

# Ngôn ngữ lập trình Java (Java Programming Language)

---

**Môn học: Ngôn ngữ lập trình Java**

**Khoa: CNPM, Trường Đại học CNTT – ĐHQG TpHCM**

**GV: Th.S. Huỳnh Tuấn Anh**

**Email: [anhht@uit.edu.vn](mailto:anhht@uit.edu.vn)**

# Nội dung môn học

---

Chương 1: Giới thiệu về NNLT Java

Chương 2: Hướng đối tượng trong Java

Chương 3: Các lớp tiện ích trong Java

Chương 4: Quản lý Exception

Chương 5: Nhập xuất

Chương 6: Xử lý đa luồng

Chương 7: Kết nối và thao tác CSDL với JDBC

Chương 8: Lập trình GUI với AWT & Swing

## Chương 2

---

# ĐẶC ĐIỂM CƠ BẢN VỀ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG TRONG JAVA

# Các khái niệm cơ bản

❖ **Đối tượng (object):** trong thế giới thực khái niệm đối tượng có thể xem như một thực thể: người, vật, bảng dữ liệu,...

- ✓ Đối tượng giúp hiểu rõ thế giới thực
- ✓ Cơ sở cho việc cài đặt trên máy tính
- ✓ Mỗi đối tượng có định danh, thuộc tính, hành vi

**Ví dụ:** đối tượng sinh viên

MSSV: "TH0701001"; Tên sinh viên: "Nguyễn Văn A"

❖ **Hệ thống các đối tượng:** là 1 tập hợp các đối tượng

- ✓ Mỗi đối tượng đảm trách 1 công việc
- ✓ Các đối tượng có thể quan hệ với nhau
- ✓ Các đối tượng có thể trao đổi thông tin với nhau
- ✓ Các đối tượng có thể xử lý song song, hay phân tán

# Các khái niệm cơ bản

---

❖ **Lớp (class):** là khuôn mẫu (template) để sinh ra đối tượng. Lớp là sự trừu tượng hóa của tập các đối tượng có các thuộc tính, hành vi tương tự nhau, và được gom chung lại thành 1 lớp.

**Ví dụ:** lớp các đối tượng ***Sinhviên***

✓ Sinh viên “Nguyễn Văn A”, mã số TH0701001 → 1 đối tượng thuộc lớp ***Sinhviên***

✓ Sinh viên “Nguyễn Văn B”, mã số TH0701002 → là 1 đối tượng thuộc lớp ***Sinhviên***

❖ **Đối tượng (object) của lớp:** một đối tượng cụ thể thuộc 1 lớp là 1 thể hiện cụ thể của 1 lớp đó.

# Lớp và đối tượng trong java

---

## ❖ Khai báo lớp

```
class <ClassName>  
{  
    <danh sách thuộc tính>  
    <các khởi tạo>  
    <danh sách các phương thức>  
}
```

# Lớp và đối tượng trong java

❖ **Thuộc tính:** các đặc điểm mang giá trị của đối tượng, là vùng dữ liệu được khai báo bên trong lớp

```
class <ClassName>    {  
    <Tiền tố> <kiểu dữ liệu> <tên thuộc tính>;  
}
```

Kiểm soát truy cập đối với thuộc tính

- \* **public:** có thể truy xuất từ bất kỳ 1 lớp khác.
- \* **protected:** có thể truy xuất được từ những lớp con.
- \* **private:** không thể truy xuất từ 1 lớp khác.
- \* **static:** dùng chung cho mọi thể hiện của lớp.
- \* **final:** hằng
- \* **default:** (không phải từ khóa) có thể truy cập từ các class trong cùng gói

# Lớp và đối tượng trong java

❖ **Phương thức:** chức năng xử lý, hành vi của các đối tượng.

```
class <ClassName>      {  
    ...  
    <Tiền tố> <kiểu trả về> <tên phương thức>(<các đối số>){  
        ...  
    }  
}
```



# Lớp và đối tượng trong java

---

- \* **public**: có thể truy cập được từ bên ngoài lớp khai báo.
- \* **protected**: có thể truy cập được từ lớp khai báo và các lớp dẫn xuất (lớp con).
- \* **private**: chỉ được truy cập bên trong lớp khai báo.
- \* **static**: phương thức lớp dùng chung cho tất cả các thể hiện của lớp, có thể được thực hiện kể cả khi không có đối tượng của lớp
- \* **final**: không được khai báo chồng ở các lớp dẫn xuất.
- \* **abstract**: không có phần source code, sẽ được cài đặt trong các lớp dẫn xuất.
- \* **synchronized**: dùng để ngăn những tác động của các đối tượng khác lên đối tượng đang xét trong khi đang đồng bộ hóa. Dùng trong lập trình multithreads.

# Lớp và đối tượng trong java

*Ví dụ 1: class Sinhvien {*

*// Danh sách thuộc tính*

*String   maSv, tenSv, dcLienlac;*

*int       tuoi;*

*...*

*// Danh sách các khởi tạo*

*Sinhvien(){}*

*Sinhvien (...) { ...}*

*...*

*// Danh sách các phương thức*

*public void capnhatSV (...) {...}*

*public void xemThongTinSV() {...}*

*...*

*}*

# Lớp và đối tượng trong java

---

...

*// Tạo đối tượng mới thuộc lớp Sinhvien*

*Sinhvien sv = new Sinhvien();*

...

*// Gán giá trị cho thuộc tính của đối tượng*

*sv.maSv = "TH0601001";*

*sv.tenSv = "Nguyen Van A";*

*sv.tuoi = "20";*

*sv.dcLienlac = "KP6, Linh Trung, Thu Duc";*

...

*// Gọi thực hiện phương thức*

*sv.xemThongTinSV();*

# Lớp và đối tượng trong java

*Ví dụ 2:*

```
class Sinhvien {  
    // Danh sách thuộc tính  
    private String    maSv;  
    String    tenSv, dcLienlac;  
    int    tuoi;  
    ...  
}  
...  
Sinhvien sv = new Sinhvien();  
sv.maSv = "TH0601001"; /* Lỗi truy cập thuộc tính private từ  
                           bên ngoài lớp khai báo */  
Sv.tenSv = "Nguyen Van A";  
...  
}
```

# Lớp và đối tượng trong java

---

❖ **Khởi tạo (constructor):** là một loại phương thức đặc biệt của lớp, dùng để khởi tạo một đối tượng.

