BỘ MÔN CÔNG NGHỆ PHẦN MỀM
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Bài 06. Một số kỹ thuật trong kế thừa

Cao Tuấn Dũng – Nguyễn Thị Thu Trang

Mục tiêu của bài học

- Trình bày nguyên lý định nghĩa lại trong kế thừa
- Đơn kế thừa và đa kế thừa
- Giao diện và lớp trừu tượng
- Sử dụng các vấn đề trên với ngôn ngữ lập trình Java.

Nội dung

1. Định nghĩa lại (Redefine/Overiding)
2. Lớp trừu tượng (Abstract class)
3. Đơn kế thừa và đa kế thừa
4. Giao diện (Interface)

Nội dung

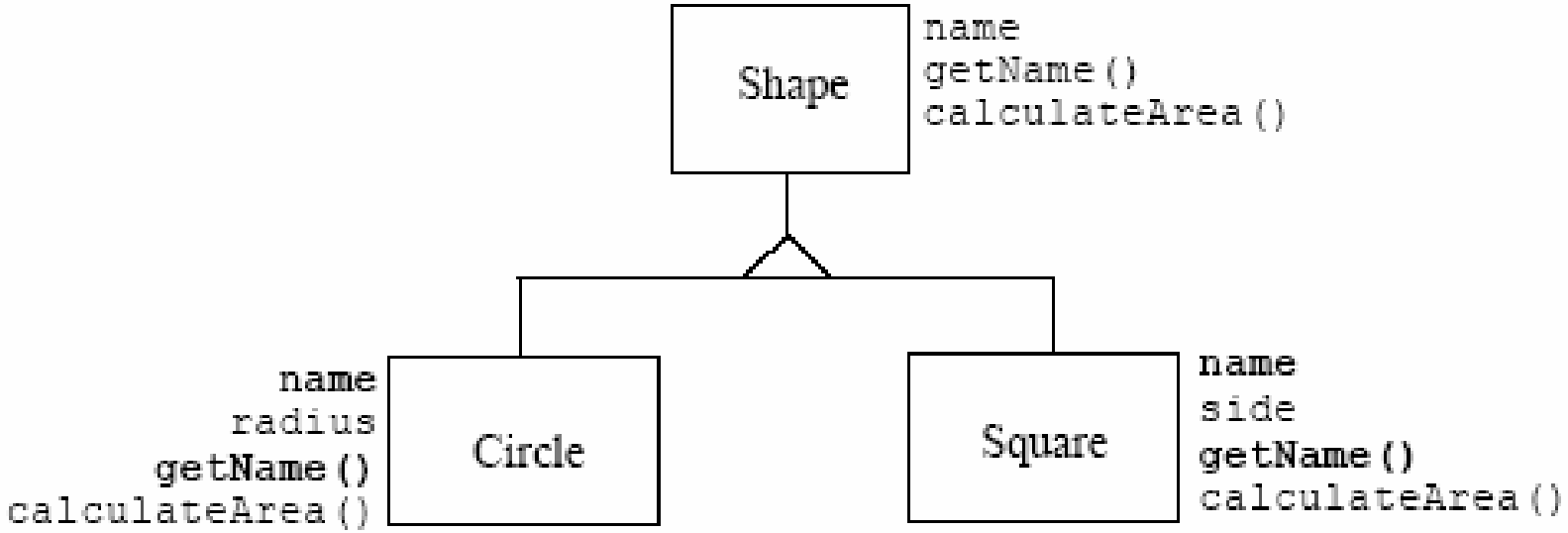
1. Định nghĩa lại (Redefine/Overiding)
2. Lớp trừu tượng (Abstract class)
3. Đơn kế thừa và đa kế thừa
4. Giao diện (Interface)

1. Định nghĩa lại hay ghi đè

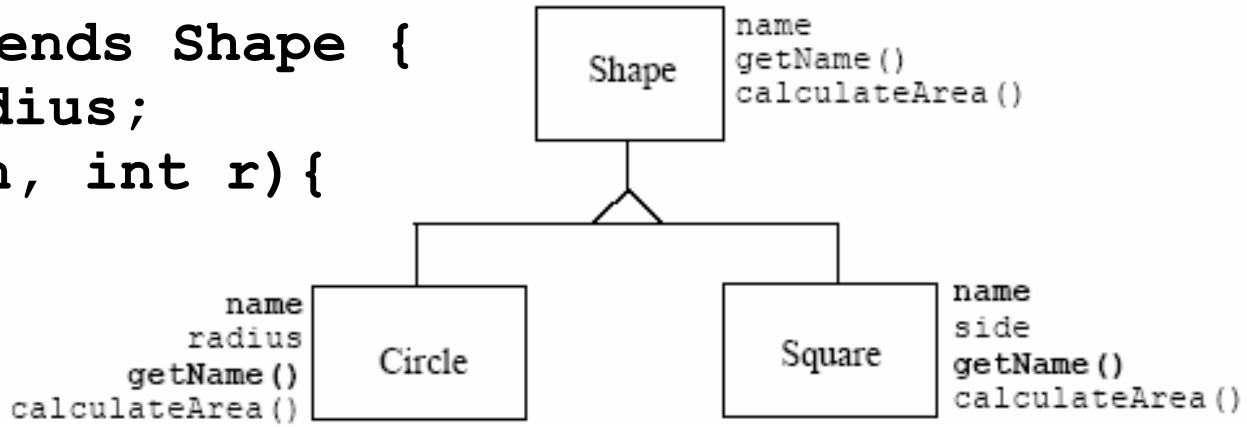
- Lớp con có thể định nghĩa phương thức trùng tên với phương thức trong lớp cha:
  - Nếu phương thức mới chỉ trùng tên và khác chữ ký (số lượng hay kiểu dữ liệu của đối số)
  - → Chồng phương thức (Method Overloading)
  - Nếu phương thức mới hoàn toàn giống về giao diện (chữ ký)
  - → Định nghĩa lại hoặc ghi đè
  - (Method Redefine/Override)

1. Định nghĩa lại hay ghi đè (2)

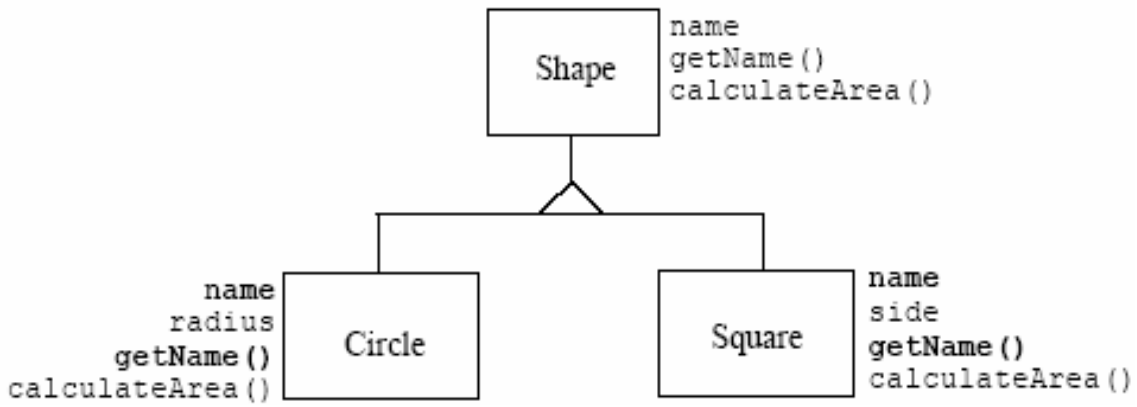
- Phương thức ghi đè sẽ thay thế hoặc làm rõ hơn cho phương thức cùng tên trong lớp cha
- Đối tượng của lớp con sẽ hoạt động với phương thức mới phù hợp với nó



```
class Shape {
  protected String name;
  Shape(String n) { name = n; }
  public String getName() { return name; }
  public float calculateArea() { return 0.0f; }
}
class Circle extends Shape {
  private int radius;
  Circle(String n, int r){
      super(n);
      radius = r;
  }

  public float calculateArea() {
      float area = (float) (3.14 * radius *
  radius);
      return area;
  }
}
```



```
class Square extends Shape {
  private int side;
  Square(String n, int s) {
     super(n);
     side = s;
  }
  public float calculateArea() {
     float area = (float) side * side;
     return area;
  }
}
```



Thêm lớp Triangle

```
class Triangle extends Shape {
  private int base, height;
  Triangle(String n, int b, int h) {
     super(n);
     base = b; height = h;
  }
  public float calculateArea() {
     float area = 0.5f * base * height;
     return area;
  }
}
```

this và super

- this và super có thể sử dụng cho các phương thức/thuộc tính non-static và phương thức khởi tạo
  - **this**: tìm kiếm phương thức/thuộc tính trong lớp hiện tại
  - **super**: tìm kiếm phương thức/thuộc tính trong lớp cha trực tiếp
- Từ khóa **super** cho phép tái sử dụng các đoạn mã của lớp cha trong lớp con

```
package abc;
public class Person {
  protected String name;
  protected int age;
  public String getDetail() {
      String s = name + "," + age;
      return s;
  }
}

import abc.Person;
public class Employee extends Person {
   double salary;
   public String getDetail() {
     String s = super.getDetail() + "," + salary;
     return s;
   }
}
```

1. Định nghĩa lại hay ghi đè (3)

- Một số quy định
  - Phương thức ghi đè trong lớp con phải
    - Có danh sách tham số giống hệt phương thức kế thừa trong lớp cha.
    - Có cùng kiểu trả về với phương thức kế thừa trong lớp cha
  - Không được phép ghi đè:
    - Các phương thức hằng (final) trong lớp cha
    - Các phương thức static trong lớp cha
    - Các phương thức private trong lớp cha

1. Định nghĩa lại hay ghi đè (3)

- Một số quy định (tiếp)
  - Các chỉ định truy cập không giới hạn chặt hơn phương thức trong lớp cha
    - Ví dụ, nếu ghi đè một phương thức protected, thì phương thức mới có thể là protected hoặc public, mà không được là private.

