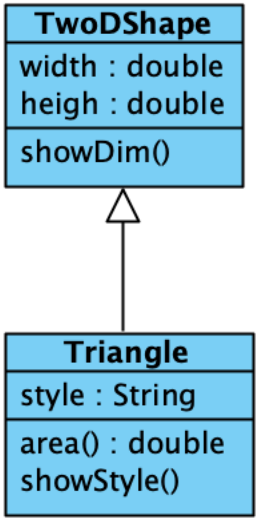
Example [1]


```
1.  //A simple class hierarchy.
2.
3.  //A class for two-dimensional objects.
4.  class TwoDShape {
5.      double width;
6.      double height;
7.
8.      void showDim() {
9.          System.out.println("Width and height are " +
10.                                 width + " and " + height);
11.     }
12. }
```



Example [2]


```
1.  //A subclass of TwoDShape for triangles.
2.  class Triangle extends TwoDShape {
3.      String style;
4.
5.      double area() {
6.          return width * height / 2;
7.      }
8.
9.      void showStyle() {
10.         System.out.println("Triangle is " + style);
11.     }
12. }
```



Example [3]

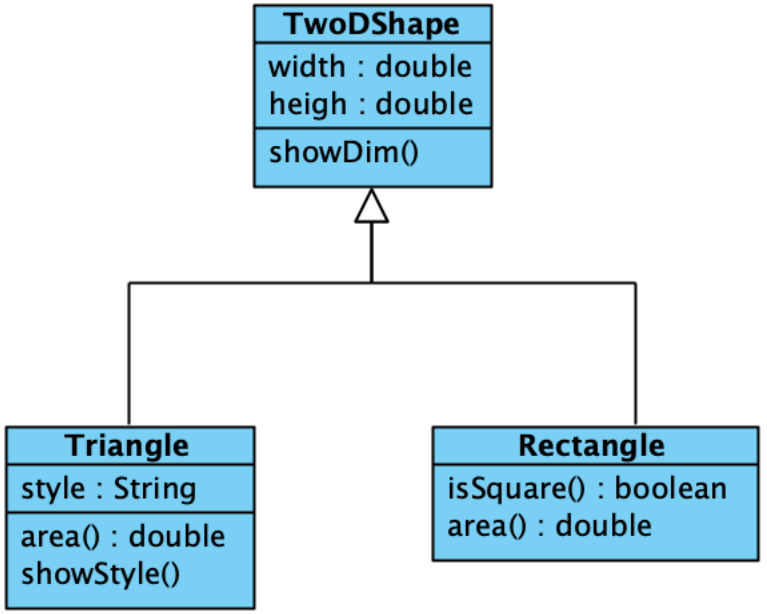

```
1.  class Shapes {
2.      public static void main(String args[]) {
3.          Triangle t1 = new Triangle();
4.          Triangle t2 = new Triangle();
5.
6.          t1.width = 4.0;
7.          t1.height = 4.0;
8.          t1.style = "filled";
9.
10.         t2.width = 8.0;
11.         t2.height = 12.0;
12.         t2.style = "outlined";
13.
14.         System.out.println("Info for t1: ");
15.         ...
```

Example [4]


```
1.              ...
2.              t1.showStyle();
3.              t1.showDim();
4.              System.out.println("Area is " + t1.area());
5.
6.              System.out.println();
7.
8.              System.out.println("Info for t2: ");
9.              t2.showStyle();
10.             t2.showDim();
11.             System.out.println("Area is " + t2.area());
12.         }
13. }
```

Another example


```
1.  //A subclass of TwoDShape for rectangles.
2.  class Rectangle extends TwoDShape {
3.      boolean isSquare() {
4.          if (width == height)
5.              return true;
6.          return false;
7.      }
8.
9.      double area() {
10.         return width * height;
11.     }
12. }
```



Plan

1. Understand inheritance basics
2. Member access and inheritance
3. Constructors and inheritance
4. Using super
5. Create a multilevel class hierarchy
6. Override methods
7. Use abstract classes

MEMBER ACCESS AND INHERITANCE

- An instance variable of a class will be declared private to prevent its unauthorized use or tampering.
- Inheriting a class does not overrule the private access restriction.
- → 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.