- ✓ Dùng để khởi tạo giá trị cho các thuộc tính của đối tượng.
- ✓ Cùng tên với lớp.
- ✓ Không có giá trị trả về.
- ✓ Tự động thi hành khi tạo ra đối tượng (new)
- ✓ Có thể có tham số hoặc không.

❖ **Lưu ý:** Mỗi lớp sẽ có 1 constructor mặc định (nếu ta không khai báo constructor nào). Ngược lại nếu ta có khai báo 1 constructor khác thì constructor mặc định chỉ dùng được khi khai báo tường minh.

# Lớp và đối tượng trong java

---

- *Ví dụ 1*

```
class Sinhvien
```

```
{
```

```
    ...
```

```
    // Không có định nghĩa constructor nào
```

```
}
```

```
    ...
```

```
// Dùng constructor mặc định
```

```
Sinhvien sv = new Sinhvien();
```

# Lớp và đối tượng trong java

## *Ví dụ 2:*

```
class Sinhvien
{
    ...
    // không có constructor mặc định
    Sinhvien(<các đối số>) {...}
}
...
Sinhvien sv = new Sinhvien();
// lỗi biên dịch
```

```
class Sinhvien
{
    ...
    // khai báo constructor mặc định
    Sinhvien(){}
    Sinhvien(<các đối số>) {...}
}
...
Sinhvien sv = new Sinhvien();
```

# Lớp và đối tượng trong java

❖ **Overloading method:** Việc khai báo trong một lớp nhiều phương thức có cùng tên nhưng khác tham số (khác kiểu dữ liệu, khác số lượng tham số) gọi là khai báo chồng phương thức.

Ví dụ: *class Sinhvien {*

*...*

*public void xemThongTinSV() {*

*...*

*}*

*public void xemThongTinSV(String psMaSv) {*

*...*

*}*

*}*



# Lớp và đối tượng trong java

❖ **Tham chiếu *this*:** là một biến ẩn tồn tại trong tất cả các lớp, ***this*** được sử dụng trong khi chạy và tham khảo đến bản thân lớp chứa nó.

**Ví dụ:**

```
class Sinhvien    {  
    String    maSv, tenSv, dcLienlac;  
    int      tuoi;  
    ...  
    public void xemThongTinSV() {  
        System.out.println(this.maSv);  
        System.out.println(this.tenSv);  
        ...  
    }  
}
```

# Tính đóng gói

❖ **Đóng gói:** nhóm những gì có liên quan với nhau vào thành một và có thể sử dụng một cái tên để gọi.

## ***Ví dụ:***

- ✓ Các phương thức đóng gói các câu lệnh.
- ✓ Đối tượng đóng gói dữ liệu và các hành vi/phương thức liên quan.

(Đối tượng = Dữ liệu + Hành vi/Phương thức)

# Tính đóng gói

❖ **Đóng gói:** dùng để che dấu một phần hoặc tất cả thông tin, chi tiết cài đặt bên trong với bên ngoài.

- Ví dụ: khai báo các lớp thuộc cùng gói trong java

*package <tên gói>; // khai báo trước khi khai báo lớp*

*class <tên lớp> {*

*...*

*}*

# Tính kế thừa (Inheritance)

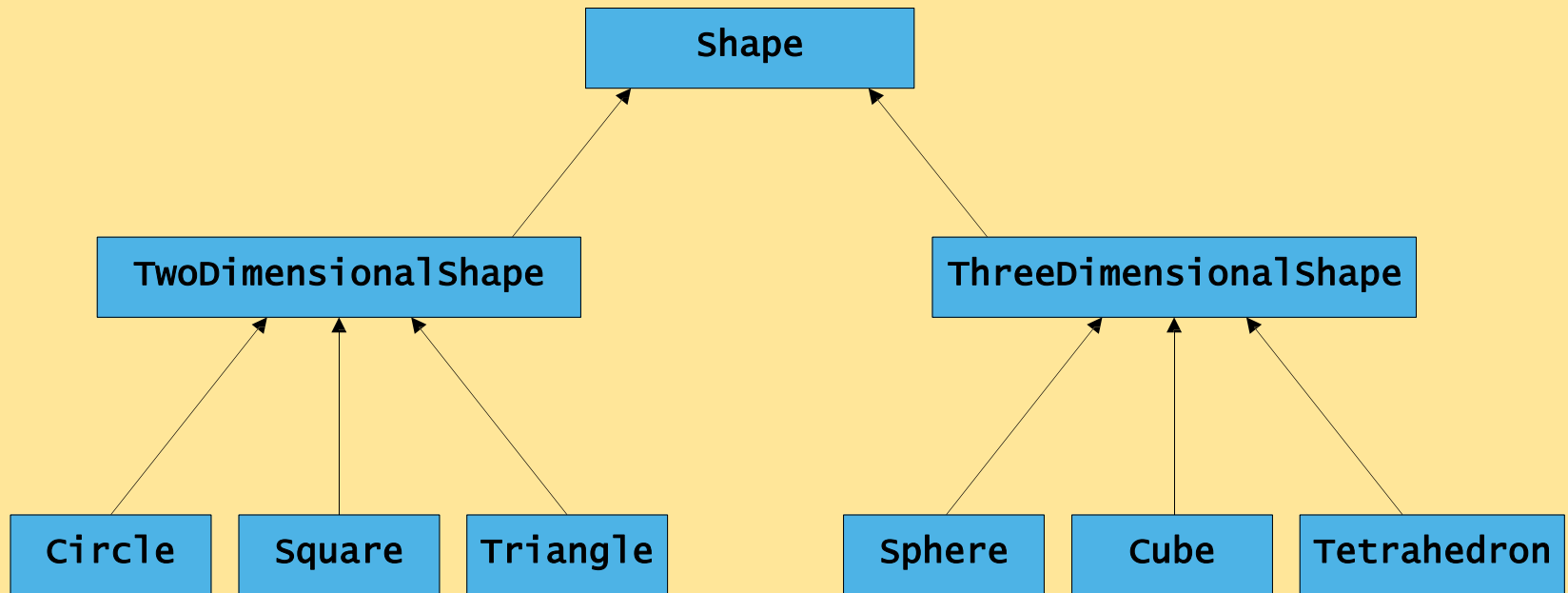
---

- ❑ Nâng cao khả năng tái sử dụng của phần mềm.
- ❑ Tạo lớp con mới (subclass) từ một lớp cha hiện có (super class)
  - Thừa hưởng các thuộc tính và phương thức đã có
  - Bổ sung, chi tiết hóa cho phù hợp với mục đích sử dụng mới
    - ✓ Thuộc tính: thêm mới
    - ✓ Phương thức: thêm mới hay hiệu chỉnh