Ví dụ

```
class Parent {
  public void doSomething() {}
  protected int doSomething2() {
      return 0;
  }
}
class Child extends Parent {
  protected void doSomething() {}
  protected void doSomething2() {}
}
```

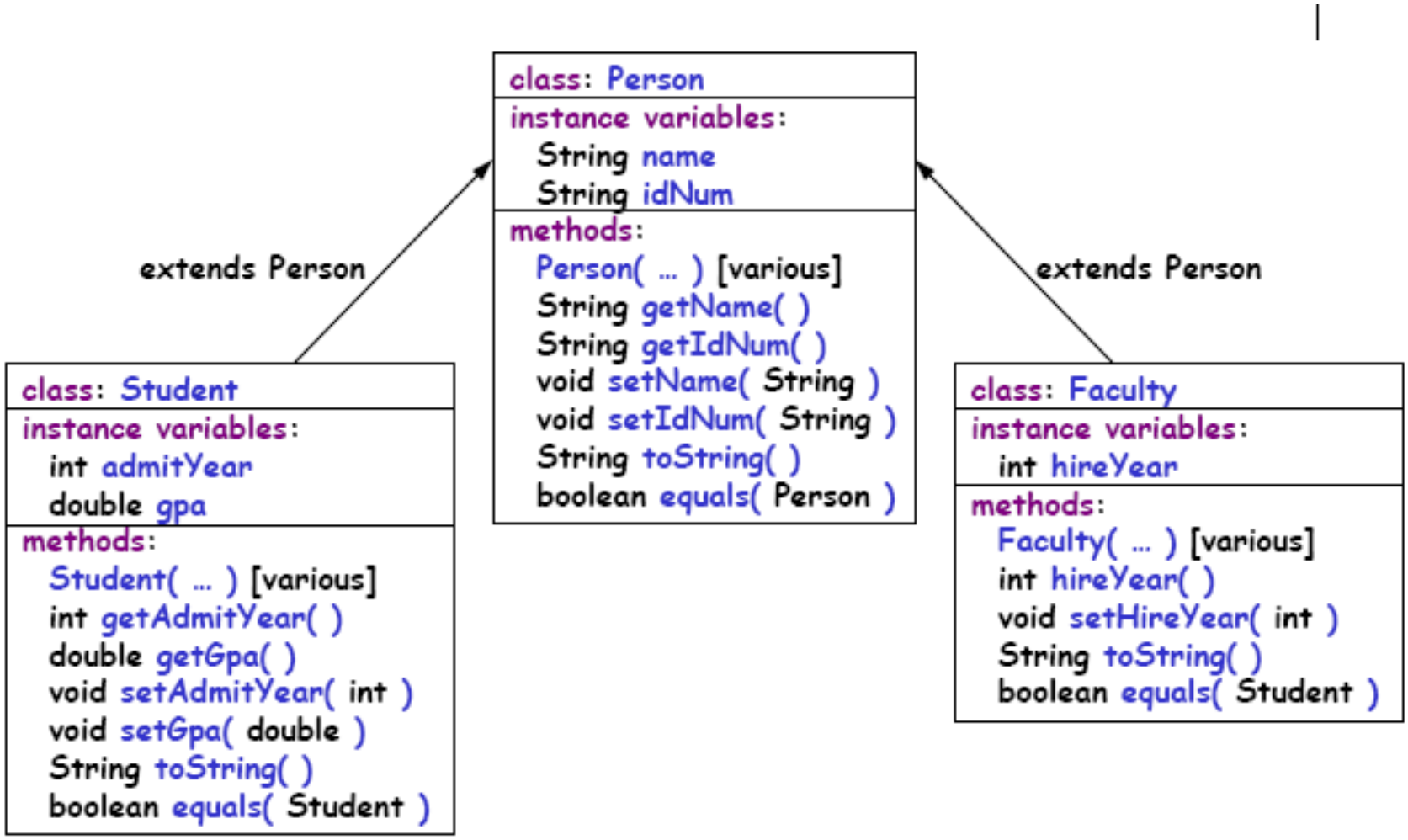
cannot override: attempting to use incompatible return type

cannot override: attempting to assign weaker access privileges; was public

Ví dụ

```
class Parent {
  public void doSomething() {}
  private int doSomething2() {
     return 0;
  }
}
class Child extends Parent {
  public void doSomething() {}
  private void doSomething2() {}
}
```

Person, Student và Faculty



Lớp Faculty

```
package university;
public class Faculty extends Person {
  private int hireYear;
  public Faculty( ) { super( ); hireYear = -1; }
  public Faculty( String n, String id, int yr ) {
      super(n, id);
      hireYear = yr;
  }
  public Faculty( Faculty f ) {
  this( f.getName( ), f.getIdNum( ), f.hireYear );
  }
  int getHireYear( ) { return hireYear; }
  void setHireYear( int yr ) { hireYear = yr; }
  public String toString( ) {
      return super.toString( ) + " " + hireYear;
  }
  public boolean equals( Faculty f ) {
      return super.equals( f ) && hireYear == f.hireYear;
  }
}
```

Định nghĩa lại phương thức (overriding)

- Xảy ra khi lớp kế thừa muốn thay đổi chức năng của môt phương thức kế thừa từ lớp cha (super).

```
public class Person {
…
  public String toString( ) { … }
}
public class Student extends Person {
…
  public String toString( ) { … }
}
Student bob = new Student("Bob Goodstudent","123-45-
  6789",2004,4.0 );
System.out.println( "Bob's info: " + bob.toString( ) );
```

Định nghĩa lại phương thức của lớp cha

Lời gọi đến phương thức của lớp con

Cấm định nghĩa lại với final

- Đôi lúc ta muốn hạn chế việc định nghĩa lại vì các lý do sau:
  - Tính đúng đắn: Định nghĩa lại một phương thức trong lớp dẫn xuất có thể làm sai lạc ý nghĩa của nó
  - Tính hiệu quả: Cơ chế kết nối động không hiệu quả về mặt thời gian bằng kết nối tĩnh. Nếu biết trước sẽ không định nghĩa lại phương thức của lớp cơ sở thì nên dùng từ khóa final đi với phương thức

```
public final String baseName () {
return “Person”;}
}
```

Lớp cơ sở ship

```
public class Ship {
  public double x=0.0, y=0.0, speed=1.0,
  direction=0.0;
  public String name;
  public Ship(double x, double y, double speed, double
  direction, String name) {
     this.x = x;
     this.y = y;
     this.speed = speed;
     this.direction = direction;
     this.name = name;
  }
  public Ship(String name) {
      this.name = name;
  }
  private double degreesToRadians(double degrees) {
      return(degrees * Math.PI / 180.0);
  }
...
```

Lớp cơ sở ship

```
  public void move() {
      move(1);
  }
  public void move(int steps) {
    double angle = degreesToRadians(direction);
    x = x + (double)steps * speed * Math.cos(angle);
    y = y + (double)steps * speed * Math.sin(angle);
  }
  public void printLocation() {
    System.out.println(name + " is at ("+ x + "," + y +
    ").");
  }
}
...
```

Lớp dẫn xuất Speedboat

```
public class Speedboat extends Ship {
  private String color = "red";
  public Speedboat(String name) {
      super(name);
      setSpeed(20);
  }
  public Speedboat(double x, double y, double speed,
  double direction, String name, String color) {
      super(x, y, speed, direction, name);
      setColor(color);
  }
  public void printLocation() {
      System.out.print(getColor().toUpperCase() + " ");
      super.printLocation();
  }
...
```

Lớp Book2

```
class Book2 {
  protected int pages;

  public Book2(int pages) {
    this.pages = pages;
  }

  public void pageMessage() {
    System.out.println("Number of pages: " +
                       pages);
  }
}
```

Lớp Dictionary2

```
class Dictionary2 extends Book2 {
  private int definitions;

  public Dictionary2(int pages, int definitions) {
    super (pages);
    this.definitions = definitions;
  }

  public void definitionMessage () {
    System.out.println("Number of definitions: " +
                       definitions);
    System.out.println("Definitions per page: " +
                       definitions/pages);
  }
}
```

Lớp Words2

```
class Words2 {
  public static void main (String[] args) {
    Dictionary2 webster = new Dictionary2(1500, 52500);
    webster.pageMessage();
    webster.definitionMessage();
  }
}
```

Kết quả:

```
C:\Examples>java Words2
Number of pages: 1500
Number of definitions: 52500
Definitions per page: 35
```

Lớp Book3

```
class Book3 {
  protected String title;
  protected int pages;

  public Book3(String title, int pages) {
    this.title = title;
    this.pages = pages;
  }

  public void info() {
    System.out.println("Title: " + title);
    System.out.println("Number of pages: " + pages);
  }
}
```

Lớp : Dictionary3a

```
class Dictionary3a extends Book3 {
  private int definitions;

  public Dictionary3a(String title, int pages,
                      int definitions) {
    super (title, pages);
    this.definitions = definitions;
  }

  public void info() {
    System.out.println("Dictionary: " + title);
    System.out.println("Number of definitions: " +
                       definitions);
    System.out.println("Definitions per page: " +
                       definitions/pages);
  }
}
```

Lớp : Dictionary3b

```
class Dictionary3b extends Book3 {
  private int definitions;

  public Dictionary3b(String title, int pages,
                      int definitions) {
    super (title, pages);
    this.definitions = definitions;
  }

  public void info() {
    super.info();
    System.out.println("Number of definitions: " +
                       definitions);
    System.out.println("Definitions per page: " +
                       definitions/pages);
  }
}
```

Lớp Books

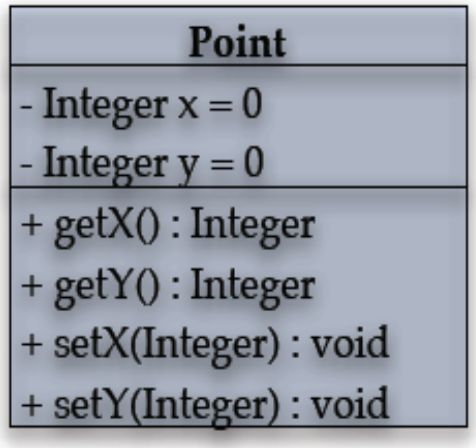
```
class Books {
  public static void main (String[] args) {
    Book3 java = new Book3("Introduction to Java", 350);
    java.info();
    System.out.println();
    Dictionary3a webster1 =
      new Dictionary3a("Webster English Dictionary",
                       1500, 52500);
    webster1.info();
    System.out.println();
    Dictionary3b webster2 =
      new Dictionary3b("Webster English Dictionary",
                       1500, 52500);
    webster2.info();
  }
}
```