Example [1]


```
1.  //Private members are not inherited.
2.  //This example will not compile.
3.  //A class for two-dimensional objects.
4.  class TwoDShape {
5.      private double width; // these are
6.      private double height; // now private
7.
8.      void showDim() {
9.          System.out.println("Width and height are " +
10.                                 width + " and " + height);
11.     }
12. }
```

Example [2]

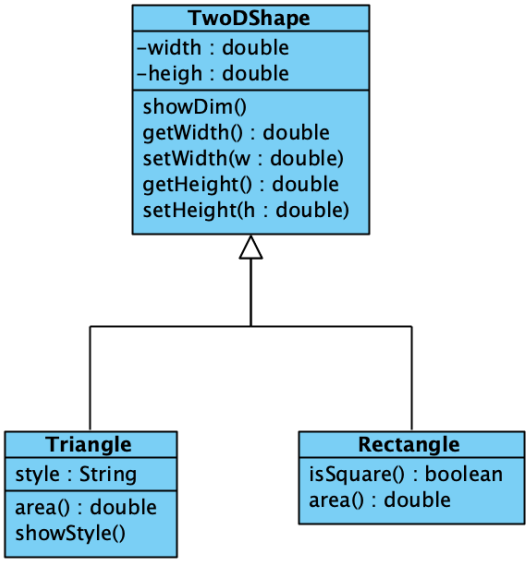

```
1.  //A subclass of TwoDShape for triangles.
2.  class Triangle extends TwoDShape {
3.      String style;
4.
5.      double area() {
6.          return width * height / 2; // Error! can't access
7.      }
8.
9.      void showStyle() {
10.         System.out.println("Triangle is " + style);
11.     }
12. }
```

Improved version [1]


```
1.  //Use accessor methods to set and get private members.
2.  class TwoDShape {
3.      private double width; // these are
4.      private double height; // now private
5.      // Accessor methods for width and height.
6.      double getWidth() { return width; }
7.      double getHeight() { return height; }
8.      void setWidth(double w) { width = w; }
9.      void setHeight(double h) { height = h; }
10.     void showDim() {
11.         System.out.println("Width and height are " +
12.                                 width + " and " + height);
13.     }
14. }
```

Improved version [2]


```
1.  //A subclass of TwoDShape for triangles.
2.  class Triangle extends TwoDShape {
3.      String style;
4.
5.      double area() {
6.          return getWidth() * getHeight() / 2;
7.      }
8.
9.      void showStyle() {
10.         System.out.println("Triangle is " + style);
11.     }
12. }
13.
```



Plan

1. Understand inheritance basics
2. Member access and inheritance
3. Constructors and inheritance
4. Using super
5. Create a multilevel class hierarchy
6. Override methods
7. Use abstract classes

CONSTRUCTORS AND INHERITANCE

- In a hierarchy: it is possible for both superclasses and subclasses to have their own constructors.
- The constructor for the superclass constructs the superclass portion of the object, and the constructor for the subclass constructs the subclass part.
- When only the subclass defines a constructor: simply construct the subclass object.
  - The superclass portion of the object is constructed automatically using its default constructor.

Example: Constructors [1]


```
1.  //A class for two-dimensional objects.
2.  class TwoDShape {
3.      private double width; // these are
4.      private double height; // now private
5.      // Accessor methods for width and height.
6.      double getWidth() { return width; }
7.      double getHeight() { return height; }
8.      void setWidth(double w) { width = w; }
9.      void setHeight(double h) { height = h; }
10.     void showDim() {
11.         System.out.println("Width and height are " +
12.                                 width + " and " + height);
13.     }
14. }
```

```
1.  //A subclass of TwoDShape for triangles.
2.  class Triangle extends TwoDShape {
3.      private String style;
4.      // Constructor
5.      Triangle(String s, double w, double h) {
6.          setWidth(w);
7.          setHeight(h);
8.          style = s;
9.      }
10.     double area() { return getWidth() * getHeight() / 2; }
11.     void showStyle() {
12.         System.out.println("Triangle is " + style);
13.     }
14. }
```

