ĐẠI NAM
UNIVERSITY

ĐÓNG GÓI

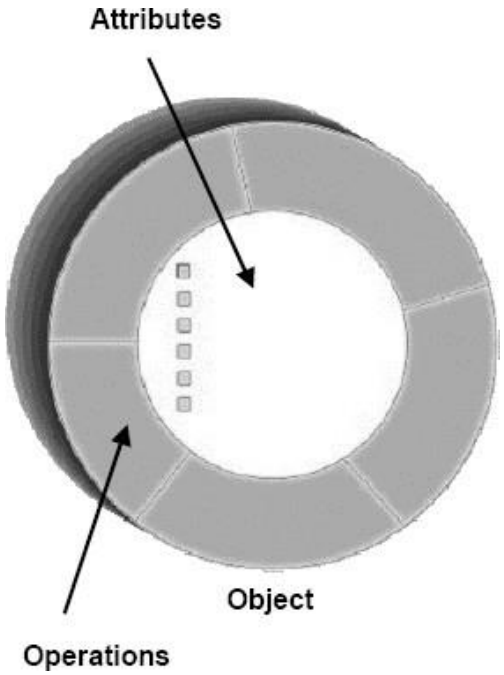
Thời gian: 4 tiết

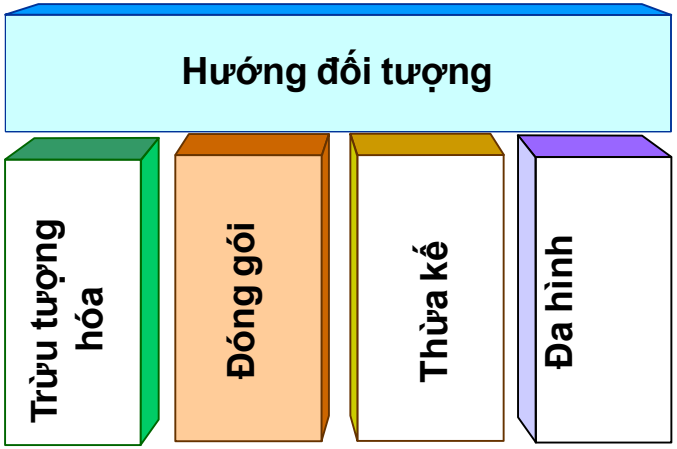
NỘI DUNG

- ✓ Đóng gói và xây dựng lớp
- ✓ Khai báo và sử dụng đối tượng
- ✓ Truyền tham số cho phương thức

Đóng gói (Encapsulation)

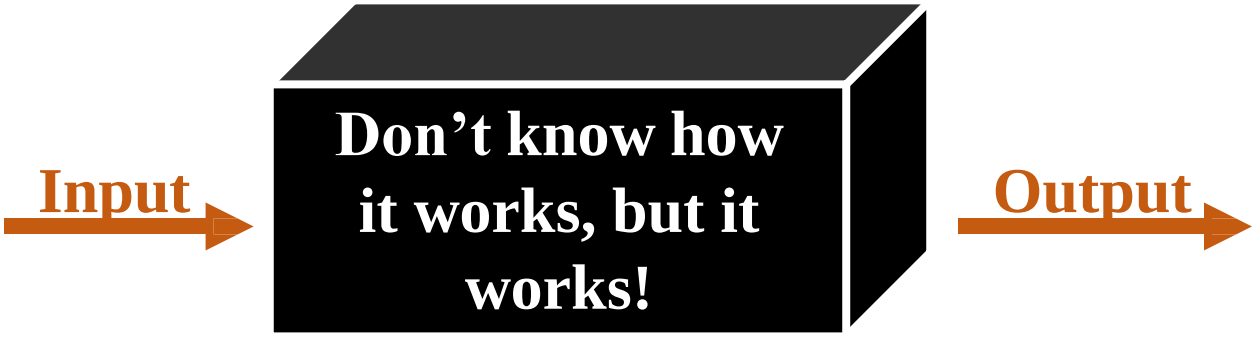
- Là một trong 4 nguyên lý cơ bản của lập trình HĐT.
- Dữ liệu/thuộc tính và hành vi/phương thức được đóng gói trong một lớp.