Lớp : Books

Kết quả:

```
C:\Examples>java Books
Title: Introduction to Java
Number of pages: 350

Dictionary: Webster English Dictionary
Number of definitions: 52500
Definitions per page: 35

Title: Webster English Dictionary
Number of pages: 1500
Number of definitions: 52500
Definitions per page: 35
```

Ví dụ: Point, Circle, Cylinder



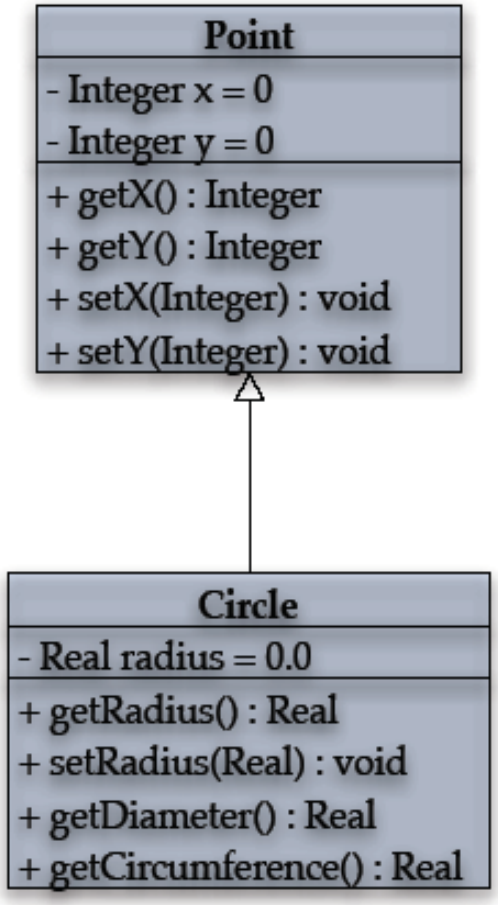

```
public class Point {
    private int x;
    private int y;

    // default constructor
    public Point() { this(0, 0); }

    // constructor
    public Point (int xValue, int yValue) {
        x = xValue;
        y = yValue;
    }

    public void setX (int xValue) { x = xValue; }

    public void setY (int yValue) { y = yValue; }

    public int getX () { return x; }

    public int getY () { return y; }
}
```

state variables are declared *private*

default constructor brings instance to a *consistent state*

mutator methods to *change state*

accessor methods to *read state*




```
public class Circle extends Point {
    private double radius;

    public Circle() {}

    public Circle(int xValue, int yValue) {
        super(xValue, yValue);
        setRadius(0.0);
    }

    // constructor
    public Circle (int xValue, int yValue, double radius) {
        super(xValue, yValue);
        setRadius(radius);
    }

    public void setRadius (double radius) {
        this.radius = (radius < 0.0 ? 0.0 : radius);
    }

    public double getRadius () { return radius; }

    public double getDiameter() { return 2 * getRadius(); }

    public double getCircumference() { return Math.PI * getDiameter(); }
}
```

implicit call to `Point()`

explicit call to `Point(xValue, yValue)`

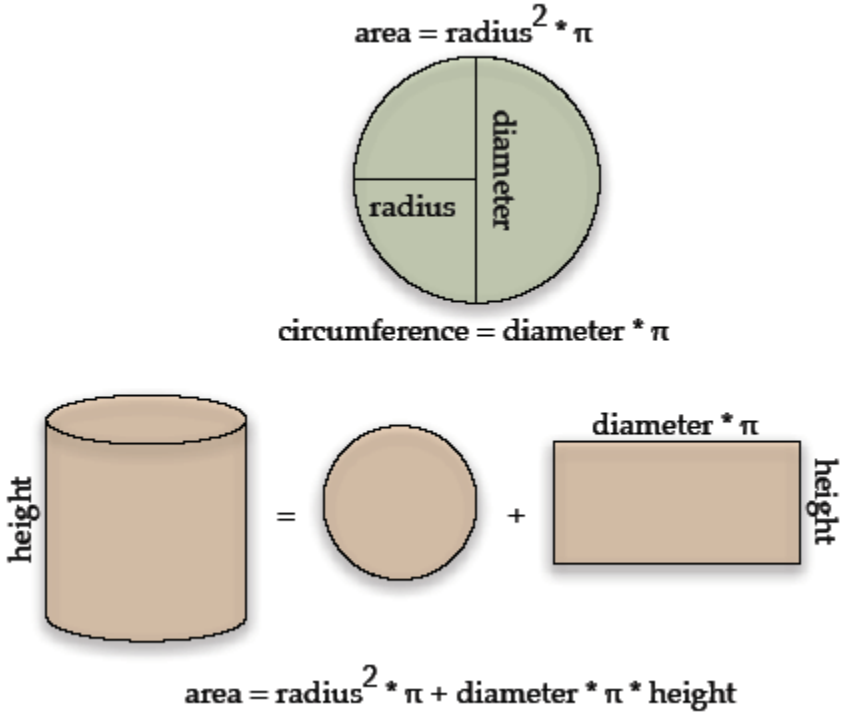
Good practice: call the *mutator* method

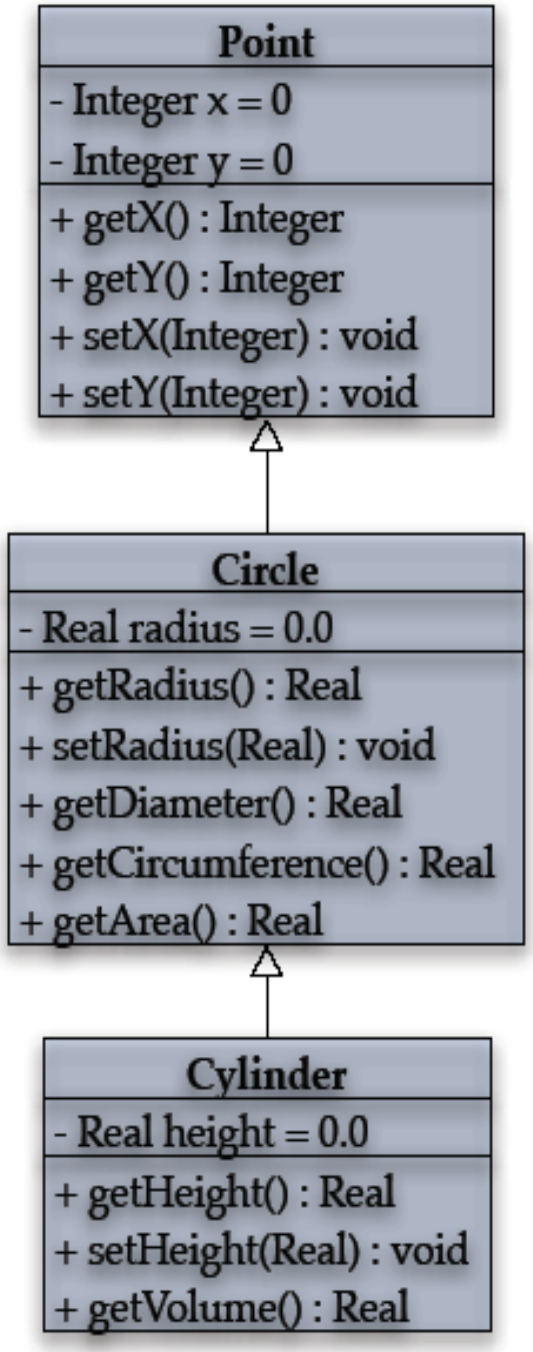
Good practice: call the *accessor* methods

inheritance: *no need* to redefine all the `Point` methods

Định nghĩa lại: Cylinder

- Cylinder phải định nghĩa lại getArea() thừa kế từ Circle
- Dùng getArea() và getCircumference() của lớp cha






```
public class Cylinder extends Circle {
    private double height;

    public Cylinder() {
        // implicit call to Circle default
        // constructor occurs here
    }

    public Cylinder (int xValue, int yValue,
                     double radiusValue, double heightValue) {
        super(xValue, yValue, radiusValue);
        setHeight(heightValue);
    }

    public void setHeight (double height) {
        this.height = (height < 0.0 ? 0.0 : height);
    }

    public double getHeight () { return height; }

    public double getArea() {
        return 2 * super.getArea()
               + getCircumference() * getHeight();
    }

    public double getVolume() {
        return super.getArea() * getHeight();
    }
}
```

inheritance
receive *data*
and *behaviour*
from `Circle`

super constructor
to *initialise*
inherited fields

use mutator
to *initialise own*
data

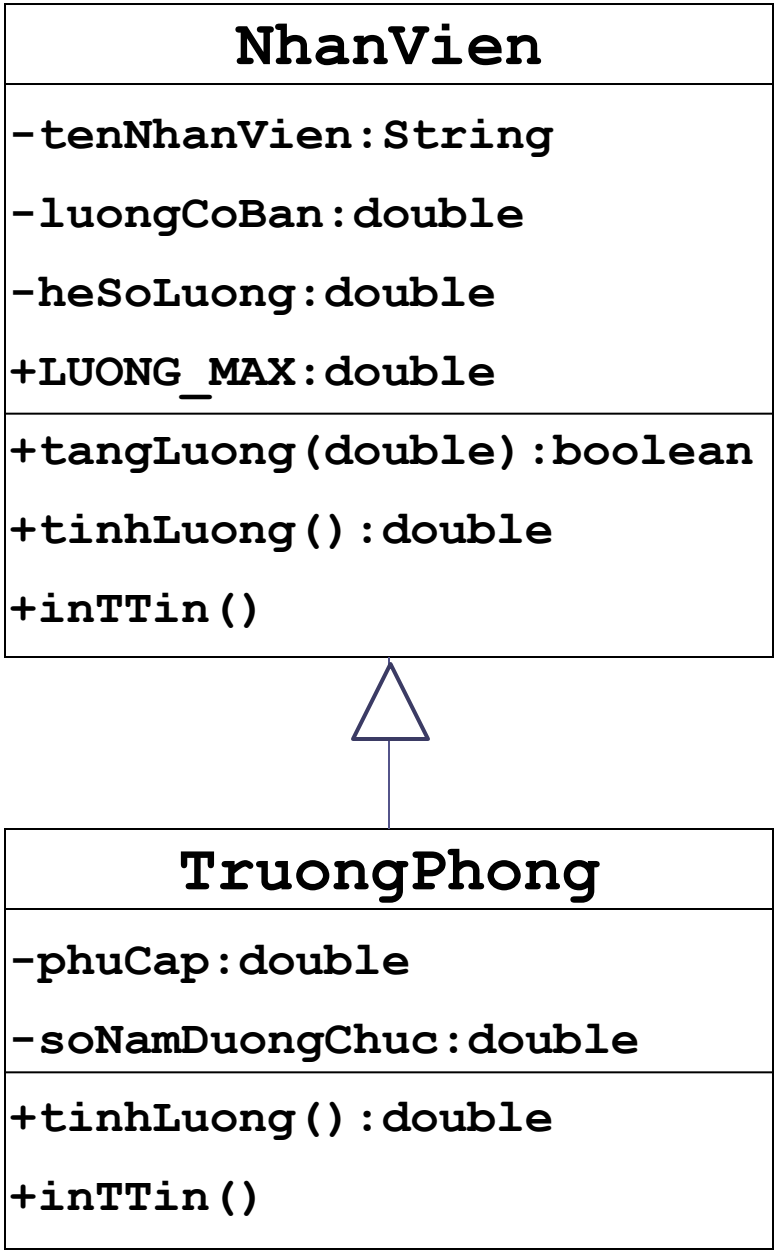
encapsulation
provide *mutators*
and *accessors*