Plan

1. Understand inheritance basics
2. Member access and inheritance
3. Constructors and inheritance
4. Using super
5. Create a multilevel class hierarchy
6. Override methods
7. Use abstract classes

USING SUPER TO CALL SUPERCLASS CONSTRUCTORS

- A subclass can call a constructor defined by its superclass by use of the following form of super:

```
super(parameterlist);
```

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

```
1.  //Add constructors to TwoDShape.
2.  class TwoDShape {
3.      private double width;
4.      private double height;
5.      // Parameterized constructor.
6.      TwoDShape(double w, double h) {
7.          width = w;
8.          height = h;
9.      }
10.     // Accessor methods for width and height.
11.     double getWidth() { return width; }
12.     double getHeight() { return height; }
13.     void setWidth(double w) { width = w; }
14.     void setHeight(double h) { height = h; }
15.     void showDim() {
16.         System.out.println("Width and height are " +
17.                                 width + " and " + height);
18.     }
19. }
```

```
TwoDShape(double w, double h) {
    width = w;
    height = h;
}
```

```
1.  //A subclass of TwoDShape for triangles.
2.  class Triangle extends TwoDShape {
3.      private String style;
4.      Triangle(String s, double w, double h) {
5.          super(w, h); // call superclass constructor
6.          style = s;
7.      }
8.      double area() {
9.          return getWidth() * getHeight() / 2;
10.     }
11.     void showStyle() {
12.         System.out.println("Triangle is " + style);
13.     }
14. }
```

USING SUPER TO ACCESS SUPERCLASS MEMBERS

super.member

- member can be either a method or an instance variable.

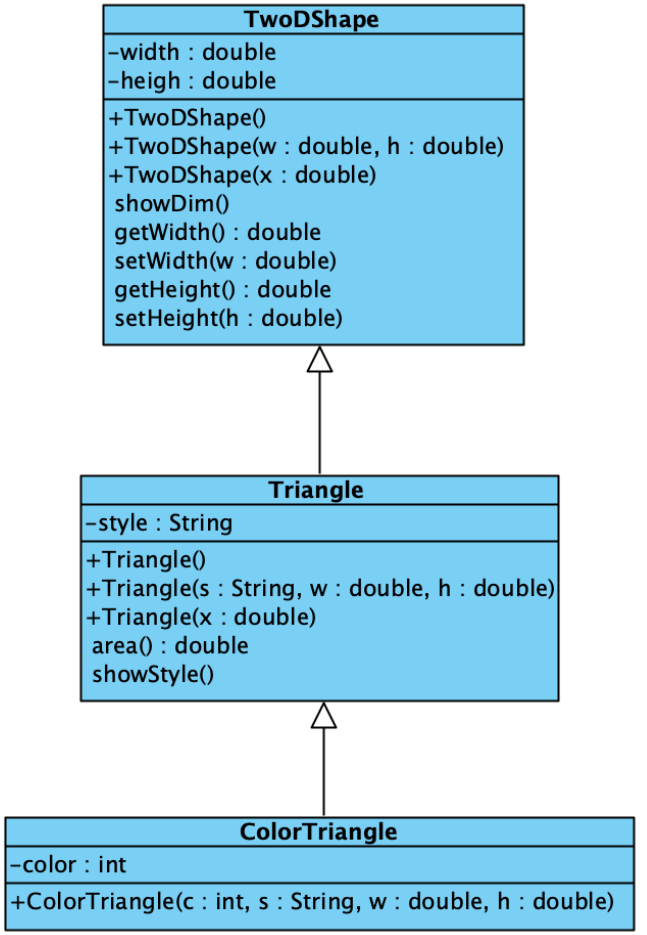
- This form of super is most applicable to situations in which member names of a subclass hide members by the same name in the superclass.

```
1.  //Using super to overcome name hiding.
2.  class A { int i; }
3.  //Create a subclass by extending class A.
4.  class B extends A {
5.      int i; // this i hides the i in A
6.      B(int a, int b) {
7.          super.i = a; // i in A
8.          i = b; // i in B
9.      }
10.     void show() {
11.         System.out.println("i in superclass: " + super.i);
12.         System.out.println("i in subclass: " + i);
13.     }
14. }
15. class UseSuper {
16.     public static void main(String args[]) {
17.         B subOb = new B(1, 2); subOb.show();
18.     }
19. }
```

