# LẬP TRÌNH JAVA

## CHƯƠNG 3: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG TRONG JAVA

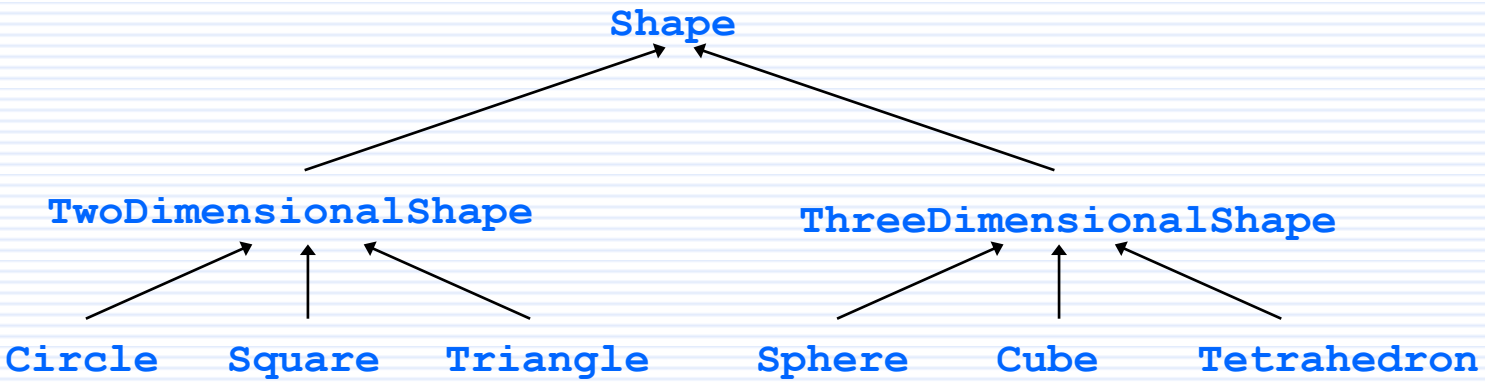
# Nội dung

- ✓ Nhắc lại các khái niệm cơ bản trong lập trình hướng đối tượng
- ✓ Lập trình hướng đối tượng trong Java
  - Tính kế thừa
  - Tính đa hình
    - Lớp trừu tượng
    - Giao diện

# Các khái niệm cơ bản trong Lập trình hướng đối tượng

- ✓ Đóng gói hay Lớp (Class):
  - Gồm 2 thành phần cơ bản: thuộc tính và phương thức.
- ✓ Kế thừa (Inheritance): lớp mới định nghĩa từ lớp có sẵn.
  - Lớp con kế thừa thuộc tính và phương thức của lớp cha.
  - Có 2 kiểu kế thừa: Đơn kế thừa và đa kế thừa
- ✓ Đa hình (Polymorphism)
  - Lớp trừu tượng - Abstraction
  - Giao diện - Interface