# Tính kế thừa (Inheritance)

- ❑ Quan hệ “IS-A” và “HAS-A”
- ❑ “IS-A”
  - ❑ Một kiểu kế thừa
  - ❑ Đối tượng Subclass hành xử như đối tượng Superclass.
  - ❑ VD: **Circle** “IS A” **2D shape**; **Car** “IS A” **vehicle**. Thuộc tính, phương thức của **vehicle** cũng là thuộc tính và phương thức của **Car**.
- ❑ “HAS-A”: A “HAS A” B khi trong lớp A có một reference đến một instance của B.
  - ❑ Cấu thành (B cấu thành A).
  - ❑ Đối tượng chứa một hay nhiều đối tượng khác làm thành viên.
  - ❑ VD: Car “has a” SteeringWheel (vô lăng)

# Tính kế thừa (Inheritance)



# Tính kế thừa (Inheritance)

- ✓ **Lớp dẫn xuất hay lớp con (SubClass)**
- ✓ **Lớp cơ sở hay lớp cha (SuperClass)**
- ✓ Lớp con có thể kế thừa tất cả hay một phần các thành phần dữ liệu (thuộc tính), phương thức của lớp cha (public, protected, default)
- ✓ Dùng từ khóa ***extends***.

**Ví dụ:** *class nguoi { ...*

*}*

*class sinhvien **extends** nguoi { ...*

*}*

**Lưu ý:** default không phải là 1 từ khóa

# Tính kế thừa (Inheritance)

---

- ❑ Final class: Một lớp mà mô tả final không cho phép lớp khác kế thừa.
- ❑ Một lớp abstract không thể là lớp final.

```
public final class A {  
    ...  
}
```

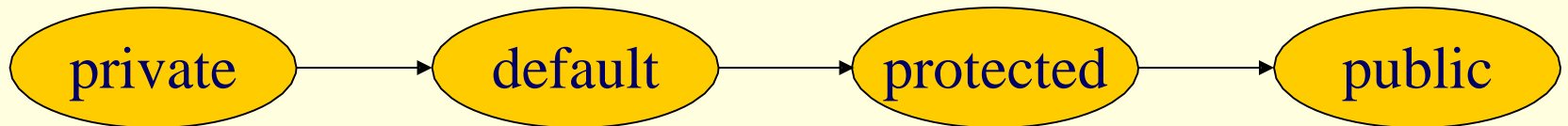


# Tính kế thừa (Inheritance)

---

## ❖ Overriding Method

- Được định nghĩa trong lớp con
- Có tên, kiểu trả về & các đối số giống với phương thức của lớp cha
- Có kiểu, phạm vi truy cập k0 “nhỏ hơn” phương thức trong lớp cha



# Tính kế thừa (Inheritance)

• Ví dụ: *class Hinhhoc { ...*

```
    public float tinhdientich() {  
        return 0;  
    }  
    ...
```

```
}
```

*class HinhVuong extends Hinhhoc {*

```
    private int canh;  
    public float tinhdientich() {  
        return canh*canh;
```

```
    }
```

```
    ...
```

```
}
```

Chỉ có thể **public** do phương thức *tinhdientich()* của lớp cha là **public**

# Tính kế thừa (Inheritance)

```
class HinhChuNhat extends HinhVuong {  
    private int cd;  
    private int cr;  
    public float tinhdientich() {  
        return cd*cr;  
    }  
    ...  
}
```

Chỉ có thể **public** do phương thức *tinhdientich()* của lớp cha là **public**

# Từ khóa super

---

- ❑ *Gọi constructor của lớp cha*
- ❑ *Nếu gọi tường minh thì phải là câu lệnh đầu tiên.*
- ❑ *Constructor cuối cùng được gọi là của lớp Object*
- ❑ *Truy cập đến thuộc tính bị che ở lớp cha.*

# Point.java

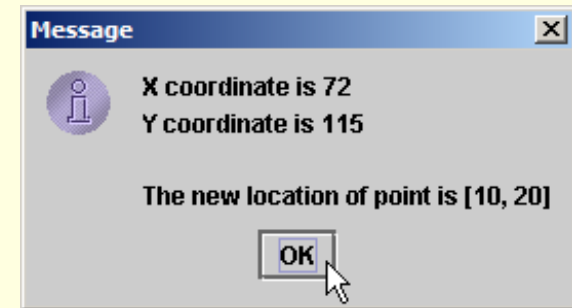
```
1  // Point.java
2  // Point class declaration represents an x-y coordinate pair.
3
4  public class Point {
5      private int x; // x part of coordinate pair
6      private int y; // y part of coordinate pair
7
8      // no-argument constructor
9      public Point()
10     {
11         // implicit call to Object constructor occurs here
12     }
13
14     // constructor
15     public Point( int xValue, int yValue )
16     {
17         // implicit call to Object constructor occurs here
18         x = xValue; // no need for validation
19         y = yValue; // no need for validation
20     }
21
22     // set x in coordinate pair
23     public void setX( int xValue )
24     {
25         x = xValue; // no need for validation
26     }
27
```

# Point.java

```
28      // return x from coordinate pair
29      public int getX()
30      {
31          return x;
32      }
33
34      // set y in coordinate pair
35      public void setY( int yValue )
36      {
37          y = yValue;  // no need for validation
38      }
39
40      // return y from coordinate pair
41      public int getY()
42      {
43          return y;
44      }
45
46      // return String representation of Point object
47      public String toString()
48      {
49          return "[" + x + ", " + y + "]";
50      }
51
52  } // end class Point
```

# PointTest.java

```
1  // PointTest.java
2  // Testing class Point.
3  import javax.swing.JOptionPane;
4
5  public class PointTest {
6
7      public static void main( String[] args )
8      {
9          Point point = new Point( 72, 115 ); // create Point object
10
11         // get point coordinates
12         String output = "X coordinate is " + point.getX() +
13             "\nY coordinate is " + point.getY();
14
15         point.setX( 10 ); // set x-coordinate
16         point.setY( 20 ); // set y-coordinate
17
18         // get String representation of new point value
19         output += "\n\nThe new location of point is " + point;
20
21         JOptionPane.showMessageDialog( null, output ); // display output
22
23         System.exit( 0 );
24
25     } // end main
26
27 } // end class PointTest
```