Plan

1. Understand inheritance basics
2. Member access and inheritance
3. Constructors and inheritance
4. Using super
5. Create a multilevel class hierarchy
6. Override methods
7. Use abstract classes

Example


TwoDShape
-width : double
-heigh : double
+TwoDShape()
+TwoDShape(w : double, h : double)
+TwoDShape(x : double)
showDim()
getWidth() : double
setWidth(w : double)
getHeight() : double
setHeight(h : double)
Triangle
-style : String
+Triangle()
+Triangle(s : String, w : double, h : double)
+Triangle(x : double)
area() : double
showStyle()
ColorTriangle
-color : int
+ColorTriangle(c : int, s : String, w : double, h : double)

```
1.  class TwoDShape {
2.      private double width;
3.      private double height;
4.      // A default constructor.
5.      TwoDShape() { width = height = 0.0; }
6.      // Parameterized constructor.
7.      TwoDShape(double w, double h) { width = w; height = h; }
8.      // Construct object with equal width and height.
9.      TwoDShape(double x) { width = height = x; }
10.     // Accessor methods for width and height.
11.     double getWidth() { return width; }
12.     double getHeight() { return height; }
13.     void setWidth(double w) { width = w; }
14.     void setHeight(double h) { height = h; }
15.     void showDim() {
16.         System.out.println("Width and height are " + width +
17.                                                 " and " + height);
18.     }
19. }
```

```
1.  class Triangle extends TwoDShape {
2.      private String style;
3.      Triangle() {// A default constructor.
4.          super(); style = "none";
5.      }
6.      Triangle(String s, double w, double h) {
7.          super(w, h); // call superclass constructor
8.          style = s;
9.      }
10.     // One argument constructor.
11.     Triangle(double x) {
12.         super(x); // call superclass constructor
13.         style = "filled";
14.     }
15.     double area() { return getWidth() * getHeight() / 2; }
16.     void showStyle() {
17.         System.out.println("Triangle is " + style);
18.     }
19. }
```

```
1.  //Extend Triangle.
2.  class ColorTriangle extends Triangle {
3.      private String color;
4.      ColorTriangle(String c, String s, double w, double h) {
5.          super(s, w, h);
6.          color = c;
7.      }
8.      String getColor() {
9.          return color;
10.     }
11.     void showColor() {
12.         System.out.println("Color is " + color);
13.     }
14. }
```

```
1.  class Shapes {
2.      public static void main(String args[]) {
3.          ColorTriangle t1 = new ColorTriangle("Blue",
4.                                               "outlined", 8.0, 12.0);
5.          ColorTriangle t2 = new ColorTriangle("Red",
6.                                               "filled", 4.0, 2.0);
7.          System.out.println("Info for t1: ");
8.          t1.showStyle();
9.          t1.showDim();
10.         t1.showColor();
11.         System.out.println("Area is " + t1.area());
12.         System.out.println();
13.         System.out.println("Info for t2: ");
14.         t2.showStyle();
15.         t2.showDim();
16.         t2.showColor();
17.         System.out.println("Area is " + t2.area());
18.     }
19. }
```

WHEN ARE CONSTRUCTORS EXECUTED?

- When a subclass object is created, whose constructor is executed first, the one in the subclass or the one defined by the superclass?