redefinition
override existing
behaviour

extension
provide extra
behaviour

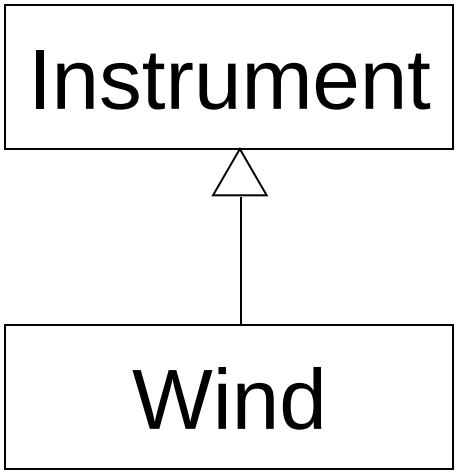
Bài tập

- Sửa lại lớp NhanVien:
  - 3 thuộc tính không hằng của NhanVien kế thừa lại cho lớp TruongPhong
- Viết mã nguồn của lớp TruongPhong như hình vẽ
  - Viết các phương thức khởi tạo cần thiết để khởi tạo các thuộc tính của lớp TruongPhong
  - Lương của trưởng phòng = Lương Cơ bản * hệ số lương + phụ cấp



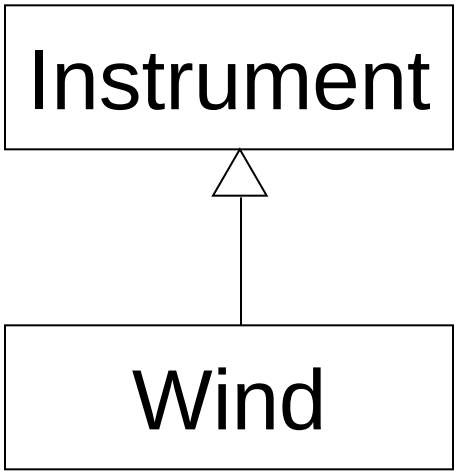
Các vấn đề trong kế thừa

- Chuyển đổi một đối tượng thuộc lớp thừa kế thành đối tượng thuộc lớp cơ sở được gọi là “upcasting”
- Mọi thông điệp mà ta có thể gửi cho đối tượng lớp cơ sở đều có thể gửi cho đối tượng lớp thừa kế thay thế nó.



Upcasting – Java

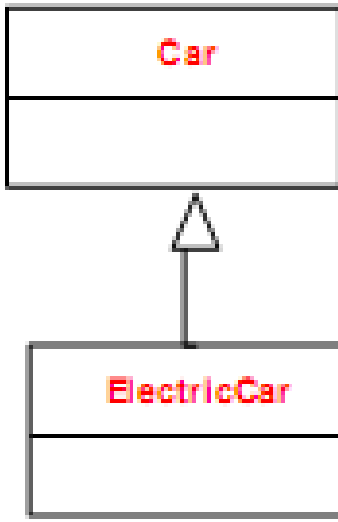
```
import java.util.*;

class Instrument {
  public void play() {}
  static void tune(Instrument i) {
    // ...
    i.play();
  }
}
```




```
// Wind objects are instruments
// because they have the same
 // interface:
class Wind extends Instrument {
  public static void main(String[] args) {
    Wind flute = new Wind();
    Instrument.tune(flute); // Upcasting
  }
}
```

Upcast

```
class Car{};
class ElectricCar extends Car{};
Car c = new ElectricCar ();
```

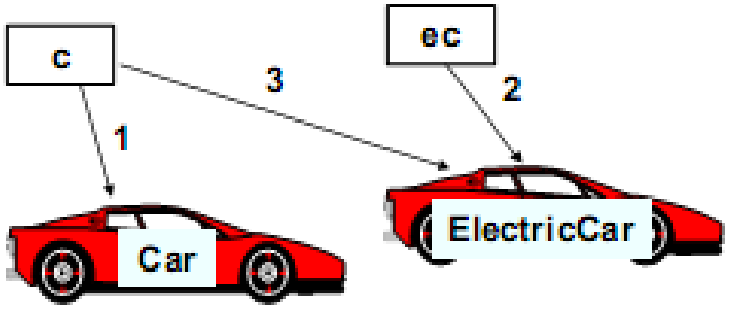


- kiểu tham chiếu và kiểu đối tượng là các khái niệm riêng biệt
- Đối tượng tham chiếu bởi ‘c’ sẽ thuộc kiểu ElectricCar

Upcast

- Car c = new Car();
- ElectricCar ec = new ElectricCar ();
- c = ec;

- Tự động upcast (ngầm)
- Kiểu của đối tượng không thay đổi



Down Cast

- Xảy ra khi gán một đối tượng thuộc kiểu khái quát (cơ sở) hơn về một kiểu cụ thể hơn (dẫn xuất).
- Cẩn thận trong sử dụng
- Thực hiện tường minh

```
Car c = new ElectricCar(); // nâng kiểu không tường
  minh
c.recharge(); // lỗi
// Hạ kiểu tường minh
ElectricCar ec = (ElectricCar)c;
ec.recharge(); // ok
```

Down Cast

```
Car c = new Car();
c.recharge(); // lỗi
// explicit downcast
ElectricCar ec = (ElectricCar)c;
ec.recharge(); // lỗi
```

Lỗi Runtime

Tránh lỗi Down Cast

- Sử dụng toán tử instanceof

```
Car c = new Car();
ElectricCar ec;

if (c instanceof ElectricCar ){
  ec = (ElectricCar) c;
  ec.recharge();
}
```

((ElectricCar)c).recharge();

Nội dung

1. Định nghĩa lại (Redefine/Overiding)
2. Lớp trừu tượng (Abstract class)
3. Đa kế thừa và đơn kế thừa
4. Giao diện (Interface)

Lớp trừu tượng (Abstract Class)

- Lớp trừu tượng là lớp mà ta không thể tạo ra các đối tượng từ nó. Thường lớp trừu tượng được dùng để định nghĩa các "khái niệm chung", đóng vai trò làm lớp cơ sở cho các lớp "cụ thể" khác.
- Đi cùng từ khóa abstract

```
public abstract class Product
{
    // contents
}
```

2. Lớp trừu tượng (Abstract Class)

- Không thể thể hiện hóa (instantiate – tạo đối tượng của lớp) trực tiếp
- Chưa đầy đủ, thường được sử dụng làm lớp cha. Lớp con kế thừa nó sẽ hoàn thiện nốt.

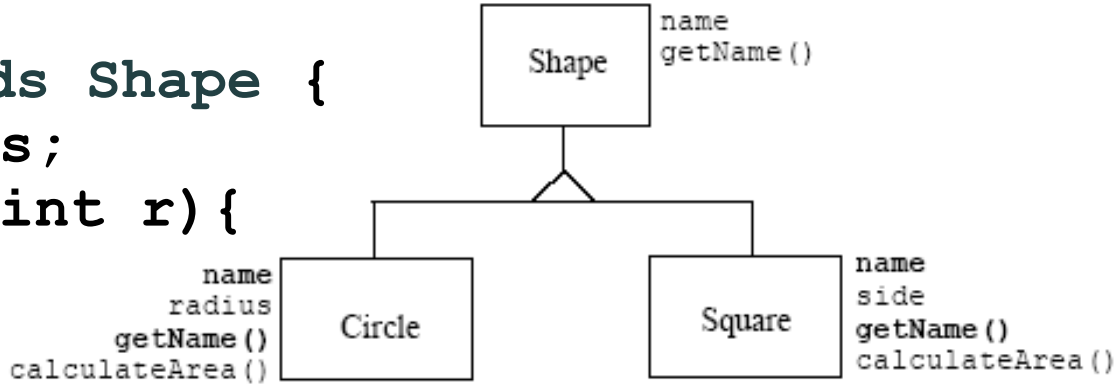
Lớp trừu tượng (Abstract Class)

- Lớp trừu tượng có thể chứa các phương thức trừu tượng không được định nghĩa
- Các lớp dẫn xuất có trách nhiệm định nghĩa lại (overriding) các phương thức trừu tượng này
- Sử dụng các lớp trừu tượng đóng vai trò quan trọng trong thiết kế phần mềm. Nó cho phép định nghĩa tạo ra những phần tử dùng chung trong cây thừa kế, nhưng quá khái quát để tạo ra các thể hiện.