# Circle.java

```
1  // Circle.java
2  // Circle class contains x-y coordinate pair and radius.
3
4  public class Circle {
5      private int x;           // x-coordinate of Circle's center
6      private int y;           // y-coordinate of Circle's center
7      private double radius;   // Circle's radius
8
9      // no-argument constructor
10     public Circle()
11     {
12         // implicit call to Object constructor occurs here
13     }
14
15     // constructor
16     public Circle( int xValue, int yValue, double radiusValue )
17     {
18         // implicit call to Object constructor occurs here
19         x = xValue; // no need for validation
20         y = yValue; // no need for validation
21         setRadius( radiusValue );
22     }
23
24     // set x in coordinate pair
25     public void setX( int xValue )
26     {
27         x = xValue; // no need for validation
28     }
29
```



# Circle.java

```
30     // return x from coordinate pair
31     public int getX()
32     {
33         return x;
34     }
35
36     // set y in coordinate pair
37     public void setY( int yValue )
38     {
39         y = yValue;  // no need for validation
40     }
41
42     // return y from coordinate pair
43     public int getY()
44     {
45         return y;
46     }
47
48     // set radius
49     public void setRadius( double radiusValue )
50     {
51         radius = ( radiusValue < 0.0 ? 0.0 : radiusValue );
52     }
53
54     // return radius
55     public double getRadius()
56     {
57         return radius;
58     }
59
```

# Circle.java

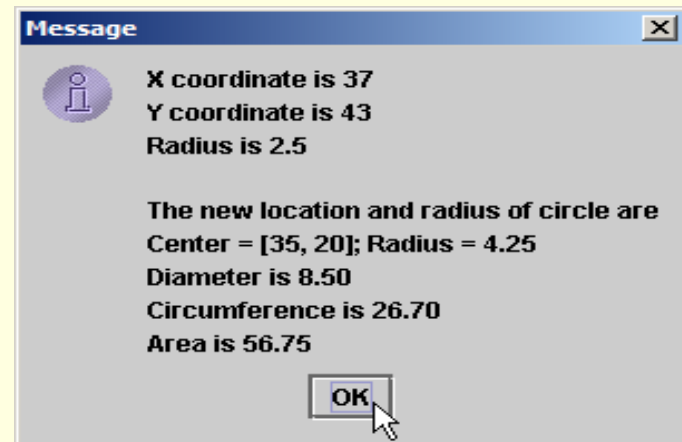
```
60     // calculate and return diameter
61     public double getDiameter()
62     {
63         return 2 * radius;
64     }
65
66     // calculate and return circumference
67     public double getCircumference()
68     {
69         return Math.PI * getDiameter();
70     }
71
72     // calculate and return area
73     public double getArea()
74     {
75         return Math.PI * radius * radius;
76     }
77
78     // return String representation of Circle object
79     public String toString()
80     {
81         return "Center = [" + x + ", " + y + "]; Radius = " + radius;
82     }
83
84 } // end class Circle
```

# CircleTest.java

```
1  // CircleTest.java
2  // Testing class Circle.
3  import java.text.DecimalFormat;
4  import javax.swing.JOptionPane;
5
6  public class CircleTest {
7
8      public static void main( String[] args )
9      {
10         Circle circle = new Circle( 37, 43, 2.5 ); // create Circle object
11
12         // get Circle's initial x-y coordinates and radius
13         String output = "X coordinate is " + circle.getX() +
14             "\nY coordinate is " + circle.getY() +
15             "\nRadius is " + circle.getRadius();
16
17         circle.setX( 35 );           // set new x-coordinate
18         circle.setY( 20 );           // set new y-coordinate
19         circle.setRadius( 4.25 );    // set new radius
20
21         // get String representation of new circle value
22         output += "\n\nThe new location and radius of circle are\n" +
23             circle.toString();
24
25         // format floating-point values with 2 digits of precision
26         DecimalFormat twoDigits = new DecimalFormat( "0.00" );
27
```

# CircleTest.java

```
28     // get Circle's diameter
29     output += "\nDiameter is " +
30         twoDigits.format( circle.getDiameter() );
31
32     // get Circle's circumference
33     output += "\nCircumference is " +
34         twoDigits.format( circle.getCircumference() );
35
36     // get Circle's area
37     output += "\nArea is " + twoDigits.format( circle.getArea() );
38
39     JOptionPane.showMessageDialog( null, output ); // display output
40
41     System.exit( 0 );
42
43     } // end main
44
45 } // end class CircleTest
```



# Circle2.java

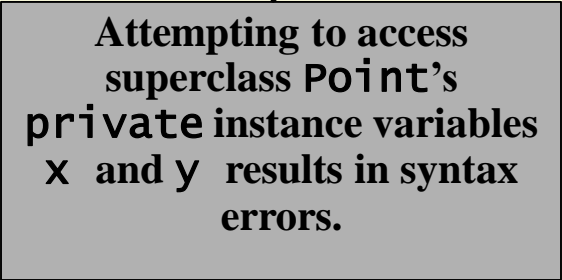
```
1  // Circle2.java
2  // Circle2 class inherits from Point.
3
4  public class Circle2 extends Point {
5      private double radius;  // Circle2's radius
6
7      // no-argument constructor
8      public Circle2()
9      {
10         // implicit call to Point constructor occurs here
11     }
12
13     // constructor
14     public Circle2( int xValue, int yValue, double radiusValue )
15     {
16         // implicit call to Point constructor occurs here
17         x = xValue;  // not allowed: x private in Point
18         y = yValue;  // not allowed: y private in Point
19         setRadius( radiusValue );
20     }
21
22     // set radius
23     public void setRadius( double radiusValue )
24     {
25         radius = ( radiusValue < 0.0 ? 0.0 : radiusValue );
26     }
27
```

# Circle2.java

```
34     // calculate and return diameter
35     public double getDiameter()
36     {
37         return 2 * radius;
38     }
39
40     // calculate and return circumference
41     public double getCircumference()
42     {
43         return Math.PI * getDiameter();
44     }
45
46     // calculate and return area
47     public double getArea()
48     {
49         return Math.PI * radius * radius;
50     }
51
52     // return String representation of Circle object
53     public String toString()
54     {
55         // use of x and y not allowed: x and y private in Point
56         return "Center = [" + x + ", " + y + "]; Radius = " + radius;
57     }
58 } //end class Circle2
59
```