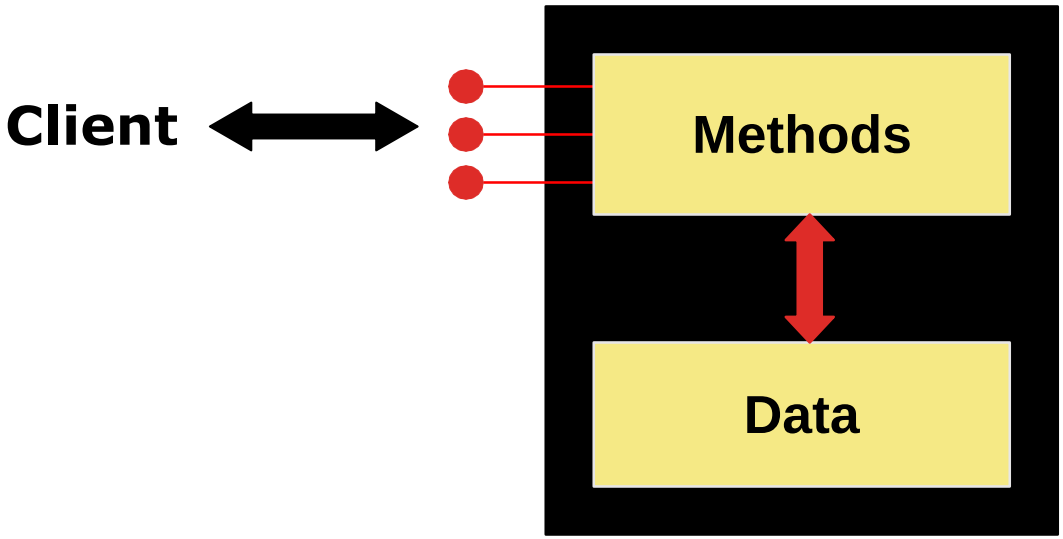
Đóng gói

- Một đối tượng là một thực thể được đóng gói với mục đích:
  - Cung cấp tập các dịch vụ nhất định
  - Đối tượng được đóng gói có thể được xem như một hộp đen – các công việc bên trong là ẩn so với client
  - Dù thay đổi thiết kế/mã nguồn bên trong nhưng giao diện bên ngoài không bị thay đổi theo



Đóng gói

- Sau khi đóng gói, một đối tượng có hai khung nhìn:
  - Bên trong: Chi tiết về các thuộc tính và các phương thức của lớp tương ứng với đối tượng
  - Bên ngoài: Các dịch vụ mà một đối tượng có thể cung cấp và cách đối tượng đó tương tác với phần còn lại của hệ thống

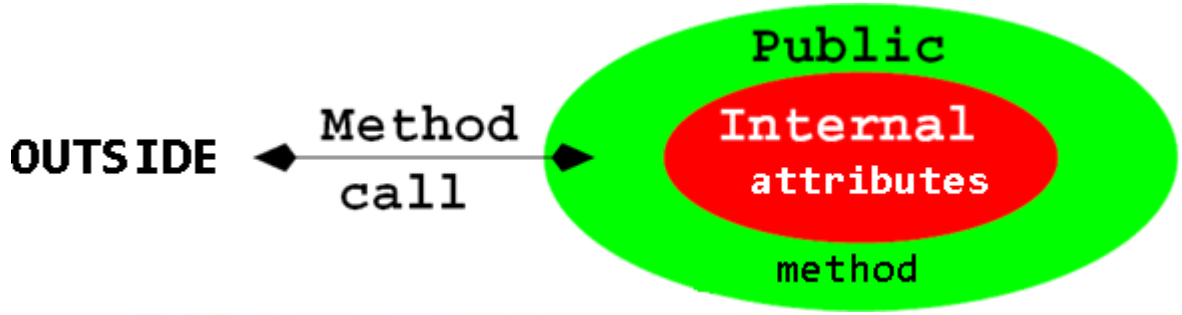


Phạm vi truy cập

- Phạm vi truy cập xác định khả năng nhìn thấy được của một thành phần của chương trình với các thành phần khác của chương trình
- Đối với lớp
  - Phạm vi truy cập có thể được áp dụng cho các thành phần của lớp
    - **private**: chỉ truy cập được từ bên trong lớp đó
    - **public**: có thể truy cập được tại mọi nơi, từ trong và ngoài lớp

Che giấu dữ liệu

- Sử dụng phạm vi truy cập để che giấu dữ liệu: tránh thay đổi trái phép hoặc làm sai lệch dữ liệu
- Dữ liệu được che giấu ở bên trong lớp bằng cách gán phạm vi truy cập *private*.
  - chỉ có thể truy cập từ các phương thức bên trong lớp
- Các đối tượng khác muốn truy nhập vào dữ liệu riêng tư này phải thông qua các phương thức của lớp có phạm vi truy cập *public*.