→In a class hierarchy, constructors complete their execution in order of derivation, from superclass to subclass.

→Further, since super( ) must be the first statement executed in a subclass' constructor, this order is the same whether or not super( ) is used. If super( ) is not used, then the default (parameterless) constructor of each superclass will be executed.

```
1.  //Demonstrate when constructors are called.
2.  //Create a super class.
3.  class A {
4.      A() { System.out.println("Constructing A."); }
5.  }
6.  //Create a subclass by extending class A.
7.  class B extends A {
8.      B() { System.out.println("Constructing B."); }
9.  }
10. //Create another subclass by extending B.
11. class C extends B {
12.     C() { System.out.println("Constructing C."); }
13. }
14.
15. class OrderOfConstruction {
16.     public static void main(String args[]) {
17.         C c = new C();
18.     }
19. }
```

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

SUPERCLASS REFERENCES AND SUBCLASS OBJECTS

- Java is a strongly typed language.
- Aside from the standard conversions and automatic promotions that apply to its primitive types, type compatibility is strictly enforced.
- → A reference variable for one class type cannot normally refer to an object of another class type.

```
1.  //This will not compile.
2.  class X {
3.          int a;
4.          X(int i) { a = i; }
5.  }
6.  class Y {
7.          int a;
8.          Y(int i) { a = i; }
9.  }
10. class IncompatibleRef {
11.         public static void main(String args[]) {
12.                 X x = new X(10);
13.                 X x2;
14.                 Y y = new Y(5);
15.                 x2 = x; // OK, both of same type
16.                 x2 = y; // Error, not of same type
17.         }
18. }
```

Exemple [1]


```
1.  //A superclass reference can refer to a subclass object.
2.  class X {
3.      int a;
4.      X(int i) {
5.          a = i;
6.      }
7.  }
8.  class Y extends X {
9.      int b;
10.     Y(int i, int j) {
11.         super(j);
12.         b = i;
13.     }
14. }
```

Exemple [2]


```
1.  class SupSubRef {
2.      public static void main(String args[]) {
3.          X x = new X(10);
4.          X x2;
5.          Y y = new Y(5, 6);
6.          x2 = x; // OK, both of same type
7.          System.out.println("x2.a: " + x2.a);
8.          x2 = y; // still Ok because Y is derived from X
9.          System.out.println("x2.a: " + x2.a);
10.         // X references know only about X members
11.         x2.a = 19; // OK
12.         // x2.b = 27; // Error, X doesn't have a b member
13.     }
14. }
```

```
1.  class TwoDShape {
2.      private double width;
3.      private double height;
4.      TwoDShape() { width = height = 0.0; }
5.      TwoDShape(double w, double h) { width = w; height = h; }
6.      TwoDShape(double x) { width = height = x; }
7.      TwoDShape(TwoDShape ob) {
8.          width = ob.width; height = ob.height;
9.      }
10.     // Accessor methods for width and height.
11.     double getWidth() { return width; }
12.     double getHeight() { return height; }
13.     void setWidth(double w) { width = w; }
14.     void setHeight(double h) { height = h; }
15.     void showDim() {
16.         System.out.println("Width and height are " + width
17.                                                 + " and " + height);
18.     }
19. }
```

```
1.  class Triangle extends TwoDShape {
2.      private String style;
3.      // Constructors
4.      Triangle() { super(); style = "none"; }
5.      Triangle(String s, double w, double h) {
6.          super(w, h);  style = s;
7.      }
8.      Triangle(double x) {
9.          super(x);  style = "filled";
10.     }
11.     Triangle(Triangle ob) {
12.         super(ob); // pass object to TwoDShape constructor
13.         style = ob.style;
14.     }
15.     double area() { return getWidth() * getHeight() / 2; }
16.     void showStyle(){
17.         System.out.println("Triangle is " + style);
18.     }
19. }
```

```
1.  class Shapes {
2.      public static void main(String args[]) {
3.          Triangle t1 = new Triangle("outlined", 8.0, 12.0);
4.          Triangle t2 = new Triangle(t1);
5.          System.out.println("Info for t1: ");
6.          t1.showStyle();
7.          t1.showDim();
8.          System.out.println("Area is " + t1.area());
9.          System.out.println();
10.         System.out.println("Info for t2: ");
11.         t2.showStyle();
12.         t2.showDim();
13.         System.out.println("Area is " + t2.area());
14.     }
15. }
```