# Kế thừa trong Java



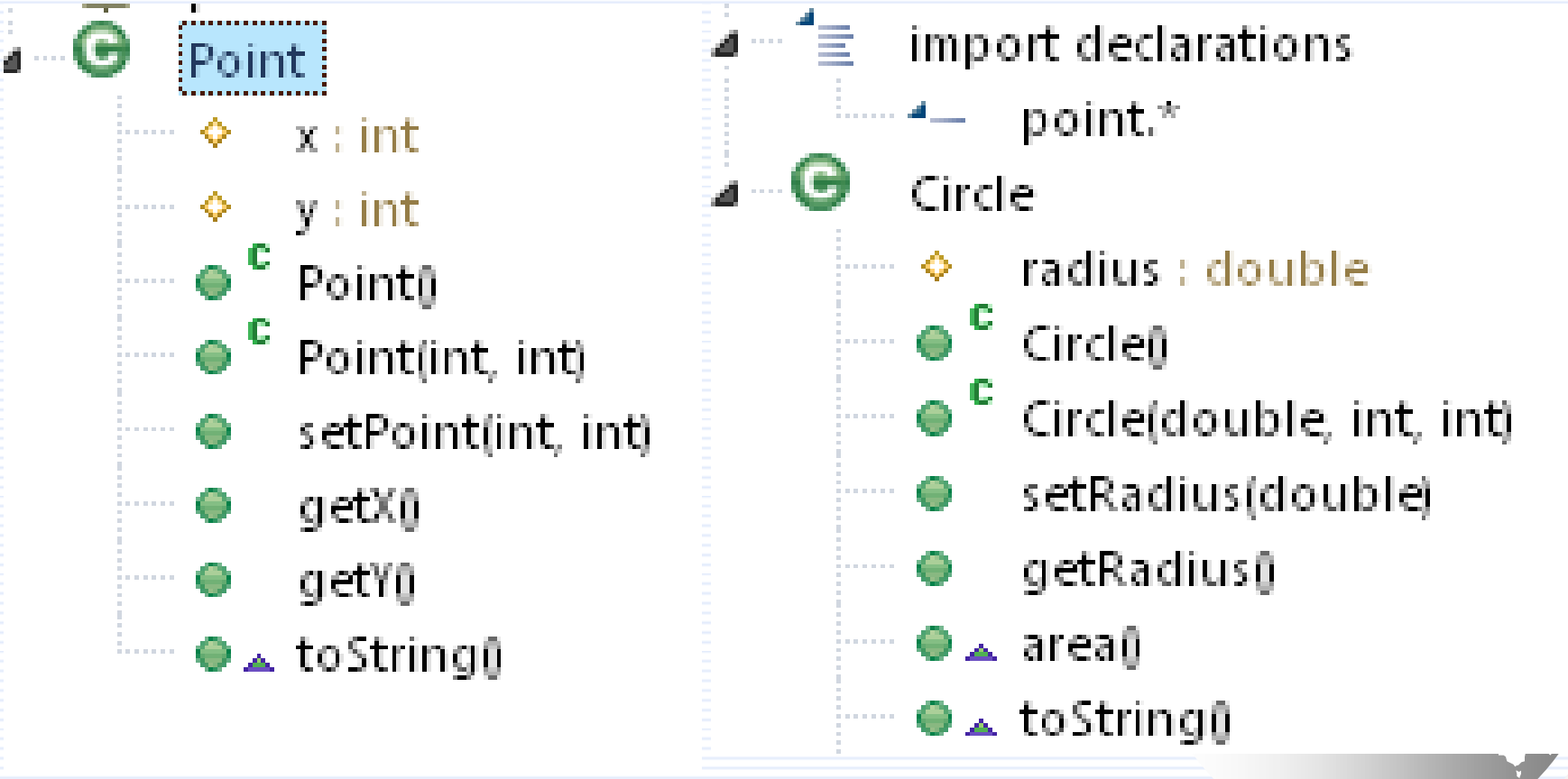

- ✓ Dùng từ khóa **`extends`**: **Đơn kế thừa**

```
class TwoDimensionalShape extends Shape{
... }
```

- ✓ Các thành phần **`private`** của lớp cha không được truy xuất trực tiếp từ lớp con.
- ✓ Tất cả các thuộc tính khác giữ nguyên quyền truy xuất.

# Kế thừa trong Java

- ✓ Sơ đồ lớp một ví dụ kế thừa



```java
// Fig. 27.3: Point.java
// Definition of class Point

public class Point {
   protected int x, y; // coordinates of the Point

   // No-argument constructor
   public Point()
   {
      // implicit call to superclass constructor occurs here
      setPoint( 0, 0 );
   }

   // Constructor
   public Point( int a, int b )
   {
      // implicit call to superclass constructor occurs here
      setPoint( a, b );
   }

   // Set x and y coordinates of Point
   public void setPoint( int a, int b )
   {
      x = a;
      y = b;
   }

   // get x coordinate
   public int getX() { return x; }

```

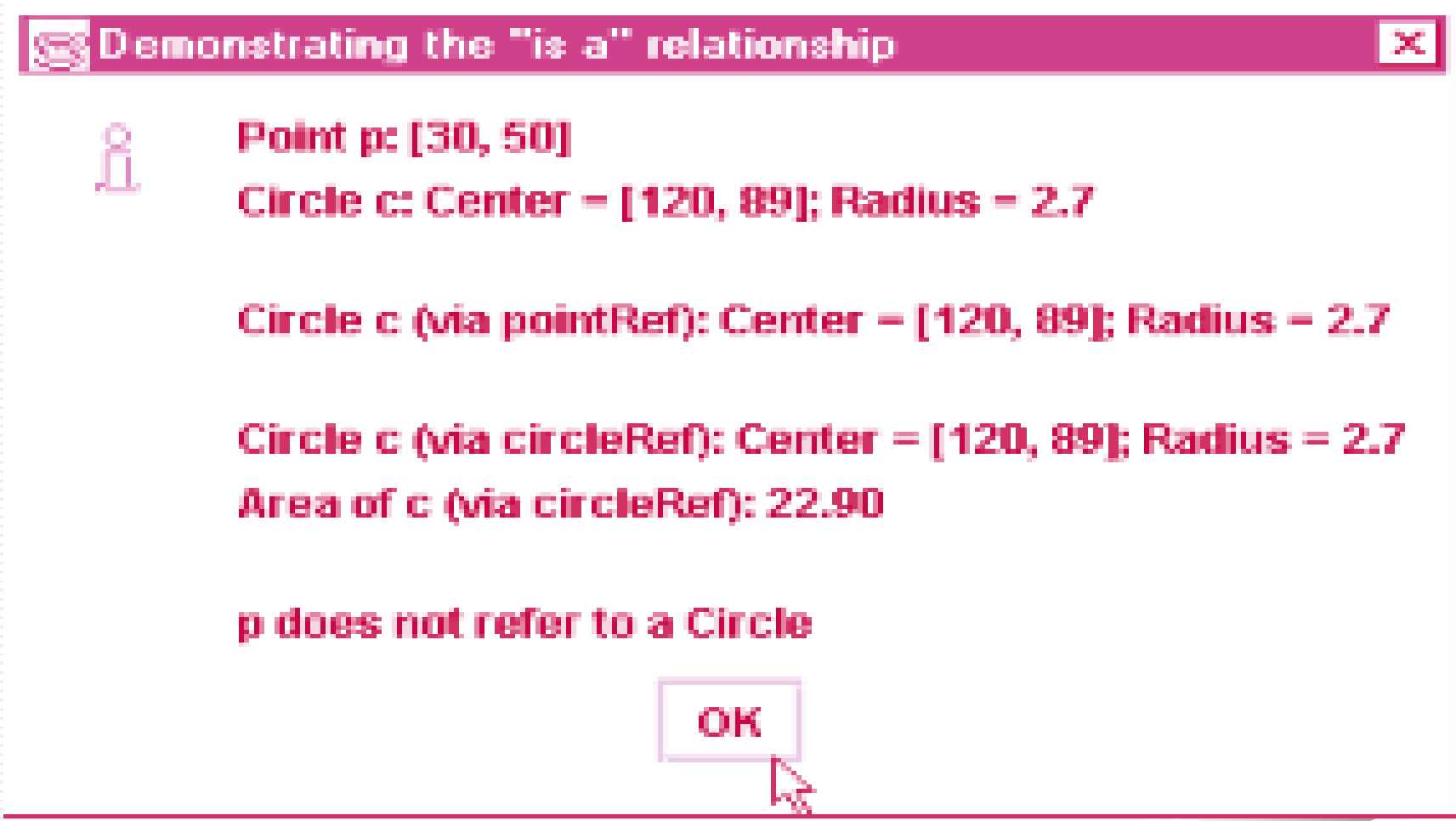
```
31    // get y coordinate
32    public int getY() { return y; }
33
34    // convert the point into a String representation
35    public String toString()
36       { return "[" + x + ", " + y + "]"; }
37 }
38 // Fig. 27.3: Circle.java
39 // Definition of class Circle
40
41 public class Circle extends Point {  // inherits from Point
42    protected double radius;
43
44    // No-argument constructor
45    public Circle()
46    {
47       // implicit call to superclass constructor occurs here
48       setRadius( 0 );
49    }
50
51    // Constructor
52    public Circle( double r, int a, int b )
53    {
54       super( a, b );  // call to superclass constructor
55       setRadius( r );
56    }
57
58    // Set radius of Circle
59    public void setRadius( double r )
60       { radius = ( r >= 0.0 ? r : 0.0 ); }
```

```
61
62      // Get radius of Circle
63      public double getRadius() { return radius; }
64
65      // Calculate area of Circle
66      public double area() { return Math.PI * radius * radius; }
67
68      // convert the Circle to a String
69      public String toString()
70      {
71         return "Center = " + "[" + x + ", " + y + "]" +
72                "; Radius = " + radius;
73      }
```

```
75 // Fig. 27.3: InheritanceTest.java
76 // Demonstrating the "is a" relationship
77 import java.text.DecimalFormat;
78 import javax.swing.JOptionPane;
79
80 public class InheritanceTest {
81    public static void main( String args[] )
82    {
83       Point pointRef, p;
84       Circle circleRef, c;
85       String output;
86
87       p = new Point( 30, 50 );
88       c = new Circle( 2.7, 120, 89 );
89
90       output = "Point p: " + p.toString() +
91                "\nCircle c: " + c.toString();
92
93       // use the "is a" relationship to refer to a Circle
94       // with a Point reference
95       pointRef = c;   // assign Circle to pointRef
96
97       output += "\n\nCircle c (via pointRef): " +
98                 pointRef.toString();
99
100      // Use downcasting (casting a superclass reference to a
101      // subclass data type) to assign pointRef to circleRef
102      circleRef = (Circle) pointRef;
103
104      output += "\n\nCircle c (via circleRef): " +
105                circleRef.toString();
```