2. Lớp trừu tượng (2)

- Để trở thành một lớp trừu tượng, cần:
  - Khai báo với từ khóa abstract
  - Chứa ít nhất một phương thức trừu tượng (abstract method - chỉ có chữ ký mà không có cài đặt cụ thể)
    - public abstract float calculateArea();
  - Lớp con khi kế thừa phải cài đặt cụ thể cho các phương thức trừu tượng của lớp cha → Phương thức trừu tượng không thể khai báo là final hoặc static.
- Nếu một lớp có một hay nhiều phương thức trừu tượng thì nó phải là lớp trừu tượng

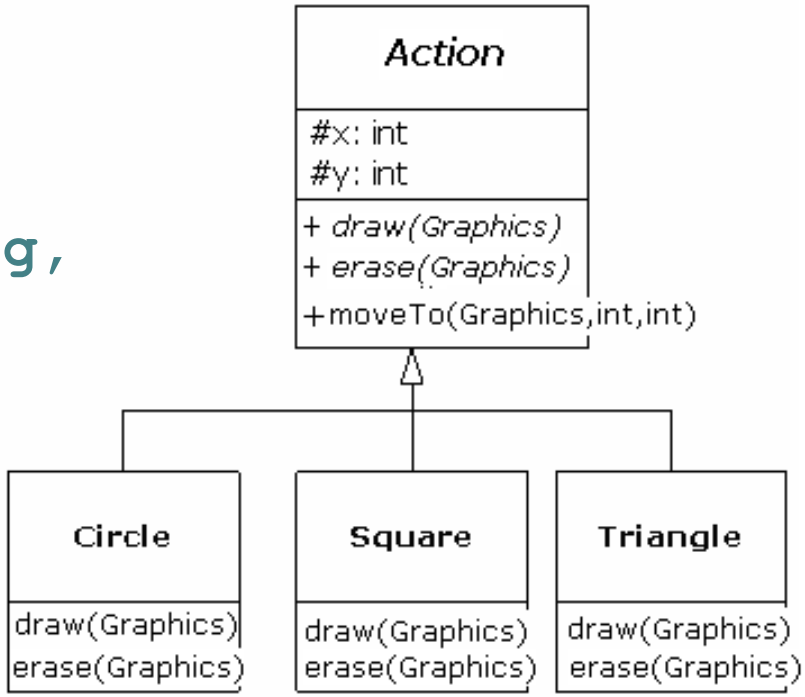
```
abstract class Shape {
  protected String name;
  Shape(String n) { name = n; }
  public String getName() { return name; }
  public abstract float calculateArea();
}
class Circle extends Shape {
  private int radius;
  Circle(String n, int r){
    super(n);
    radius = r;
  }

  public float calculateArea() {
   float area = (float) (3.14 * radius * radius);
   return area;
  }
}
```



Lớp con bắt buộc phải override tất cả các phương thức abstract của lớp chả

Ví dụ lớp trừu tượng

```
import java.awt.Graphics;
abstract class Action {
  protected int x, y;
  public void moveTo(Graphics g,
           int x1, int y1) {
    erase(g);
    x = x1; y = y1;
    draw(g);
   }

  abstract public void erase(Graphics g);
  abstract public void draw(Graphics g);
}
```



Ví dụ lớp trừu tượng (2)

```
class Circle extends Action {
  int radius;
  public Circle(int x, int y, int r) {
    super(x, y); radius = r;
  }
  public void draw(Graphics g) {
    System out println("Draw circle at ("
                              + x + "," + y + ")");
    g.drawOval(x-radius, y-radius,
                  2*radius, 2*radius);
  }
  public void erase(Graphics g) {
     System.out.println("Erase circle at ("
                                + x + "," + y + ")");
    // paint the circle with background color...
  }
}
```

Abstract Class

```
abstract class Point {
    private int x, y;
    public Point(int x, int y) { this.x = x; this.y = y; }
    public void move(int dx, int dy) {
    x += dx; y += dy;
    plot();
    }
    public abstract void plot(); // phương thức trừu tượng không
       có code thực hiện
}
```

Abstract Class

```
abstract class ColoredPoint extends Point {
    int color;
    public ColoredPoint(int x, int y, int color) { super(x, y); this.color =
    color; }
}

class SimpleColoredPoint extends ColoredPoint {
    public SimpleColoredPoint(int x, int y, int color) {
            super(x,y,color);
     }
    public void plot() { … } // code to plot a SimplePoint
}
```

Abstract Class

- Lớp ColoredPoint không cài đặt mã cho phương thức plot() do đó phải khai báo: abstract

- Chỉ có thể tạo ra đối tượng của lớp SimpleColoredPoint.

- Tuy nhiên có thể:

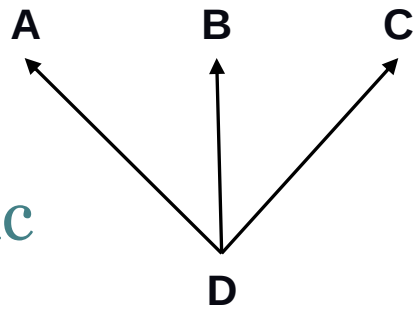
```
Point p = new SimpleColoredPoint(a, b, red);
p.plot();
```

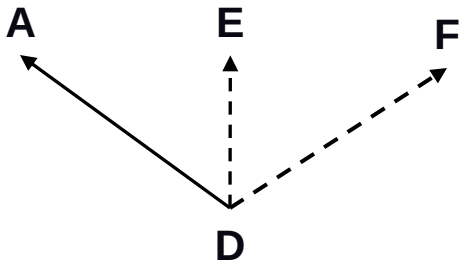
Nội dung

1. Định nghĩa lại (Redefine/Overiding)
2. Lớp trừu tượng (Abstract class)
3. Đa kế thừa và đơn kế thừa
4. Giao diện (Interface)

Đa kế thừa và đơn kế thừa

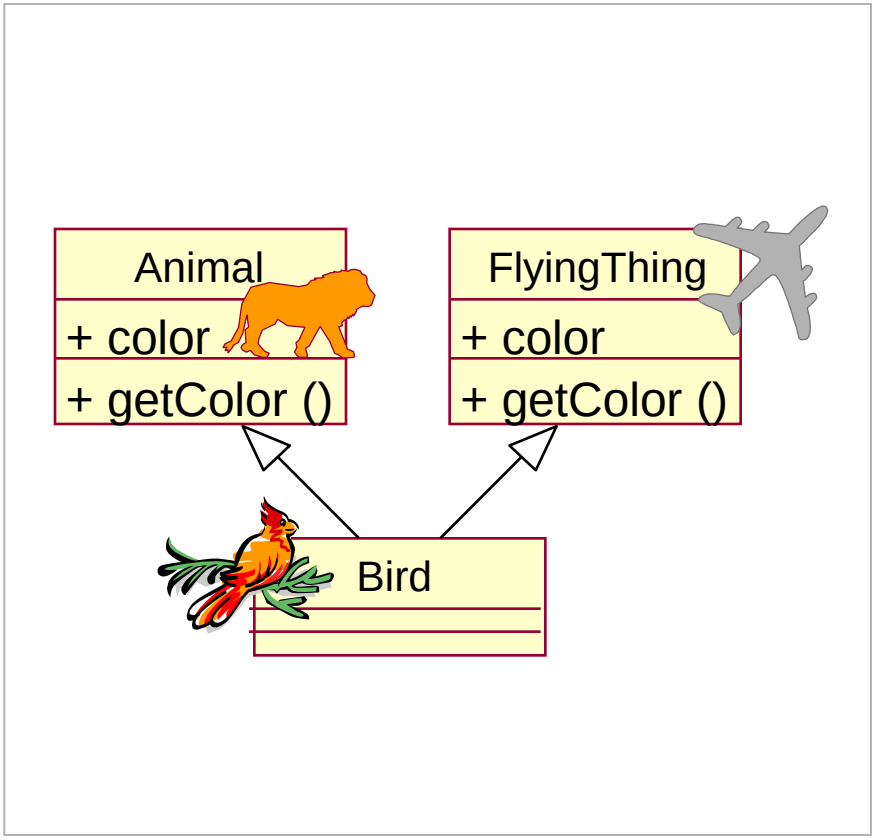
- Đa kế thừa (Multiple Inheritance)
  - Một lớp có thể kế thừa nhiều lớp khác
  - C++ hỗ trợ đa kế thừa
- Đơn kế thừa (Single Inheritance)
  - Một lớp chỉ được kế thừa từ một lớp khác
  - Java chỉ hỗ trợ đơn kế thừa
  - → Đưa thêm khái niệm Giao diện (Interface)



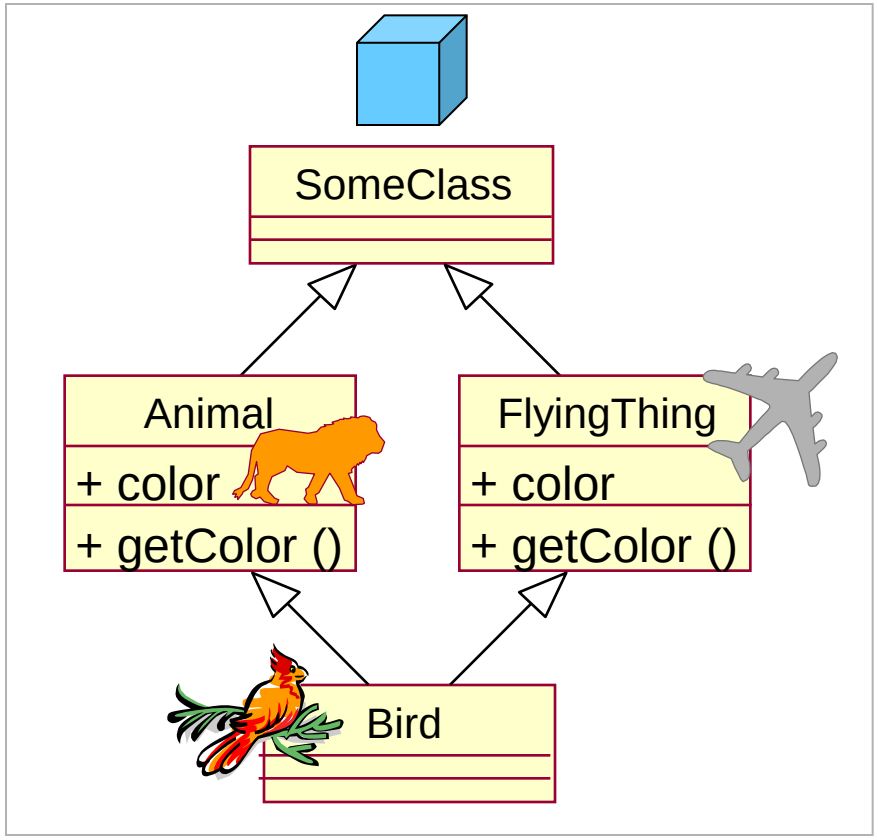


Vấn đề gặp phải trong Đa kế thừa

Name clashes on
attributes or operations



Repeated inheritance



Resolution of these problems is implementation-dependent.