# Circle2.java output

```
Circle2.java:17: x has private access in Point
    x = xValue; // not allowed: x private in Point
    ^
Circle2.java:18: y has private access in Point
    y = yValue; // not allowed: y private in Point
    ^
Circle2.java:56: x has private access in Point
    return "Center = [" + x + ", " + y + "]; Radius = " + radius;
                        ^
Circle2.java:56: y has private access in Point
    return "Center = [" + x + ", " + y + "]; Radius = " + radius;
                        ^
4 errors
```



**Attempting to access  
superclass `Point`'s  
private instance variables  
`x` and `y` results in syntax  
errors.**

# Point2.java

```
1  // Point2.java
2  // Point2 class declaration represents an x-y coordinate pair.
3
4  public class Point2 {
5      protected int x;  // x part of coordinate pair
6      protected int y;  // y part of coordinate pair
7
8      // no-argument constructor
9      public Point2()
10     {
11         // implicit call to Object constructor occurs here
12     }
13
14     // constructor
15     public Point2( int xValue, int yValue )
16     {
17         // implicit call to Object constructor occurs here
18         x = xValue;  // no need for validation
19         y = yValue;  // no need for validation
20     }
21
22     // set x in coordinate pair
23     public void setX( int xValue )
24     {
25         x = xValue;  // no need for validation
26     }
27 }
```



# Point2.java

```
28     // return x from coordinate pair
29     public int getX()
30     {
31         return x;
32     }
33
34     // set y in coordinate pair
35     public void setY( int yValue )
36     {
37         y = yValue;  // no need for validation
38     }
39
40     // return y from coordinate pair
41     public int getY()
42     {
43         return y;
44     }
45
46     // return String representation of Point2 object
47     public String toString()
48     {
49         return "[" + x + ", " + y + "]";
50     }
51 } // end class Point2
52
```

# Circle3.java

```
1    // Circle3.java
2    // Circle3 class inherits from Point2 and has access to Point2
3    // protected members x and y.
4
5    public class Circle3 extends Point2 {
6        private double radius; // Circle3's radius
7
8        // no-argument constructor
9        public Circle3()
10       {
11           // implicit call to Point2 constructor occurs here
12       }
13
14       // constructor
15       public Circle3( int xValue, int yValue, double radiusValue )
16       {
17           // implicit call to Point2 constructor occurs here
18           x = xValue; // no need for validation
19           y = yValue; // no need for validation
20           setRadius( radiusValue );
21       }
22
23       // set radius
24       public void setRadius( double radiusValue )
25       {
26           radius = ( radiusValue < 0.0 ? 0.0 : radiusValue );
27       }
28
```

# Circle3.java

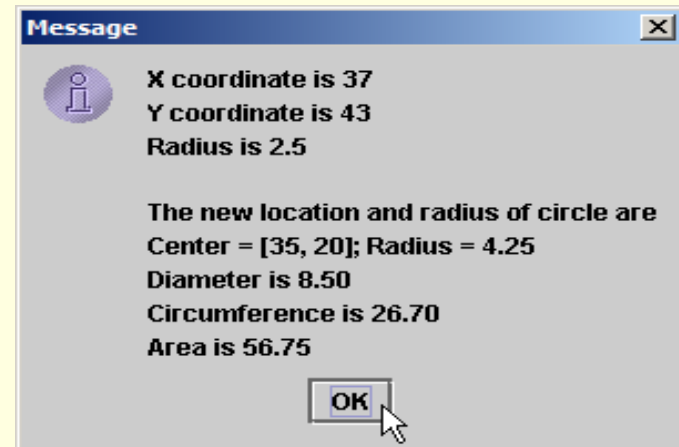
```
29     // return radius
30     public double getRadius()
31     {
32         return radius;
33     }
34
35     // calculate and return diameter
36     public double getDiameter()
37     {
38         return 2 * radius;
39     }
40
41     // calculate and return circumference
42     public double getCircumference()
43     {
44         return Math.PI * getDiameter();
45     }
46
47     // calculate and return area
48     public double getArea()
49     {
50         return Math.PI * radius * radius;
51     }
52
53     // return String representation of Circle3 object
54     public String toString()
55     {
56         return "Center = [" + x + ", " + y + "]; Radius = " + radius;
57     }
58 } // end class Circle3
```

# CircleTest3.java

```
1  // CircleTest3.java
2  // Testing class Circle3.
3  import java.text.DecimalFormat;
4  import javax.swing.JOptionPane;
5
6  public class CircleTest3 {
7
8      public static void main( String[] args )
9      {
10         // instantiate Circle object
11         Circle3 circle = new Circle3( 37, 43, 2.5 );
12
13         // get Circle3's initial x-y coordinates and radius
14         String output = "X coordinate is " + circle.getX() +
15             "\nY coordinate is " + circle.getY() +
16             "\nRadius is " + circle.getRadius();
17
18         circle.setX( 35 );           // set new x-coordinate
19         circle.setY( 20 );           // set new y-coordinate
20         circle.setRadius( 4.25 );    // set new radius
21
22         // get String representation of new circle value
23         output += "\n\nThe new location and radius of circle are\n" +
24             circle.toString();
25     }
```

# CircleTest3.java

```
26      // format floating-point values with 2 digits of precision
27      DecimalFormat twoDigits = new DecimalFormat( "0.00" );
28
29      // get Circle's diameter
30      output += "\nDiameter is " +
31          twoDigits.format( circle.getDiameter() );
32
33      // get Circle's circumference
34      output += "\nCircumference is " +
35          twoDigits.format( circle.getCircumference() );
36
37      // get Circle's area
38      output += "\nArea is " + twoDigits.format( circle.getArea() );
39
40      JOptionPane.showMessageDialog( null, output ); // display output
41
42      System.exit( 0 );
43
44      } // end method main
45
46      } // end class CircleTest3
```



# Quan hệ giữa superclass và subclass

- ❑ Dùng các thành phần protected
  - ❑ Ưu điểm:
    - subclasses can modify values directly
    - Slight increase in performance
      - Avoid set/get function call overhead
  - ❑ Hạn chế:
    - No validity checking
      - subclass can assign illegal value
    - Implementation dependent
      - subclass methods more likely dependent on superclass implementation
      - superclass implementation changes may result in subclass modifications
        - Fragile (brittle) software