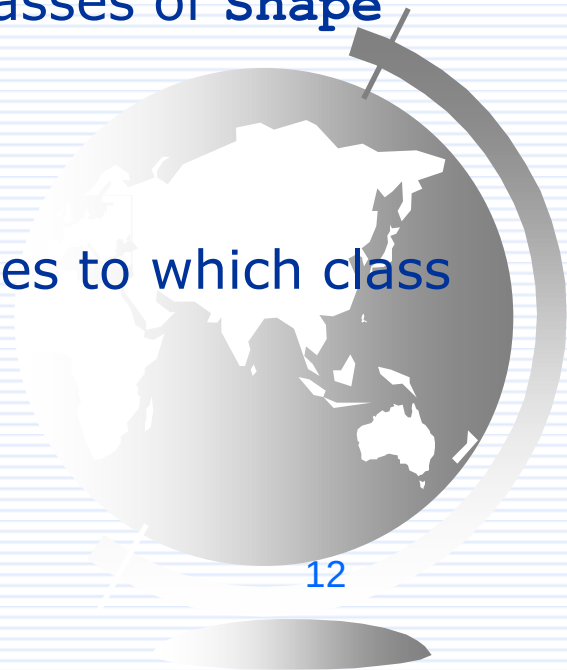
```
106
107       DecimalFormat precision2 = new DecimalFormat( "0.00" );
108       output += "\nArea of c (via circleRef): " +
109                 precision2.format( circleRef.area() );
110
111       // Attempt to refer to Point object
112       // with Circle reference
113       if ( p instanceof Circle ) {
114          circleRef = (Circle) p;  // line 40 in Test.java
115          output += "\n\ncast successful";
116       }
117       else
118          output += "\n\np does not refer to a Circle";
119
120       JOptionPane.showMessageDialog( null, output,
121          "Demonstrating the \"is a\" relationship",
122          JOptionPane.INFORMATION_MESSAGE );
123
124       System.exit( 0 );
125    }
126 }
```

# Kết quả ví dụ kế thừa



# Kết buộc phương thức động

- ✓ Kết buộc phương thức động
  - At execution time, method calls routed to appropriate version
    - Method called for appropriate class
- ✓ Example
  - **`Triangle`**, **`Circle`**, and **`Square`** all subclasses of **`Shape`**
    - Each has an overridden **`draw`** method
  - Call **`draw`** using superclass references
    - At execution time, program determines to which class the reference is actually pointing
    - Calls appropriate **`draw`** method