Plan

1. Understand inheritance basics
2. Member access and inheritance
3. Constructors and inheritance
4. Using super
5. Create a multilevel class hierarchy
6. Override methods
7. Use abstract classes

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.

```
1.  //Method overriding.
2.  class A {
3.          int i, j;
4.          A(int a, int b) { i = a; j = b; }
5.          void show() {
6.                  System.out.println("i and j: " + i + " " + j);
7.          }
8.  }
9.  class B extends A {
10.         int k;
11.         B(int a, int b, int c) { super(a, b); k = c; }
12.         void show() { System.out.println("k: " + k); }
13. }
14. class Override {
15.         public static void main(String args[]) {
16.                 B subOb = new B(1, 2, 3);
17.                 subOb.show(); // this calls show() in B
18.         }
19. }
```

```
1.  /* Methods with differing type signatures are
2.  overloaded and not overridden. */
3.  class A {
4.      int i, j;
5.      A(int a, int b) { i = a; j = b; }
6.      // display i and j
7.      void show() {
8.          System.out.println("i and j: " + i + " " + j);
9.      }
10. }
11. //Create a subclass by extending class A.
12. class B extends A {
13.     int k;
14.     B(int a, int b, int c) { super(a, b); k = c; }
15.     // overload show()
16.     void show(String msg) {
17.         System.out.println(msg + k);
18.     }
19. }
```

```
1.  class Overload {
2.      public static void main(String args[]) {
3.          B subOb = new B(1, 2, 3);
4.          subOb.show("This is k: "); // this calls show() in B
5.          subOb.show(); // this calls show() in A
6.      }
7.  }
```

OVERRIDDEN METHODS SUPPORT POLYMORPHISM

- Method overriding forms the basis for one of Java's most powerful concepts: dynamic method dispatch.
  - Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time rather than compile time.
  - Dynamic method dispatch is important because this is how Java implements runtime polymorphism.

```
1.  //Demonstrate dynamic method dispatch.
2.  class Sup {
3.          void who() {
4.                  System.out.println("who() in Sup");
5.          }
6.  }
7.
8.  class Sub1 extends Sup {
9.          void who() {
10.                 System.out.println("who() in Sub1");
11.         }
12. }
13.
14. class Sub2 extends Sup {
15.         void who() {
16.                 System.out.println("who() in Sub2");
17.         }
18. }
```

```
1.  class DynDispDemo {
2.      public static void main(String args[]) {
3.          Sup superOb = new Sup();
4.          Sub1 subOb1 = new Sub1();
5.          Sub2 subOb2 = new Sub2();
6.
7.          Sup supRef;
8.
9.          supRef = superOb;
10.         supRef.who();
11.
12.         supRef = subOb1;
13.         supRef.who();
14.
15.         supRef = subOb2;
16.         supRef.who();
17.     }
18. }
```

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

WHY OVERRIDDEN METHODS?

- Overridden methods allow Java to support runtime polymorphism
- It 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.
- They are another way that Java implements the “one interface, multiple methods” aspect of polymorphism.