# Point3.java

```
1      // Point3.java
2      // Point class declaration represents an x-y coordinate pair.
3
4      public class Point3 {
5          private int x;  // x part of coordinate pair
6          private int y;  // y part of coordinate pair
7
8          // no-argument constructor
9          public Point3()
10         {
11             // implicit call to Object constructor occurs here
12         }
13
14         // constructor
15         public Point3( int xValue, int yValue )
16         {
17             // implicit call to Object constructor occurs here
18             x = xValue;  // no need for validation
19             y = yValue;  // no need for validation
20         }
21
22         // set x in coordinate pair
23         public void setX( int xValue )
24         {
25             x = xValue;  // no need for validation
26         }
27     }
```

# Point3.java

```
28     // return x from coordinate pair
29     public int getX()
30     {
31         return x;
32     }
33
34     // set y in coordinate pair
35     public void setY( int yValue )
36     {
37         y = yValue;  // no need for validation
38     }
39
40     // return y from coordinate pair
41     public int getY()
42     {
43         return y;
44     }
45
46     // return String representation of Point3 object
47     public String toString()
48     {
49         return "[" + getX() + ", " + getY() + "]";
50     }
51
52 } // end class Point3
```



# Circle4.java

```
1      // Circle4.java
2      // Circle4 class inherits from Point3 and accesses Point3's
3      // private x and y via Point3's public methods.
4
5      public class Circle4 extends Point3 {
6
7          private double radius; // Circle4's radius
8
9          // no-argument constructor
10         public Circle4()
11         {
12             // implicit call to Point3 constructor occurs here
13         }
14
15         // constructor
16         public Circle4( int xValue, int yValue, double radiusValue )
17         {
18             super( xValue, yValue ); // call Point3 constructor explicitly
19             setRadius( radiusValue );
20         }
21
22         // set radius
23         public void setRadius( double radiusValue )
24         {
25             radius = ( radiusValue < 0.0 ? 0.0 : radiusValue );
26         }
27     }
```

# Circle4.java

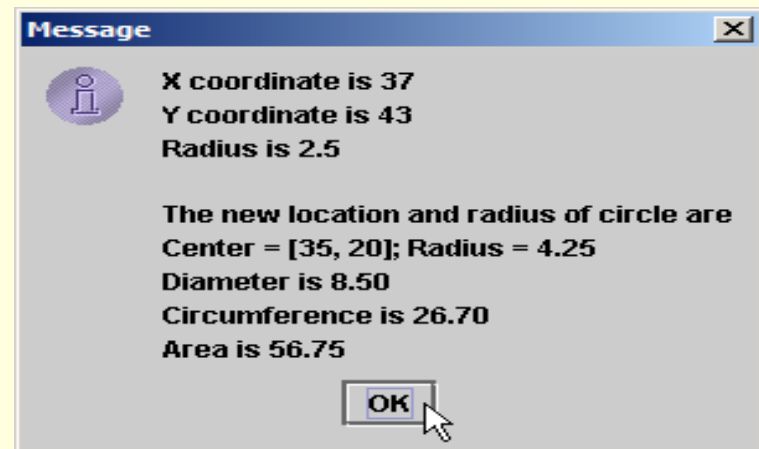
```
28     // return radius
29     public double getRadius()
30     {
31         return radius;
32     }
33
34     // calculate and return diameter
35     public double getDiameter()
36     {
37         return 2 * getRadius();
38     }
39
40     // calculate and return circumference
41     public double getCircumference()
42     {
43         return Math.PI * getDiameter();
44     }
45
46     // calculate and return area
47     public double getArea()
48     {
49         return Math.PI * getRadius() * getRadius();
50     }
51
52     // return String representation of Circle4 object
53     public String toString()
54     {
55         return "Center = " + super.toString() + "; Radius = " + getRadius();
56     }
57
58     } // end class Circle4
```

# CircleTest4.java

```
1      // CircleTest4.java
2      // Testing class Circle4.
3      import java.text.DecimalFormat;
4      import javax.swing.JOptionPane;
5
6      public class CircleTest4 {
7
8          public static void main( String[] args )
9          {
10             // instantiate Circle object
11             Circle4 circle = new Circle4( 37, 43, 2.5 );
12
13             // get Circle4's initial x-y coordinates and radius
14             String output = "X coordinate is " + circle.getX() +
15                 "\nY coordinate is " + circle.getY() +
16                 "\nRadius is " + circle.getRadius();
17
18             circle.setX( 35 );           // set new x-coordinate
19             circle.setY( 20 );           // set new y-coordinate
20             circle.setRadius( 4.25 );    // set new radius
21
22             // get String representation of new circle value
23             output += "\n\nThe new location and radius of circle are\n" +
24                 circle.toString();
25
```

# CircleTest4.java

```
26      // format floating-point values with 2 digits of precision
27      DecimalFormat twoDigits = new DecimalFormat( "0.00" );
28
29      // get Circle's diameter
30      output += "\nDiameter is " +
31          twoDigits.format( circle.getDiameter() );
32
33      // get Circle's circumference
34      output += "\nCircumference is " +
35          twoDigits.format( circle.getCircumference() );
36
37      // get Circle's area
38      output += "\nArea is " + twoDigits.format( circle.getArea() );
39
40      JOptionPane.showMessageDialog( null, output ); // display output
41
42      System.exit( 0 );
43
44      } // end main
45
46      } // end class CircleTest4
```