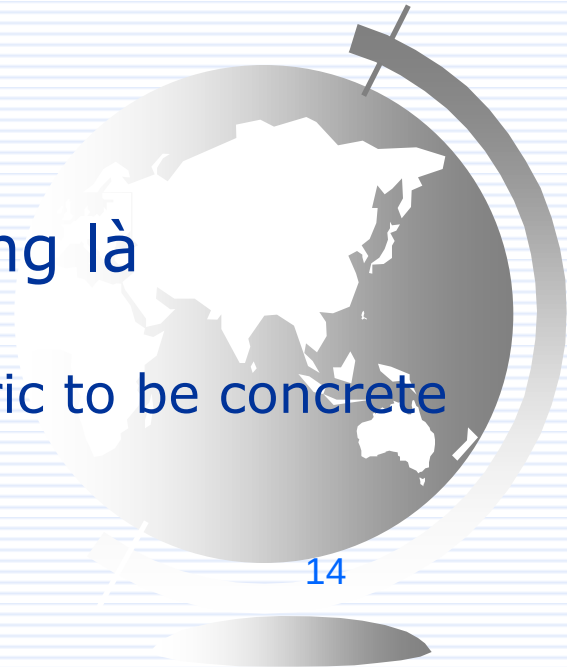


# Phương thức và lớp final

- ✓ Biến **final**
  - Indicates they cannot be modified after declaration
  - Must be initialized when declared
- ✓ Phương thức **final**
  - Cannot be overridden in a subclass
  - **static** and **private** methods are implicitly **final**
  - Program can inline **final** methods
    - Actually inserts method code at method call locations
    - Improves program performance
- ✓ Lớp **final**
  - Cannot be a superclass (cannot inherit from it)
  - All methods in class are implicitly **final**

# Lớp trừu tượng và Lớp cụ thể

- ✓ Lớp trừu tượng (**abstract class**)
  - Sole purpose is to be a superclass
    - Other classes inherit from it
  - Cannot instantiate objects of an abstract class
    - Can still define constructor
  - Too generic to define real objects
  - Declare class with keyword **abstract**
- ✓ Lớp cụ thể (**Concrete class**)
  - Can instantiate objects
  - Provide specifics
- ✓ Hầu hết các lớp quá tổng quát thường là **abstract**
    - **TwoDimensionalShape** - too generic to be concrete

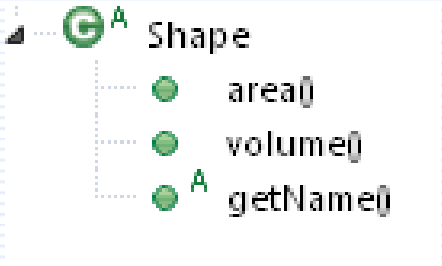


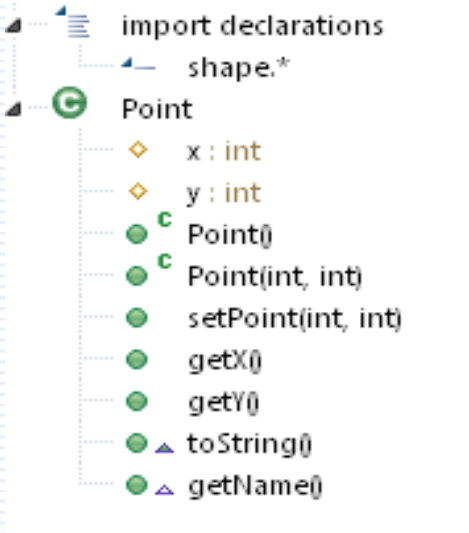
# Đa hình trong Java

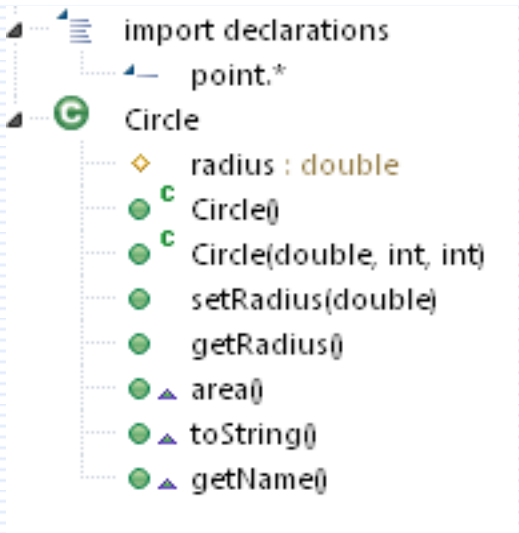
- ✓ Tính đa hình
  - Gọi một phương thức có thể sinh ra các hành động khác nhau, phụ thuộc vào đối tượng gọi.
- ✓ Tham chiếu
  - Can create references to **`abstract`** classes
    - Cannot instantiate objects of **`abstract`** classes
- ✓ **`Phương thức abstract`**
  - Dùng từ khóa **`abstract`**
    - Any class with an **`abstract`** method must be **`abstract`**
  - **`abstract`** methods must be overridden in subclass
    - Otherwise, subclass must be **`abstract`**