Example [1]


```
1.  class TwoDShape {
2.      private double width;
3.      private double height;
4.      private String name;
5.      // Constructors
6.      TwoDShape() { width = height = 0.0; name = "none"; }
7.      TwoDShape(double w, double h, String n) {
8.          width = w; height = h; name = n;
9.      }
10.     TwoDShape(double x, String n) {
11.         width = height = x; name = n;
12.     }
13.     TwoDShape(TwoDShape ob) {
14.         width = ob.width; height = ob.height; name = ob.name;
15.     }
```

Example [2]


```
1.      // Accessor methods for width and height.
2.      double getWidth() { return width; }
3.      double getHeight() { return height; }
4.      void setWidth(double w) { width = w;}
5.      void setHeight(double h) { height = h; }
6.      String getName() { return name; }
7.      void showDim() {
8.          System.out.println("Width and height are " +
9.                                          width + " and " + height);
10.     }
11.     double area() {
12.         System.out.println("area() must be overridden");
13.         return 0.0;
14.     }
15. }
```

```
1.  //A subclass of TwoDShape for triangles.
2.  class Triangle extends TwoDShape {
3.      private String style;
4.      // Constructors for Triangle
5.      Triangle() { super(); style = "none"; }
6.      Triangle(String s, double w, double h) {
7.          super(w, h, "triangle"); style = s;
8.      }
9.      Triangle(double x) {
10.         super(x, "triangle");  style = "filled";
11.     }
12.     Triangle(Triangle ob) { super(ob);  style = ob.style; }
13.     // Override area() for Triangle.
14.     double area() { return getWidth() * getHeight() / 2; }
15.     void showStyle() {
16.         System.out.println("Triangle is " + style);
17.     }
18. }
```

```
1.  //A subclass of TwoDShape for rectangles.
2.  class Rectangle extends TwoDShape {
3.          // Constructors for Rectangle
4.          Rectangle() { super(); }
5.          Rectangle(double w, double h){super(w, h, "rectangle"); }
6.          Rectangle(double x) { super(x, "rectangle");  }
7.          Rectangle(Rectangle ob) {
8.                  super(ob); // pass object to TwoDShape constructor
9.          }
10.
11.         boolean isSquare() {
12.                 if (getWidth() == getHeight()) return true;
13.                 return false;
14.         }
15.         // Override area() for Rectangle.
16.         double area() {
17.                 return getWidth() * getHeight();
18.         }
19. }
```

```
1.  class DynShapes {
2.      public static void main(String args[]) {
3.          TwoDShape shapes[] = new TwoDShape[5];
4.
5.          shapes[0] = new Triangle("outlined", 8.0, 12.0);
6.          shapes[1] = new Rectangle(10);
7.          shapes[2] = new Rectangle(10, 4);
8.          shapes[3] = new Triangle(7.0);
9.          shapes[4] = new TwoDShape(10, 20, "generic");
10.
11.         for (int i = 0; i < shapes.length; i++) {
12.             System.out.println("object is " +
13.                                                 shapes[i].getName());
14.             System.out.println("Area is " + shapes[i].area());
15.
16.             System.out.println();
17.         }
18.     }
19. }
```

Plan

1. Understand inheritance basics
2. Member access and inheritance
3. Constructors and inheritance
4. Using super
5. Create a multilevel class hierarchy
6. Override methods
7. Use abstract classes

USING ABSTRACT CLASSES

- An abstract method is created by specifying the abstract type modifier.
  - An abstract method contains no body and is, therefore, not implemented by the superclass.
  - A subclass must override it—it cannot simply use the version defined in the superclass.
- General form:

```
abstract type name(parameterlist);
```