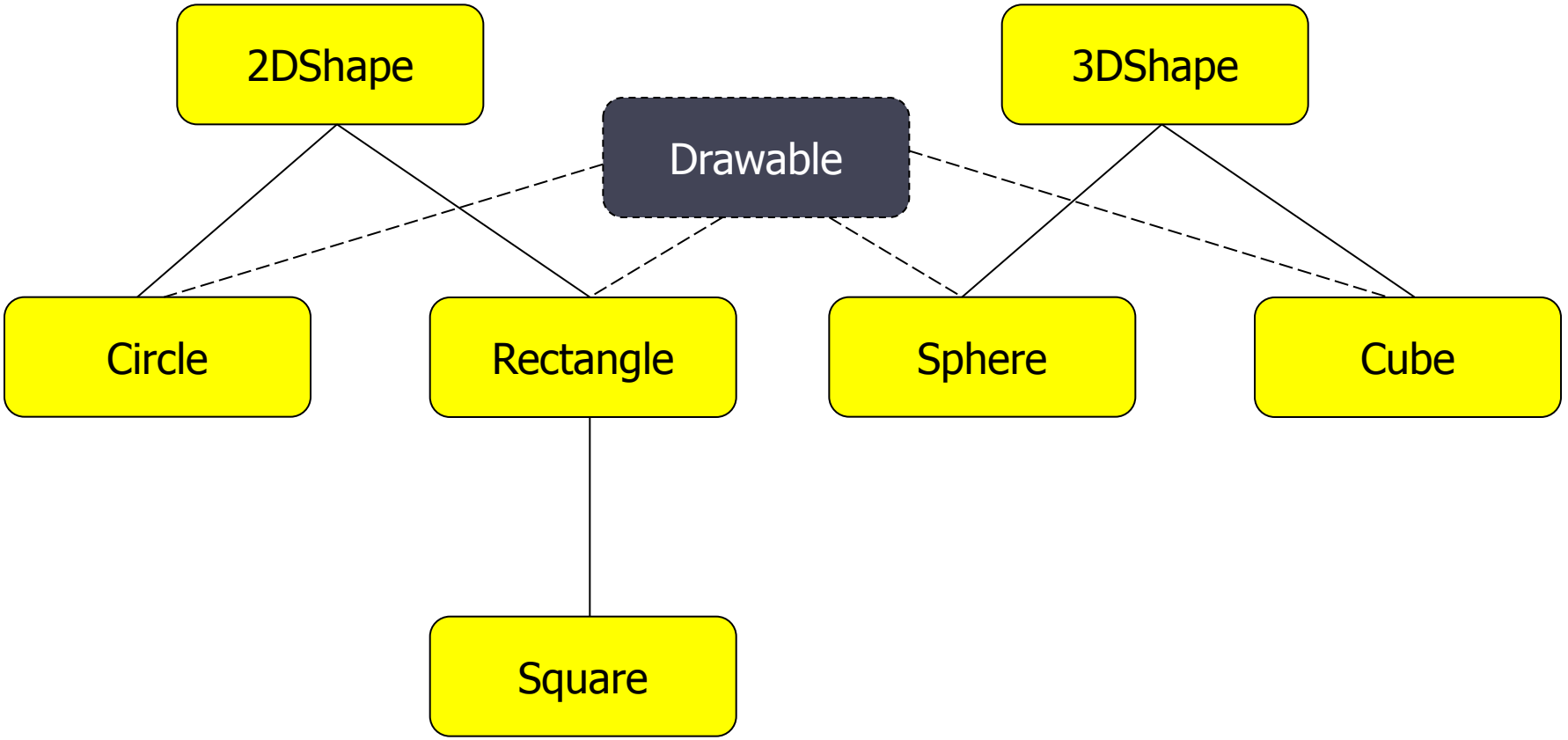
Nội dung

1. Định nghĩa lại (Redefine/Overiding)
2. Lớp trừu tượng (Abstract class)
3. Đa kế thừa và đơn kế thừa
4. Giao diện (Interface)

Thừa kế pha trộn
mix-in inheritance

- Trong dạng thừa kế này, một "lớp" sẽ cung cấp một số chức năng nhằm hòa trộn vào những lớp khác.
- Một lớp pha trộn thường sử dụng lại một số chức năng định nghĩa từ lớp cung cấp nhưng lại thừa kế từ một lớp khác.
- Là phương tiện giúp các đối tượng không có liên hệ thông qua sơ đồ phân cấp lớp có thể tương tác với nhau.
- Trong Java thừa kế pha trộn được thể hiện qua khái niệm Interface

Interface


2DShape
3DShape
Drawable
Circle
Rectangle
Sphere
Cube
Square

Giao diện – Góc nhìn quan niệm

- Interface: đặc tả cho các bản cài đặt (implementation) khác nhau.
- Phân chia ranh giới:
  - Cái gì (What) và như thế nào (How)
  - Đặc tả và Cài đặt cụ thể.

Giao diện – Góc nhìn quan niệm

- Interface không cài đặt bất cứ một phương thức nào nhưng để lại cấu trúc thiết kế trên bất cứ lớp nào sử dụng nó

- Một interface: 1 contract – mà trong đó các nhóm phát triển phần mềm thống nhất sản phẩm của họ tương tác với nhau như thế nào, mà không đòi hỏi bất cứ một tri thức về cách thức tiến hành của nhau.

Ví dụ

- Lớp xe đạp – Lớp thủ kho:
  - Thủ kho không quan tâm các hàng hóa cần quản lý có đặc thù gì ngoài thông tin về giá và mã số hàng.
- Lớp ô tô tự hành – GPS:
  - Nhà sản xuất ô tô tạo ra các ô tô với các hoạt động: khởi động, tăng tốc, dừng, quay trái, phải,..
  - GPS: thông tin tọa độ, tình trạng giao thông – ra quyết định điều khiển xe

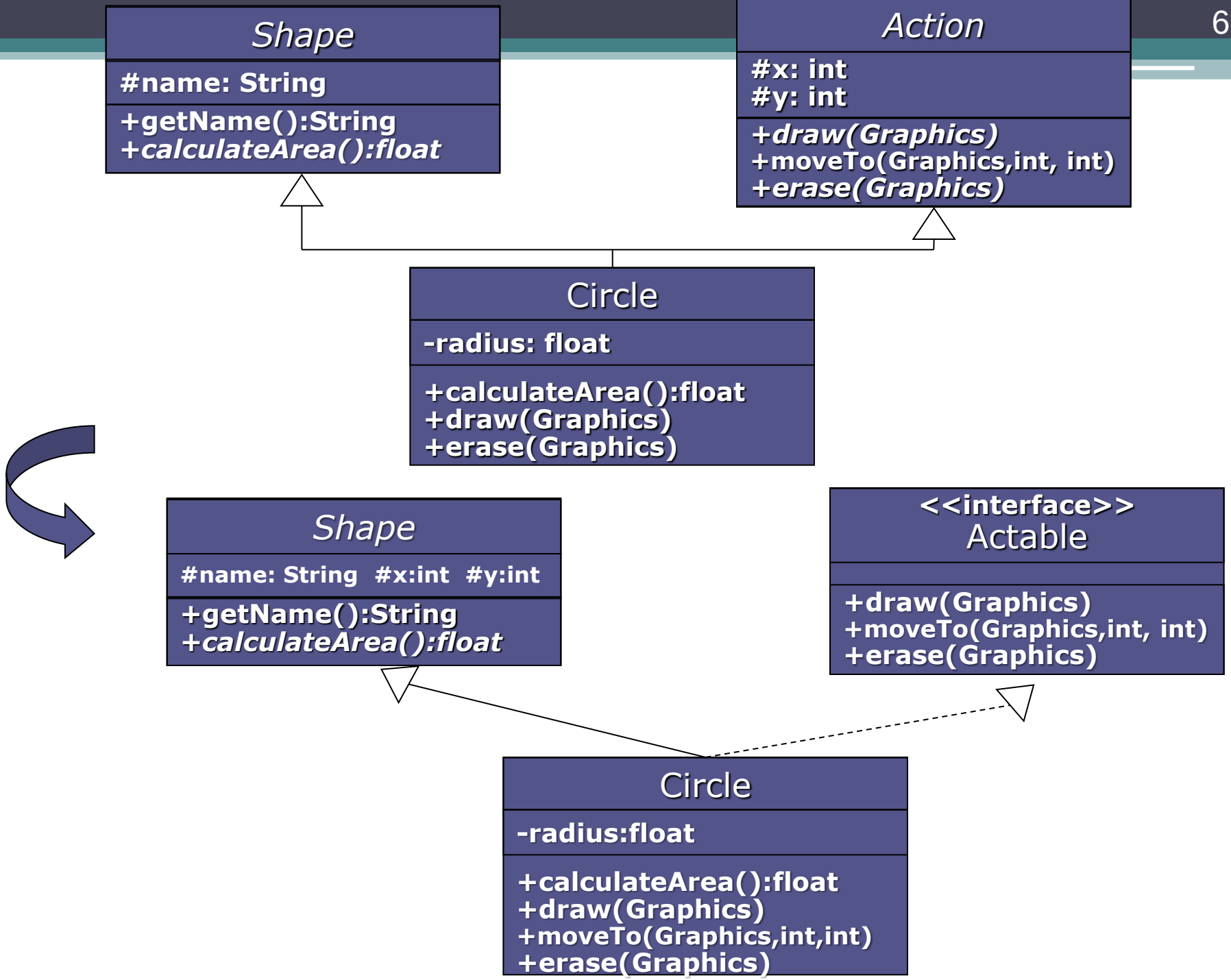
- GPS bằng cách nào có thể điều khiển ô tô lẫn phi thuyền ?