# Đa hình trong Java

- ✓ **Sơ đồ lớp một ví dụ đa hình**




```
1  // Fig. 27.4: Shape.java
2  // Definition of abstract base class Shape
3
4  public abstract class Shape {
5     public double area() { return 0.0; }
6     public double volume() { return 0.0; }
7     public abstract String getName();
```

```
 9 // Fig. 27.4: Point.java
10 // Definition of class Point
11
12 public class Point extends Shape {
13    protected int x, y; // coordinates of the Point
14
15    // no-argument constructor
16    public Point() { setPoint( 0, 0 ); }
17
18    // constructor
19    public Point( int a, int b ) { setPoint( a, b ); }
20
21    // Set x and y coordinates of Point
22    public void setPoint( int a, int b )
23    {
24       x = a;
25       y = b;
26    }
27
28    // get x coordinate
29    public int getX() { return x; }
30
31    // get y coordinate
32    public int getY() { return y; }
33
34    // convert the point into a String representation
35    public String toString()
36       { return "[" + x + ", " + y + "]"; }
37
38    // return the class name
39    public String getName() { return "Point"; }
40 }
```

```
41 // Fig. 27.10: Circle.java
42 // Definition of class Circle
43
44 public class Circle extends Point {  // inherits from Point
45    protected double radius;
46
47    // no-argument constructor
48    public Circle()
49    {
50       // implicit call to superclass constructor here
51       setRadius( 0 );
52    }
53
54    // Constructor
55    public Circle( double r, int a, int b )
56    {
57       super( a, b );  // call the superclass constructor
58       setRadius( r );
59    }
60
61    // Set radius of Circle
62    public void setRadius( double r )
63       { radius = ( r >= 0 ? r : 0 ); }
64
65    // Get radius of Circle
66    public double getRadius() { return radius; }
67
68    // Calculate area of Circle
69    public double area() { return Math.PI * radius * radius; }
70
```

```
71    // convert the Circle to a String
72    public String toString()
73       { return "Center = " + super.toString() +
74                "; Radius = " + radius; }
75
76    // return the class name
77    public String getName() { return "Circle"; }
78 }
```

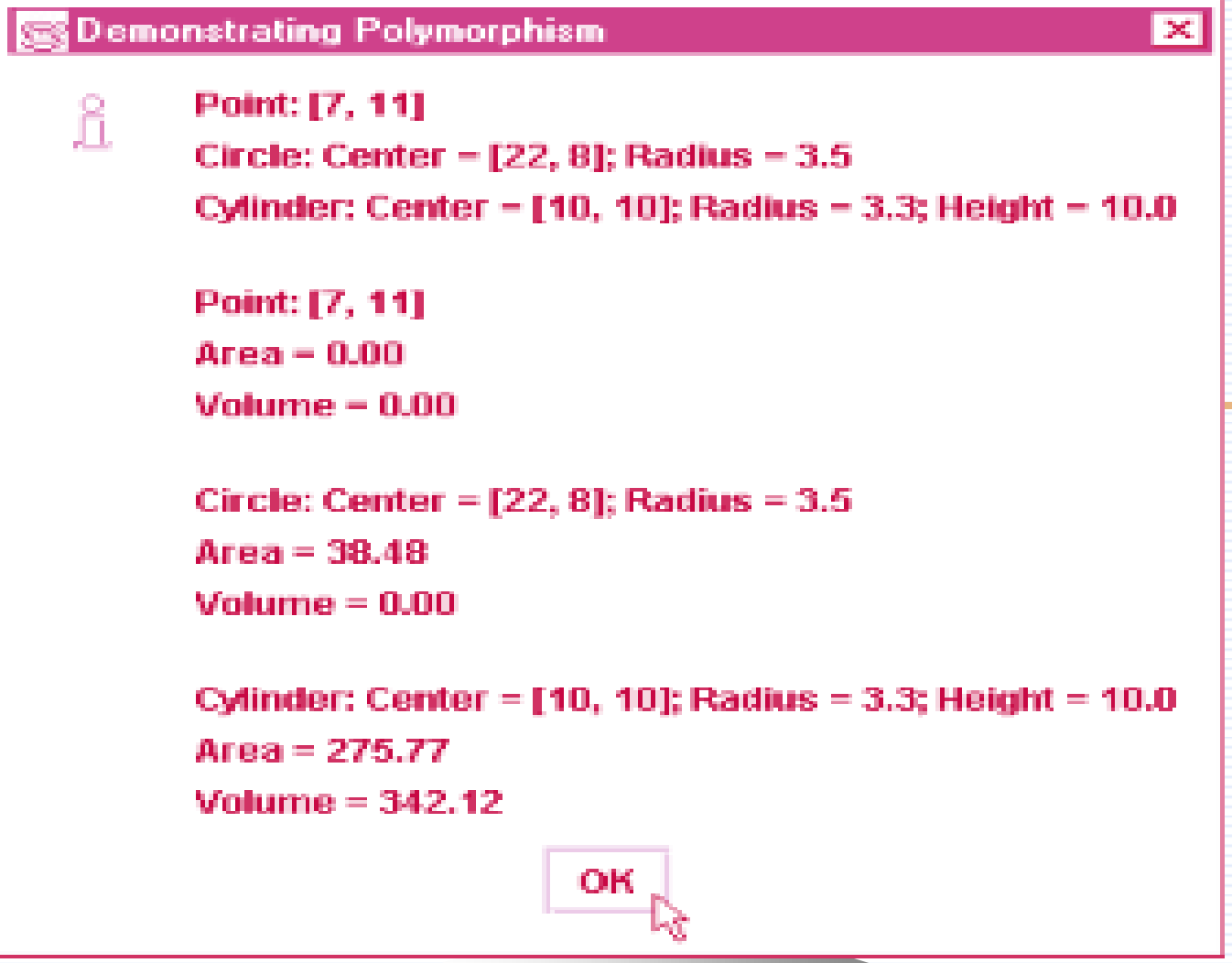
```
79 // Fig. 27.10: Cylinder.java
80 // Definition of class Cylinder
81
82 public class Cylinder extends Circle {
83    protected double height;  // height of Cylinder
84
85    // no-argument constructor
86    public Cylinder()
87    {
88       // implicit call to superclass constructor here
89       setHeight( 0 );
90    }
91
92    // constructor
93    public Cylinder( double h, double r, int a, int b )
94    {
95       super( r, a, b );   // call superclass constructor
96       setHeight( h );
97    }
98
99    // Set height of Cylinder
100   public void setHeight( double h )
101      { height = ( h >= 0 ? h : 0 ); }
102
103   // Get height of Cylinder
104   public double getHeight() { return height; }
105
106   // Calculate area of Cylinder (i.e., surface area)
107   public double area()
108   {
109      return 2 * super.area() +
110             2 * Math.PI * radius * height;
```

```
111    }
112
113    // Calculate volume of Cylinder
114    public double volume() { return super.area() * height; }
115
116    // Convert a Cylinder to a String
117    public String toString()
118       { return super.toString() + "; Height = " + height; }
119
120    // Return the class name
121    public String getName() { return "Cylinder"; }
122  }
```

```
123// Fig. 27.10: Test.java
124// Driver for point, circle, cylinder hierarchy
125import javax.swing.JOptionPane;
126import java.text.DecimalFormat;
127
128public class Test {
129   public static void main( String args[] )
130   {
131      Point point = new Point( 7, 11 );
132      Circle circle = new Circle( 3.5, 22, 8 );
133      Cylinder cylinder = new Cylinder( 10, 3.3, 10, 10 );
134
135      Shape arrayOfShapes[];
136
137      arrayOfShapes = new Shape[ 3 ];
138
139      // aim arrayOfShapes[0] at subclass Point object
140      arrayOfShapes[ 0 ] = point;
141
142      // aim arrayOfShapes[1] at subclass Circle object
143      arrayOfShapes[ 1 ] = circle;
144
145      // aim arrayOfShapes[2] at subclass Cylinder object
146      arrayOfShapes[ 2 ] = cylinder;
147
148      String output =
149         point.getName() + ": " + point.toString() + "\n" +
150         circle.getName() + ": " + circle.toString() + "\n" +
```

```
151         cylinder.getName() + ": " + cylinder.toString();
152
153      DecimalFormat precision2 = new DecimalFormat( "0.00" );
154
155      // Loop through arrayOfShapes and print the name,
156      // area, and volume of each object.
157      for ( int i = 0; i < arrayOfShapes.length; i++ ) {
158         output += "\n\n" +
159            arrayOfShapes[ i ].getName() + ": " +
160            arrayOfShapes[ i ].toString() +
161            "\nArea = " +
162            precision2.format( arrayOfShapes[ i ].area() ) +
163            "\nVolume = " +
164            precision2.format( arrayOfShapes[ i ].volume() );
165      }
166
167      JOptionPane.showMessageDialog( null, output,
168         "Demonstrating Polymorphism",
169         JOptionPane.INFORMATION_MESSAGE );
170
171      System.exit( 0 );
172   }
173}
```