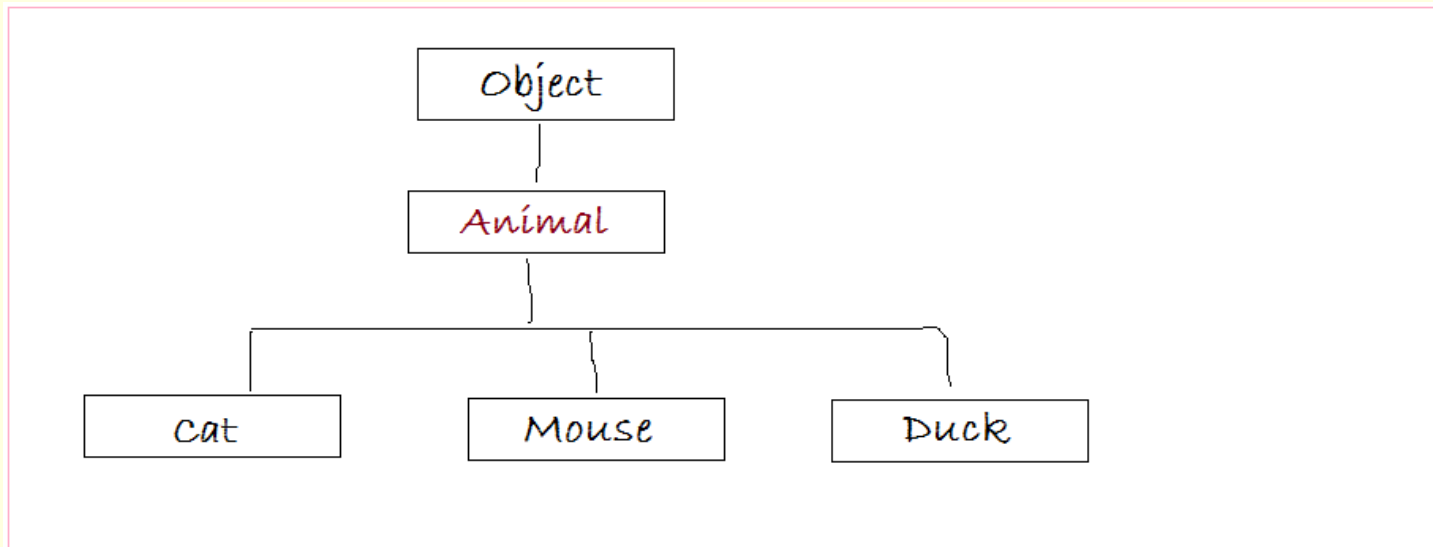


# Constructors và Finalizers trong các lớp con

- ❑ Khởi tạo đối tượng lớp con: Một dãy các lời gọi constructors
  - ❑ Lớp con gọi constructor của lớp cha: rõ ràng (explicit) hay tiềm ẩn (implicit).
  - ❑ Constructor sau cùng được gọi là constructor của Object.
  - ❑ Phần thân của constructor của lớp con hoàn tất thực thi sau cùng.
  - ❑ Ví dụ: Trong phân cấp Point3/Circle4/Cylinder, thì
    - *Constructor của Point3 gọi kề cuối (cuối cùng là constructor của Object)*
    - *Phần thân trong constructor của Point3 hoàn tất thứ 2 (đầu tiên là phần thân constructor của Object).*

# Constructors và Finalizers trong các lớp con

---



# Constructors và Finalizers trong các lớp con

```
Cat tom = new Cat("Tom", 3, 20);
```

Class Animal

```
private String name;  
  
public Animal(String name) {  
    this.name = name;  
} (2)
```

Class Cat

```
private int age;  
private int height;  
  
public Cat(String name, int age, int height) {  
    // Gọi tới cấu tử của class cha (Animal)  
    // Nhằm mục đích khởi tạo các trường trên class cha.  
    super(name);  
    (1)  
  
    // Sau đó mới tới việc khởi tạo giá trị cho các trường của nó  
    (3) this.age = age;  
    this.height = height;  
    (4)  
}
```

# Constructors và Destructors

---

- ❑ Dãy các lời gọi finalize được thực thi
  - ❑ Thứ tự ngược lại với thứ tự gọi constructor
  - ❑ Finalizer của lớp con gọi trước
  - ❑ Kế tiếp là Finalizer của lớp cha kế tiếp phía trên
  - ❑ Sau cùng là Finalizer của Object, Object được giải phóng ra khỏi bộ nhớ.



# Point.java

```
1  // Point.java
2  // Point class declaration represents an x-y coordinate pair.
3
4  public class Point {
5      private int x;  // x part of coordinate pair
6      private int y;  // y part of coordinate pair
7
8      // no-argument constructor
9      public Point()
10     {
11         // implicit call to Object constructor occurs here
12         System.out.println( "Point no-argument constructor: " + this );
13     }
14
15     // constructor
16     public Point( int xValue, int yValue )
17     {
18         // implicit call to Object constructor occurs here
19         x = xValue;  // no need for validation
20         y = yValue;  // no need for validation
21
22         System.out.println( "Point constructor: " + this );
23     }
24
25     // finalizer
26     protected void finalize()
27     {
28         System.out.println( "Point finalizer: " + this );
29     }
30
```

# Point.java

```
31     // set x in coordinate pair
32     public void setX( int xValue )
33     {
34         x = xValue;  // no need for validation
35     }
36
37     // return x from coordinate pair
38     public int getX()
39     {
40         return x;
41     }
42
43     // set y in coordinate pair
44     public void setY( int yValue )
45     {
46         y = yValue;  // no need for validation
47     }
48
49     // return y from coordinate pair
50     public int getY()
51     {
52         return y;
53     }
54
55     // return String representation of Point4 object
56     public String toString()
57     {
58         return "[" + getX() + ", " + getY() + "]";
59     }
60
61 } // end class Point
```

# Circle.java

```
1  // Circle.java
2  // Circle5 class declaration.
3
4  public class Circle extends Point {
5
6      private double radius; // Circle's radius
7
8      // no-argument constructor
9      public Circle()
10     {
11         // implicit call to Point constructor occurs here
12         System.out.println( "Circle no-argument constructor: " + this );
13     }
14
15     // constructor
16     public Circle( int xValue, int yValue, double radiusValue )
17     {
18         super( xValue, yValue ); // call Point constructor
19         setRadius( radiusValue );
20
21         System.out.println( "Circle constructor: " + this );
22     }
23
24     // finalizer
25     protected void finalize()
26     {
27         System.out.println( "Circle finalizer: " + this );
28
29         super.finalize(); // call superclass finalize method
30     }
31
```

# Circle.java

```
32     // set radius
33     public void setRadius( double radiusValue )
34     {
35         radius = ( radiusValue < 0.0 ? 0.0 : radiusValue );
36     }
37
38     // return radius
39     public double getRadius()
40     {
41         return radius;
42     }
43
44     // calculate and return diameter
45     public double getDiameter()
46     {
47         return 2 * getRadius();
48     }
49
50     // calculate and return circumference
51     public double getCircumference()
52     {
53         return Math.PI * getDiameter();
54     }
```