Interface OperateCar

```
public interface OperateCar {

  // Khai báo hằng – nếu có

  // chữ ký phương thức
  int turn(Direction direction,   // An enum with values RIGHT, LEFT
           double radius, double startSpeed, double endSpeed);
  int changeLanes(Direction direction, double startSpeed, double
   endSpeed);
  int signalTurn(Direction direction, boolean signalOn);
  int getRadarFront(double distanceToCar, double speedOfCar);
  int getRadarRear(double distanceToCar, double speedOfCar);
      ......
  // chữ ký phương thức khác
}
```

Lớp OperateBMW760i
// Nhà sản xuất xe hơi

```
public class OperateBMW760i implements OperateCar {

  // cài đặt hợp đồng định nghĩa trong giao diện
  int signalTurn(Direction direction, boolean signalOn) {
     //code to turn BMW's LEFT turn indicator lights on
     //code to turn BMW's LEFT turn indicator lights off
     //code to turn BMW's RIGHT turn indicator lights on
     //code to turn BMW's RIGHT turn indicator lights off
  }

  // Các phương thức khác, trong suốt với các clients của
  interface
  }
```


Shape
#name: String
+getName():String
+calculateArea():float
Action
#x: int
#y: int
+draw(Graphics)
+moveTo(Graphics,int, int)
+erase(Graphics)
Circle
-radius: float
+calculateArea():float
+draw(Graphics)
+erase(Graphics)
Shape
#name: String #x:int #y:int
+getName():String
+calculateArea():float
<<interface>>
Actable
+draw(Graphics)
+moveTo(Graphics,int, int)
+erase(Graphics)
Circle
-radius:float
+calculateArea():float
+draw(Graphics)
+moveTo(Graphics,int,int)
+erase(Graphics)

4. Giao diện

- Cho phép một lớp có thể kế thừa (thực thi - implement) nhiều giao diện một lúc.
- Không thể thể hiện hóa (instantiate) trực tiếp

Giao diện – góc nhìn kỹ thuật (JAVA)

- Một interface có thể được coi như một dạng "class" mà
  - Phương thức và thuộc tính là public không tường minh
  - Các thuộc tính là static và final
  - Các phương thức là abstract

4. Giao diện (2)

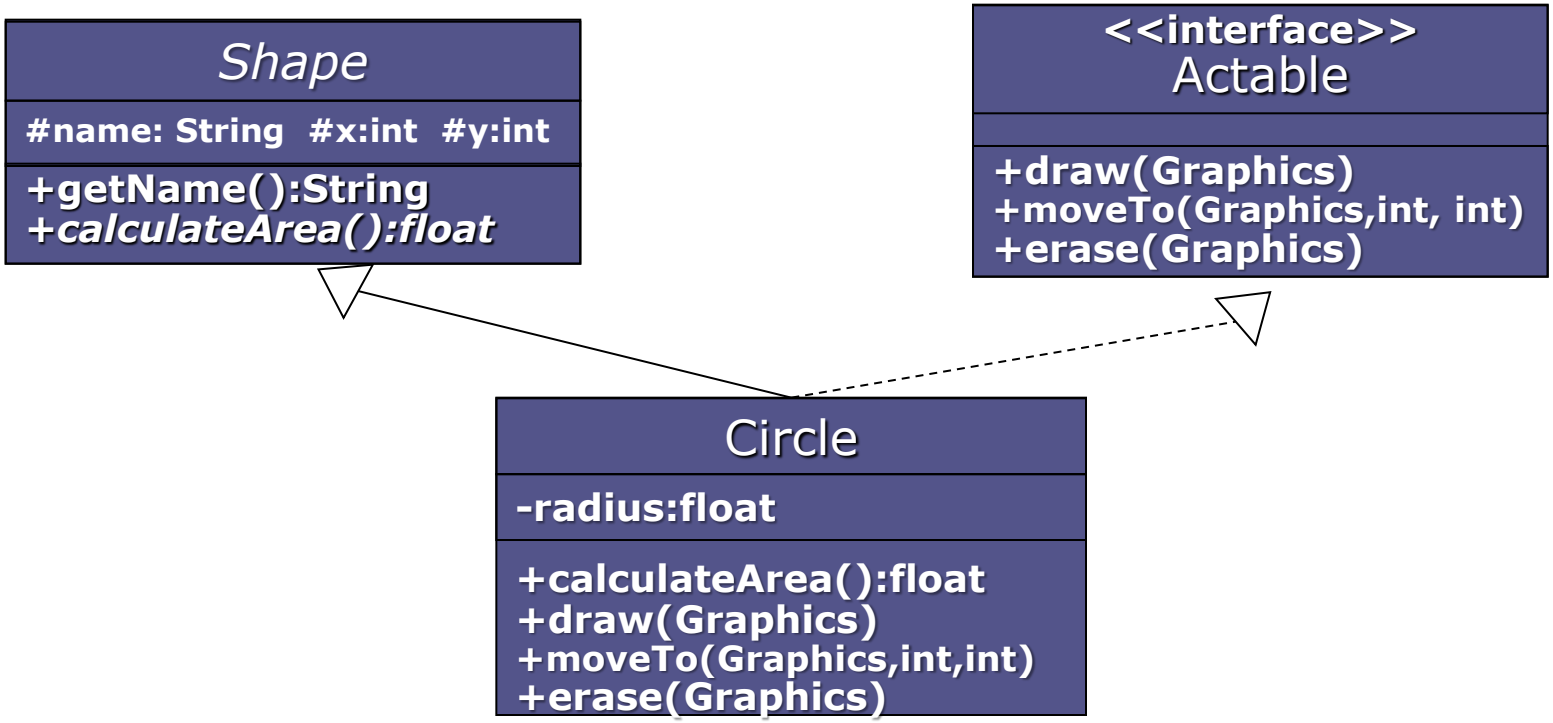
- Để trở thành giao diện, cần
  - Sử dụng từ khóa interface để định nghĩa
  - Chỉ được bao gồm:
    - Chữ ký các phương thức (method signature)
    - Các thuộc tính khai báo hằng (static & final)
- Lớp thực thi giao diện
  - Hoặc là lớp trừu tượng (abstract class)
  - Hoặc là bắt buộc phải cài đặt chi tiết toàn bộ các phương thức trong giao diện nếu là lớp instance.

4. Giao diện (3)

- Cú pháp thực thi trên Java:
  - <Lớp con> [***extends*** <Lớp cha>] ***implements*** <Danh sách giao diện>
  - <Giao diện con> ***extends*** <Giao diện cha>
- Ví dụ:

```
public interface DoiXung {…}
public interface DiChuyen {…}
public class HinhVuong extends TuGiac
                    implements DoiXung, DiChuyen {
  ...
}
```

Ví dụ


Shape
#name: String #x:int #y:int
+getName():String
+calculateArea():float
<<interface>>
Actable
+draw(Graphics)
+moveTo(Graphics,int, int)
+erase(Graphics)
Circle
-radius:float
+calculateArea():float
+draw(Graphics)
+moveTo(Graphics,int,int)
+erase(Graphics)

```
import java.awt.Graphics;
abstract class Shape {
  protected String name;
   protected int x, y;
   Shape(String n, int x, int y) {
       name = n; this.x = x; this.y = y;
   }
   public String getName() {
       return name;
   }
   public abstract float calculateArea();
}
interface Actable {
   public void draw(Graphics g);
   public void moveTo(Graphics g, int x1, int y1);
   public void erase(Graphics g);
}
```

```
class Circle extends Shape implements Actable {
  private int radius;
  public Circle(String n, int x, int y, int r){
      super(n, x, y); radius = r;
  }
  public float calculateArea() {
      float area = (float) (3.14 * radius * radius);
      return area;
  }
  public void draw(Graphics g) {
     System out println("Draw circle at ("
                              + x + "," + y + ")");
     g.drawOval(x-radius,y-radius,2*radius,2*radius);
  }
  public void moveTo(Graphics g, int x1, int y1){
      erase(g); x = x1; y = y1; draw(g);
  }
  public void erase(Graphics g) {
       System out println("Erase circle at ("
                              + x + "," + y + ")");
       // paint the region with background color...
  }
}
```

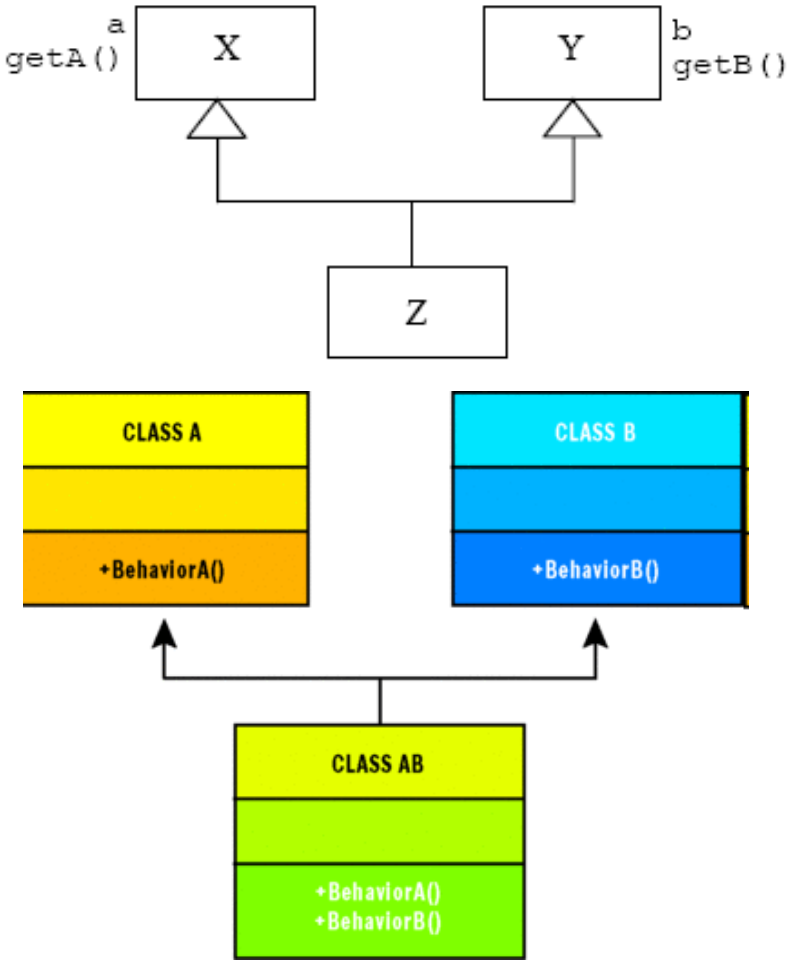
Lớp trừu trượng vs. Giao diện

- Cần có ít nhất một phương thức abstract, có thể chứa các phương thức instance
- Có thể chứa các phương thức protected và static
- Có thể chứa các thuộc tính final và non-final
- Một lớp chỉ có thể kế thừa một lớp trừu tượng