# Kết quả ví dụ

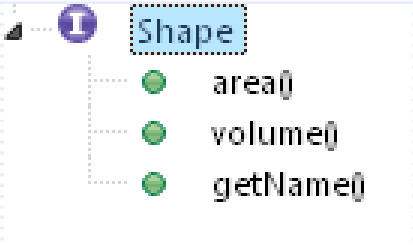

Demonstrating Polymorphism
Point: [7, 11]
Circle: Center = [22, 8]; Radius = 3.5
Cylinder: Center = [10, 10]; Radius = 3.3; Height = 10.0
Point: [7, 11]
Area = 0.00
Volume = 0.00
Circle: Center = [22, 8]; Radius = 3.5
Area = 38.48
Volume = 0.00
Cylinder: Center = [10, 10]; Radius = 3.3; Height = 10.0
Area = 275.77
Volume = 342.12
OK

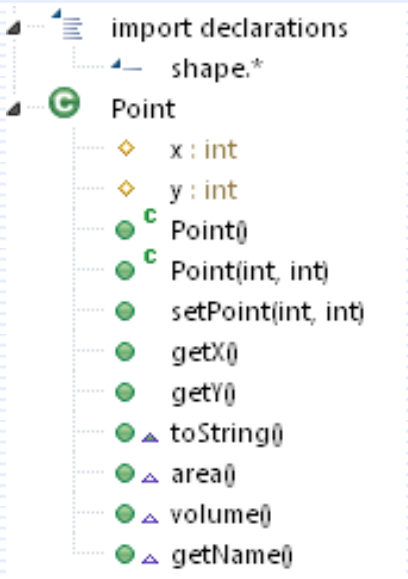
# Giao diện trong Java

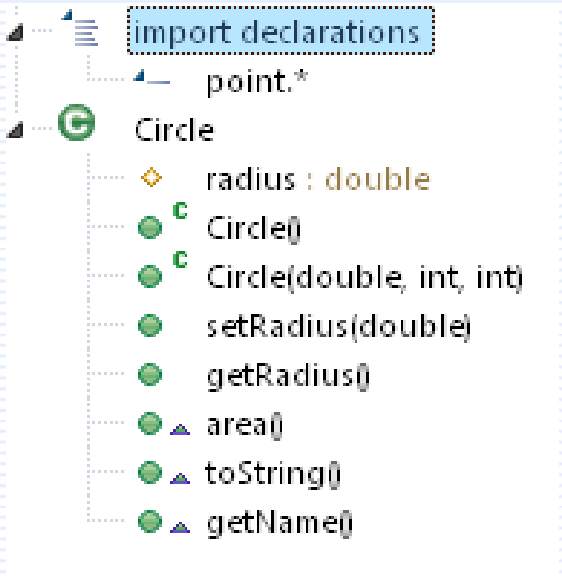
- ✓ Giao diện
  - Dùng từ khóa **`interface`**
  - Có tập phương thức **`public abstract`**
- ✓ Sử dụng giao diện
  - Dùng từ khóa **`implements`**
    - Nhiều interfaces thì sử dụng dấu phẩy để ngăn cách
  - Phải định nghĩa tất cả phương thức **`abstrac`** trong interface
    - Cùng số lượng đối số, kiểu dữ liệu trả về.