# Circle.java

---

```
55
56     // calculate and return area
57     public double getArea()
58     {
59         return Math.PI * getRadius() * getRadius();
60     }
61
62     // return String representation of Circle5 object
63     public String toString()
64     {
65         return "Center = " + super.toString() + "; Radius = " + getRadius();
66     }
67
68 } // end class Circle
```

# ConstructorFinalizerTest.java

```
1      // ConstructorFinalizerTest.java
2      // Display order in which superclass and subclass
3      // constructors and finalizers are called.
4
5      public class ConstructorFinalizerTest {
6
7          public static void main( String args[] )
8          {
9              Point point;
10             Circle circle1, circle2;
11
12             point = new Point( 11, 22 );
13
14             System.out.println();
15             circle1 = new Circle( 72, 29, 4.5 );
16
17             System.out.println();
18             circle2 = new Circle( 5, 7, 10.67 );
19
20             point = null;      // mark for garbage collection
21             circle1 = null;    // mark for garbage collection
22             circle2 = null;    // mark for garbage collection
23
24             System.out.println();
25
```

# ConstructorFinalizerTest.java

```
26         System.gc(); // call the garbage collector
27
28     } // end main
29
30 } // end class ConstructorFinalizerTest
```

Point constructor: [11, 22]

Point constructor: Center = [72, 29]; Radius = 0.0  
Circle constructor: Center = [72, 29]; Radius = 4.5

Point constructor: Center = [5, 7]; Radius = 0.0  
Circle constructor: Center = [5, 7]; Radius = 10.67

Point finalizer: [11, 22]  
Circle finalizer: Center = [72, 29]; Radius = 4.5  
Point finalizer: Center = [72, 29]; Radius = 4.5  
Circle finalizer: Center = [5, 7]; Radius = 10.67  
Point finalizer: Center = [5, 7]; Radius = 10.67

Subclass **Ci r c l e** constructor body  
executes after superclass **P o i n t 4**'s  
constructor finishes execution.

Finalizer for **C i r c l e** object  
called in reverse order of  
constructors.

# Tính đa hình (Polymorphism)

---

- ❑ Đa hình (polymorphism) là thuật ngữ nói chung liên quan đến → “nhiều hình thức”. Bạn có thể dùng cùng một tên cho những cái khác nhau và compiler sẽ tự động nhận ra mình muốn cái gì.
- ❑ Có một số hình thức đa hình phổ biến như: overriding, overloading.



# Tính đa hình (Polymorphism)

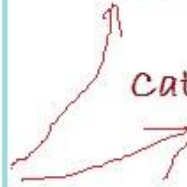
❖ **Đa hình:** cùng một phương thức có thể có những cách thi hành khác nhau.

```
public class Cat extends Animal {  
    ....  
    @Override  
    public String getAnimalName() {  
        return "Cat";  
    }  
}
```

```
public class AsianCat extends Cat {  
    ....  
    @Override  
    public String getAnimalName() {  
        return "Asian Cat";  
    }  
}
```

Cat cat1 = new Cat("Tom", 3, 20);

Cat cat2 = new AsianCat("ATom", 2, 19);



cat1.getAnimalName() = (?) "Cat";

cat2.getAnimalName() = (?) "Asian Cat";

# Tính đa hình

---

❖ **Lớp trừu tượng:** là lớp dùng để thể hiện sự trừu tượng hóa ở mức cao.

*Ví dụ:* lớp “Đối tượng hình học”, “Hình 2D”, “Hình 3D”

(Ví dụ định nghĩa lớp các đối tượng hình học cơ bản)

❖ **Từ khóa abstract:** để khai báo một lớp abstract.

❖ Lớp abstract không thể tạo ra đối tượng.

# Giao tiếp - interface

❖ **Interface:** giao tiếp của một lớp, **là phần đặc tả** (không có phần cài đặt cụ thể) của lớp, nó chứa các khai báo phương thức và thuộc tính để bên ngoài có thể truy xuất được. (java, C#, ...)

- ✓ Lớp sẽ cài đặt các phương thức trong interface.
- ✓ Trong lập trình hiện đại các đối tượng không đưa ra cách truy cập cho một lớp, thay vào đó cung cấp các interface. Người lập trình dựa vào interface để gọi các dịch vụ mà lớp cung cấp.
- ✓ Thuộc tính của interface là các hằng và các phương thức của giao tiếp là trừu tượng (mặc dù không có từ khóa abstract).

# Giao tiếp - interface

**Ví dụ:**

*// Định nghĩa một interface Shape trong tập tin shape.java*

*public interface Shape {*

*// Tính diện tích*

*public abstract double area();*

*// Tính thể tích*

*public abstract double volume();*

*// trả về tên của shape*

*public abstract String getName();*

*}*

# Giao tiếp - interface

*// Lớp Point cài đặt/hiện thực interface tên shape.*

*// Định nghĩa lớp Point trong tập tin Point.java*

*public class Point extends Object implements Shape {*

*protected int x, y; // Tọa độ x, y của 1 điểm*

*// constructor không tham số.*

*public Point() {*

*setPoint( 0, 0 );*

*}*

*// constructor có tham số.*

*public Point(int xCoordinate, int yCoordinate) {*

*setPoint( xCoordinate, yCoordinate );*

*}*

# Giao tiếp - interface

```
// gán tọa độ x, y cho 1 điểm  
public void setPoint( int xCoordinate, int yCoordinate ){  
    x = xCoordinate;  
    y = yCoordinate;  
}  
  
// lấy tọa độ x của 1 điểm  
public int getX() {  
    return x;  
}  
  
// lấy tọa độ y của 1 điểm  
public int getY() {  
    return y;  
}
```

# Giao tiếp - interface

*// Thể hiện tọa độ của 1 điểm dưới dạng chuỗi*

```
public String toString() {  
    return "[" + x + ", " + y + "];  
}
```

*// Tính diện tích*

```
public double area() {  
    return 0.0;  
}
```

*// Tính thể tích*

```
public double volume() {  
    return 0.0;  
}
```

# Giao tiếp - interface

---

```
// trả về tên của đối tượng shape  
public String getName() {  
    return "Point";  
}  
} // end class Point
```



# Giao tiếp - interface

---

## ❖ Kế thừa interface

```
public interface InterfaceName extends interface1, interface2, interface3  
{  
  
    // ...  
  
}
```

# Quan hệ giữa Class và Interface

	Class	Interface
Class	extends	implements
Interface		extends