- The abstract modifier can be used only on instance methods. It cannot be applied to static methods or to constructors.

```
1.  //Create an abstract class.
2.  abstract class TwoDShape {
3.      private double width;
4.      private double height;
5.      private String name;
6.      // Constructors
7.      TwoDShape() { width = height = 0.0; name = "none"; }
8.      TwoDShape(double w, double h, String n) {
9.          width = w; height = h; name = n;
10.     }
11.     TwoDShape(double x, String n) {
12.         width = height = x; name = n;
13.     }
14.     TwoDShape(TwoDShape ob) {
15.         width = ob.width; height = ob.height; name = ob.name;
16.     }
```

```
1.  // Accessor methods for width and height.
2.      double getWidth() { return width; }
3.      double getHeight() { return height; }
4.      void setWidth(double w) { width = w; }
5.      void setHeight(double h) { height = h; }
6.      String getName() { return name; }
7.      void showDim() {
8.          System.out.println("Width and height are " +
9.                                  width + " and " + height);
10.     }
11.     // Now, area() is abstract.
12.     abstract double area();
13. }
```

```
1.  //A subclass of TwoDShape for triangles.
2.  class Triangle extends TwoDShape {
3.      private String style;
4.      // Constructors
5.      // ....
6.
7.      double area() {
8.          return getWidth() * getHeight() / 2;
9.      }
10.     // ....
11. }
```

```
1.  //A subclass of TwoDShape for rectangles.
2.  class Rectangle extends TwoDShape {
3.      // Constructors
4.      // ...
5.      boolean isSquare() {
6.          if (getWidth() == getHeight())
7.              return true;
8.          return false;
9.      }
10.     double area() {
11.         return getWidth() * getHeight();
12.     }
13. }
```

```
1.  class AbsShape {
2.      public static void main(String args[]) {
3.          TwoDShape shapes[] = new TwoDShape[4];
4.          shapes[0] = new Triangle("outlined", 8.0, 12.0);
5.          shapes[1] = new Rectangle(10);
6.          shapes[2] = new Rectangle(10, 4);
7.          shapes[3] = new Triangle(7.0);
8.          for (int i = 0; i < shapes.length; i++) {
9.              System.out.println("object is " +
10.                                          shapes[i].getName());
11.             System.out.println("Area is " + shapes[i].area());
12.             System.out.println();
13.         }
14.     }
15. }
```

USING FINAL

- In Java it is easy to prevent a method from being overridden or a class from being inherited by using the keyword final.
- To prevent a method from being overridden, specify final as a modifier at the start of its declaration.
  - Methods declared as final cannot be overridden.
- You can prevent a class from being inherited by preceding its declaration with final.
  - Declaring a class as final implicitly declares all of its methods as final, too.

Example


```
1.  class A {
2.      final void meth() {
3.          System.out.println("This is a final method.");
4.      }
5.  }
6.
7.  class B extends A {
8.      void meth() { // ERROR! Can't override.
9.          System.out.println("Illegal!");
10.     }
11. }
```

Example


```
1.  final class A{
2.  // ...
3.  }
4.
5.  // The following class is illegal
6.  class B extends A{
7.          // Error: The type B cannot subclass the final class A
8.  }
```

Using final with Data Members [1]


```
1.  // Return a String object.
2.  class ErrorMsg {
3.      // Error codes.
4.      final int OUTERR = 0;
5.      final int INERR = 1;
6.      final int DISKERR = 2;
7.      final int INDEXERR = 3;
8.      String msgs[] = { "Output Error", "Input Error",
9.                        "Disk Full", "Index Out-Of-Bounds" };
10.     // Return the error message.
11.     String getErrorMsg(int i) {
12.         if (i >= 0 & i < msgs.length) return msgs[i];
13.         else return "Invalid Error Code";
14.     }
15. }
```

Using final with Data Members [2]


```
1.  class FinalD {
2.      public static void main(String args[]) {
3.          ErrorMsg err = new ErrorMsg();
4.
5.          System.out.println(err.getErrorMsg(err.OUTERR));
6.          System.out.println(err.getErrorMsg(err.DISKERR));
7.      }
8.  }
```

CLASS OBJECT

Method	Purpose
Object clone( )	Creates a new object that is the same as the object being cloned.
boolean equals(Object *object*)	Determines whether one object is equal to another.
void finalize( )	Called before an unused object is recycled. (Deprecated by JDK 9.)
Class<?> getClass( )	Obtains the class of an object at run time.
int hashCode( )	Returns the hash code associated with the invoking object.
void notify( )	Resumes execution of a thread waiting on the invoking object.
void notifyAll( )	Resumes execution of all threads waiting on the invoking object.
String toString( )	Returns a string that describes the object.
void wait( ) void wait(long *milliseconds*) void wait(long *milliseconds*, int *nanoseconds*)	Waits on another thread of execution.

QUESTION ?