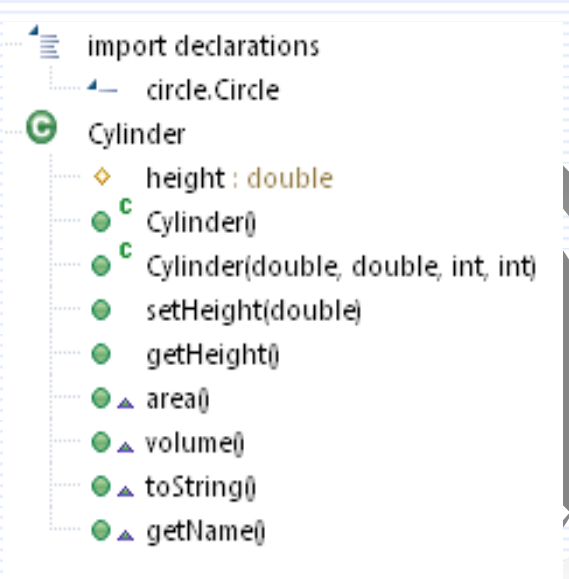
# Giao diện trong Java

## ✓ Sơ đồ lớp một ví dụ đa hình



```
1  // Fig. 27.5: Shape.java
2  // Definition of interface Shape
3
4  public interface Shape {
5     public abstract double area();
6     public abstract double volume();
7     public abstract String getName();
8  }
```

```
 9 // Fig. 27.5: Point.java
10 // Definition of class Point
11
12 public class Point extends Object implements Shape {
13    protected int x, y; // coordinates of the Point
14
15    // no-argument constructor
16    public Point() { setPoint( 0, 0 ); }
17
18    // constructor
19    public Point( int a, int b ) { setPoint( a, b ); }
20
21    // Set x and y coordinates of Point
22    public void setPoint( int a, int b )
23    {
24       x = a;
25       y = b;
26    }
27
28    // get x coordinate
29    public int getX() { return x; }
30
31    // get y coordinate
32    public int getY() { return y; }
33
34    // convert the point into a String representation
35    public String toString()
36       { return "[" + x + ", " + y + "]"; }
37
38    // return the area
39    public double area() { return 0.0; }
40
```

```
41     // return the volume
42     public double volume() { return 0.0; }
43
44     // return the class name
45     public String getName() { return "Point"; }
46 }
```

```java
// Fig. 27.5: Circle.java
// Definition of class Circle

public class Circle extends Point {  // inherits from Point
   protected double radius;

   // no-argument constructor
   public Circle()
   {
      // implicit call to superclass constructor here
      setRadius( 0 );
   }

   // Constructor
   public Circle( double r, int a, int b )
   {
      super( a, b );  // call the superclass constructor
      setRadius( r );
   }

   // Set radius of Circle
   public void setRadius( double r )
      { radius = ( r >= 0 ? r : 0 ); }

   // Get radius of Circle
   public double getRadius() { return radius; }

   // Calculate area of Circle
   public double area() { return Math.PI * radius * radius; }

```

```
31    // convert the Circle to a String
32    public String toString()
33       { return "Center = " + super.toString() +
34                "; Radius = " + radius; }
35
36    // return the class name
37    public String getName() { return "Circle"; }
```