- Chỉ có thể chứa chữ ký phương thức (danh sách các phương thức)
- Chỉ có thể chứa các phương thức public mà không có mã nguồn
- Chỉ có thể chứa các thuộc tính hằng
- Một lớp có thể thực thi (kế thừa) nhiều giao diện

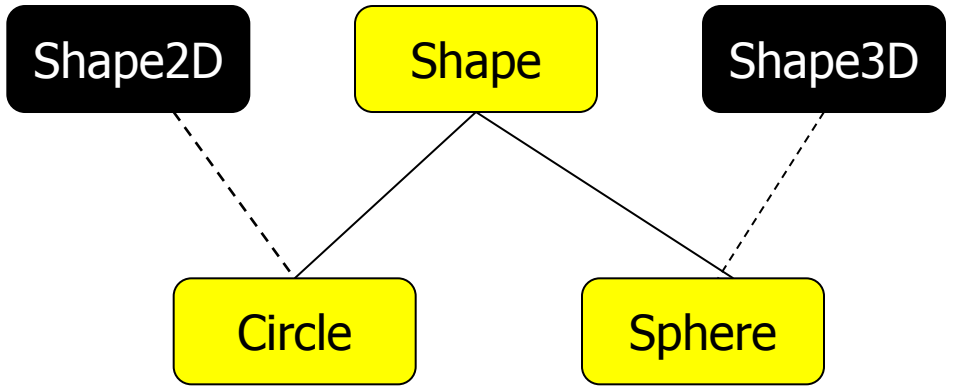
Nhược điểm của Giao diện để giải quyết vấn đề Đa kế thừa

- Không cung cấp một cách tự nhiên cho các tình huống không có sự đụng độ về kế thừa xảy ra
- Kế thừa là để Tái sử dụng mã nguồn nhưng Giao diện không làm được điều này



Ví dụ

```
interface Shape2D {
  double getArea();
}

interface Shape3D {
  double getVolume();
}

class Point3D {
  double x, y, z;

  Point3D(double x, double y, double z) {
    this.x = x;
    this.y = y;
    this.z = z;
  }
}
```



```
abstract class Shape {
  abstract void display();
}

class Circle extends Shape
implements Shape2D {
  Point3D center, p; // p is an point on
circle

  Circle(Point3D center, Point3D p) {
    this.center = center;
    this.p = p;
  }

  public void display() {
    System.out.println("Circle");
  }

  public double getArea() {
    double dx = center.x - p.x;
    double dy = center.y - p.y;
    double d = dx * dx + dy * dy;
    double radius = Math.sqrt(d);
    return Math.PI * radius * radius;
  }
}
```



```
class Sphere extends Shape
implements Shape3D {
  Point3D center;
  double radius;

  Sphere(Point3D center, double radius) {
    this.center = center;
    this.radius = radius;
  }

  public void display() {
    System.out.println("Sphere");
  }

  public double getVolume() {
    return 4 * Math.PI * radius * radius * radius / 3;
  }
}

class Shapes {

  public static void main(String args[]) {

    Circle c = new Circle(new Point3D(0, 0, 0), new
      Point3D(1, 0, 0));
    c.display();
    System.out.println(c.getArea());
    Sphere s = new Sphere(new Point3D(0, 0, 0), 1);
    s.display();
    System.out.println(s.getVolume());
  } }
```

Result :

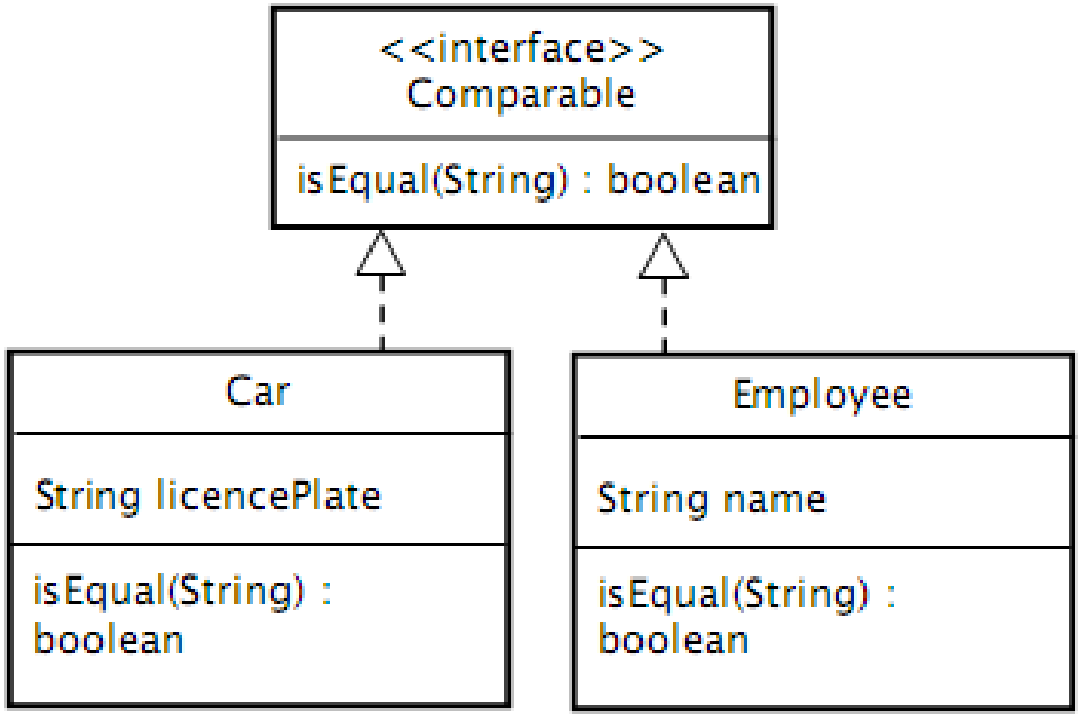
Circle

3.141592653589793

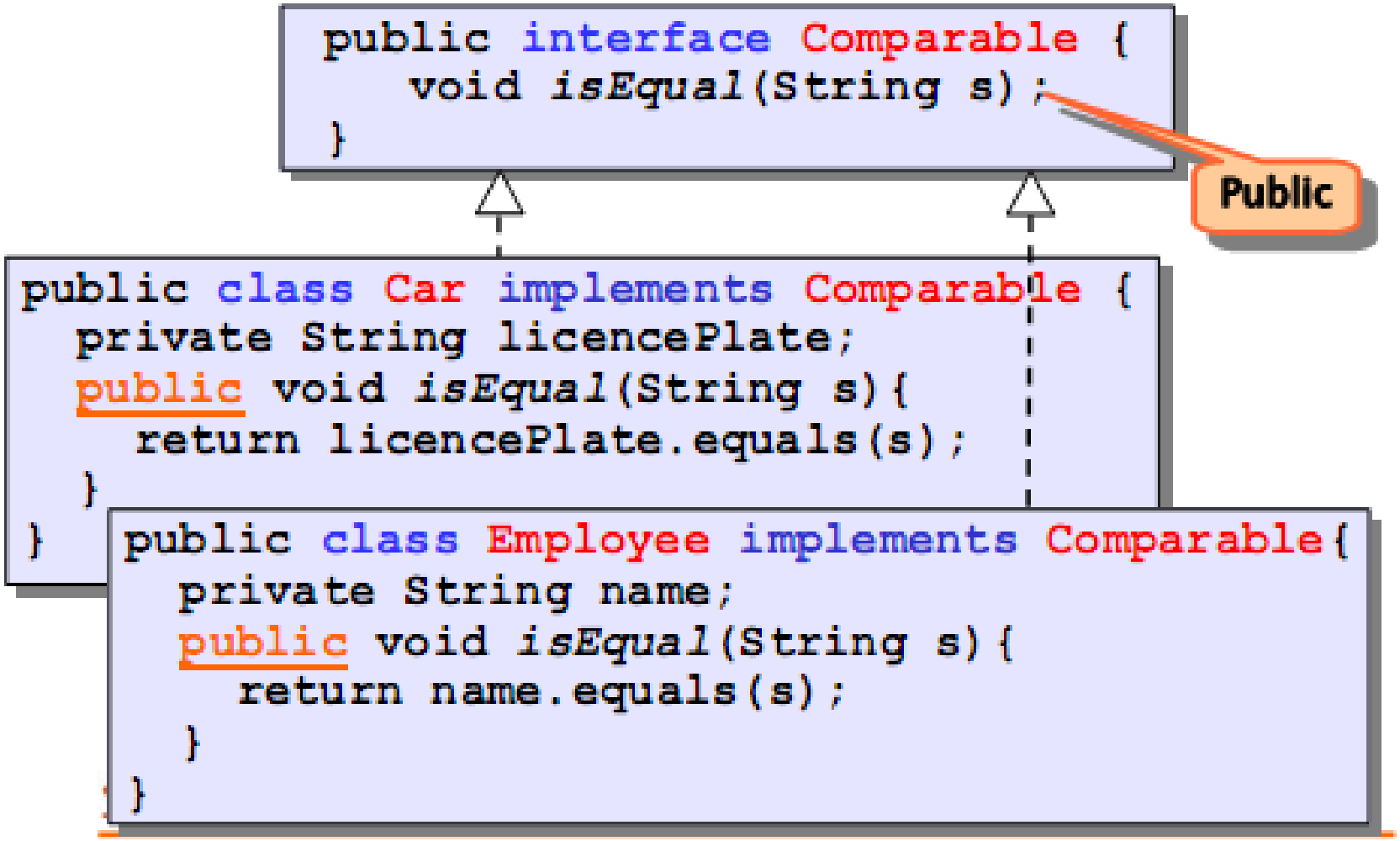
Sphere

4.1887902047863905

interface Comparable /java.lang



Ứng dụng


public interface Comparable {
void isEqual(String s);
}
Public
public class Car implements Comparable {
private String licencePlate;
public void isEqual(String s){
return licencePlate.equals(s);
}
}
public class Employee implements Comparable{
private String name;
public void isEqual(String s){
return name.equals(s);
}
}

Ứng dụng

```
public class Foo {
   private Comparable objects[];
   public Foo(){
      objects = new Comparable[3];
      objects[0] = new Employee();
      objects[1] = new Car();
      objects[2] = new Employee();
   }
   public Comparable find(String s){
      for(int i=0; i< objects.length; i++)
        if(objects[i].isEqual(s)
                          return objects[i];
   }
}
```