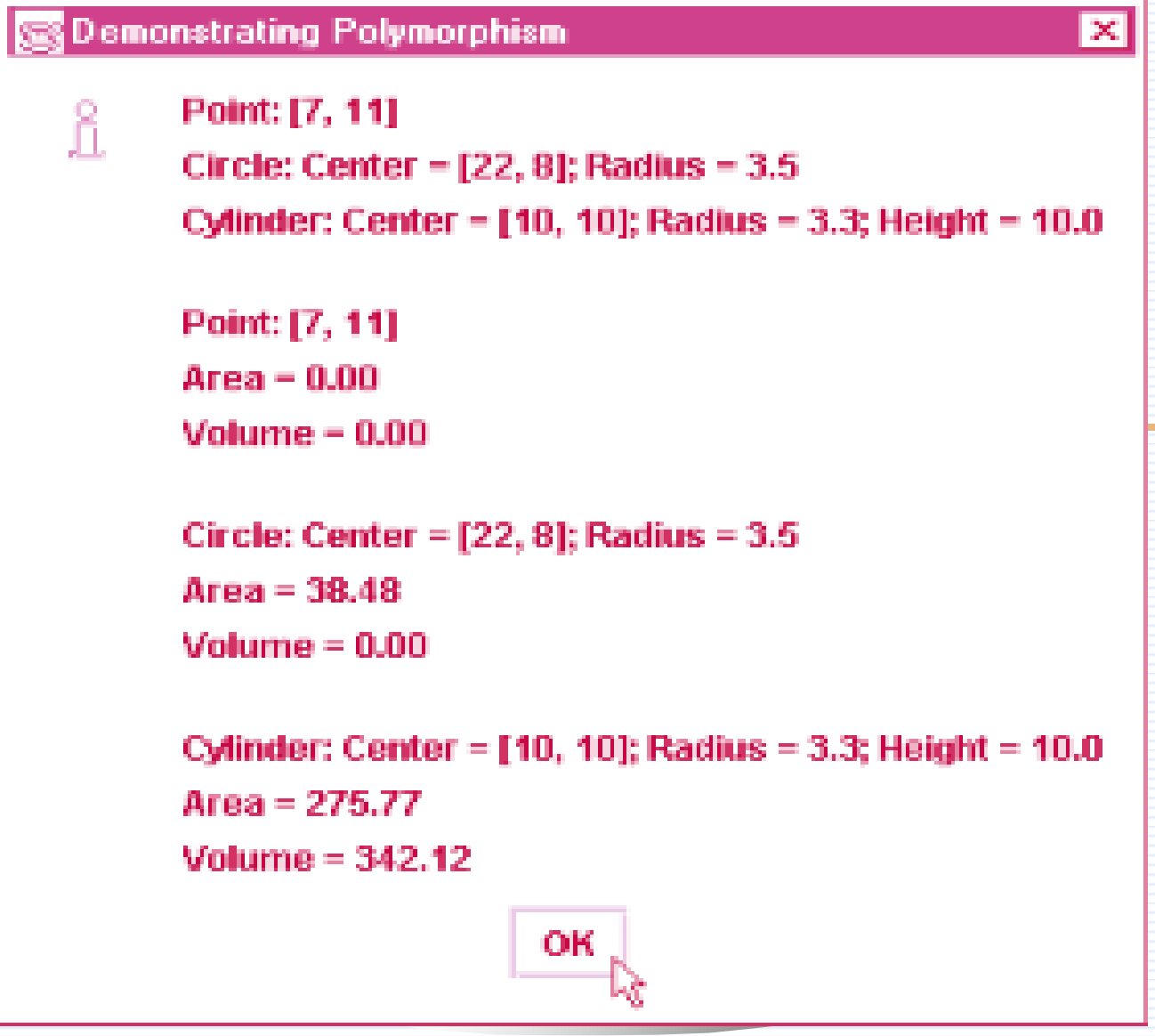
```
39 // Fig. 27.5: Cylinder.java
40 // Definition of class Cylinder
41
42 public class Cylinder extends Circle {
43    protected double height;  // height of Cylinder
44
45    // no-argument constructor
46    public Cylinder()
47    {
48       // implicit call to superclass constructor here
49       setHeight( 0 );
50    }
51
52    // constructor
53    public Cylinder( double h, double r, int a, int b )
54    {
55       super( r, a, b );   // call superclass constructor
56       setHeight( h );
57    }
58
59    // Set height of Cylinder
60    public void setHeight( double h )
61       { height = ( h >= 0 ? h : 0 ); }
62
63    // Get height of Cylinder
64    public double getHeight() { return height; }
65
66    // Calculate area of Cylinder (i.e., surface area)
67    public double area()
68    {
69       return 2 * super.area() +
70              2 * Math.PI * radius * height;
```

```
71    }
72
73    // Calculate volume of Cylinder
74    public double volume() { return super.area() * height; }
75
76    // Convert a Cylinder to a String
77    public String toString()
78       { return super.toString() + "; Height = " + height; }
79
80    // Return the class name
81    public String getName() { return "Cylinder"; }
```

```
83 // Fig. 27.5: Test.java
84 // Driver for point, circle, cylinder hierarchy
85 import javax.swing.JOptionPane;
86 import java.text.DecimalFormat;
87
88 public class Test {
89    public static void main( String args[] )
90    {
91       Point point = new Point( 7, 11 );
92       Circle circle = new Circle( 3.5, 22, 8 );
93       Cylinder cylinder = new Cylinder( 10, 3.3, 10, 10 );
94
95       Shape arrayOfShapes[];
96
97       arrayOfShapes = new Shape[ 3 ];
98
99       // aim arrayOfShapes[0] at subclass Point object
100      arrayOfShapes[ 0 ] = point;
101
102      // aim arrayOfShapes[1] at subclass Circle object
103      arrayOfShapes[ 1 ] = circle;
104
105      // aim arrayOfShapes[2] at subclass Cylinder object
106      arrayOfShapes[ 2 ] = cylinder;
107
108      String output =
109         point.getName() + ": " + point.toString() + "\n" +
110         circle.getName() + ": " + circle.toString() + "\n" +
```

```
111         cylinder.getName() + ": " + cylinder.toString();
112
113      DecimalFormat precision2 = new DecimalFormat( "#0.00" );
114
115      // Loop through arrayOfShapes and print the name,
116      // area, and volume of each object.
117      for ( int i = 0; i < arrayOfShapes.length; i++ ) {
118         output += "\n\n" +
119            arrayOfShapes[ i ].getName() + ": " +
120            arrayOfShapes[ i ].toString() +
121            "\nArea = " +
122            precision2.format( arrayOfShapes[ i ].area() ) +
123            "\nVolume = " +
124            precision2.format( arrayOfShapes[ i ].volume() );
125      }
126
127      JOptionPane.showMessageDialog( null, output,
128         "Demonstrating Polymorphism",
129         JOptionPane.INFORMATION_MESSAGE );
130
131      System.exit( 0 );
132   }
133}
```

# Kết quả ví dụ


Demonstrating Polymorphism
Point: [7, 11]
Circle: Center = [22, 8]; Radius = 3.5
Cylinder: Center = [10, 10]; Radius = 3.3; Height = 10.0
Point: [7, 11]
Area = 0.00
Volume = 0.00
Circle: Center = [22, 8]; Radius = 3.5
Area = 38.48
Volume = 0.00
Cylinder: Center = [10, 10]; Radius = 3.3; Height = 10.0
Area = 275.77
Volume = 342.12
OK

# Bài tập

- Xây dựng chương trình gồm các lớp tam giác đều, hình chữ nhật, hình tròn có chung thể hiện phương thức Area() – Tính diện tích.
  1. Dùng lớp asbtract
  2. Dùng interface