Phạm vi truy cập thuộc tính/phương thức

- **public**: Thuộc tính hoặc phương thức có thể được truy cập từ bất cứ đâu, kể cả bên ngoài lớp, ngoài gói chứa lớp đó.
- **default**: Thuộc tính hoặc phương thức chỉ có thể được truy cập từ bên trong package chứa lớp đó.
- **private**: Thuộc tính hoặc phương thức chỉ có thể được truy cập trong phạm vi lớp đó
- **protected**: Thuộc tính hoặc phương thức chỉ có thể được truy cập trong phạm vi lớp đó và từ lớp con kế thừa của lớp đó.

Access Levels

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Ví dụ - Khai báo lớp

```
class BankAccount  {
        private String owner;
        private double balance;
        public boolean debit(double amount) {
              if (amount > balance)
                    return false;
              else {
                    balance -= amount;
                    return true;
              }
        }
        public void credit(double amount) {
              balance += amount;
        }
}
```

Ví dụ: phương thức get, set

```
class Student{
    private String name;
    public String getName() {
        return this.name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

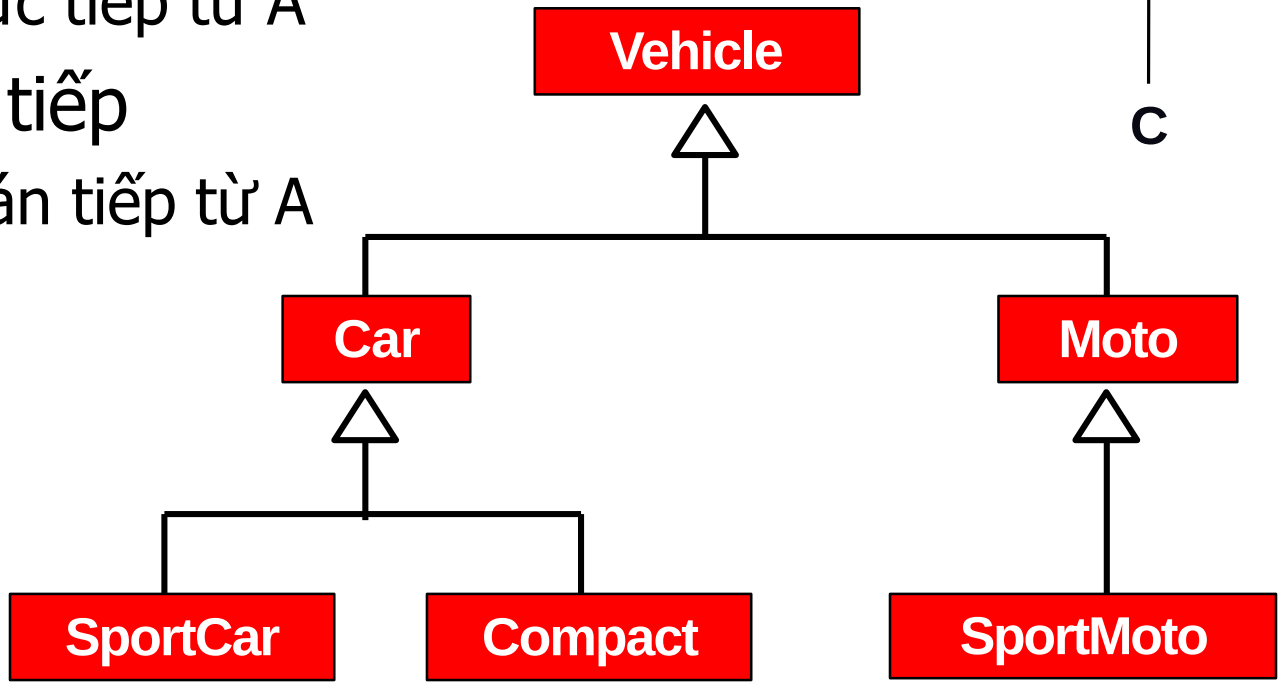
Ví dụ: phương thức get, set

```
class Student{
    private String name;
    public String getName() {
        return this.name;
    }
    public void setName(String name)
    {
        this.name = name;
    }
}
```

```
class Manager{
    private Student[] students;
    public initialize()
    {
        students = new Student[10];
        students[0] = new Student();
        //students[0].name = "Hung"; error
        students[0].setName("Hung");
    }
}
```

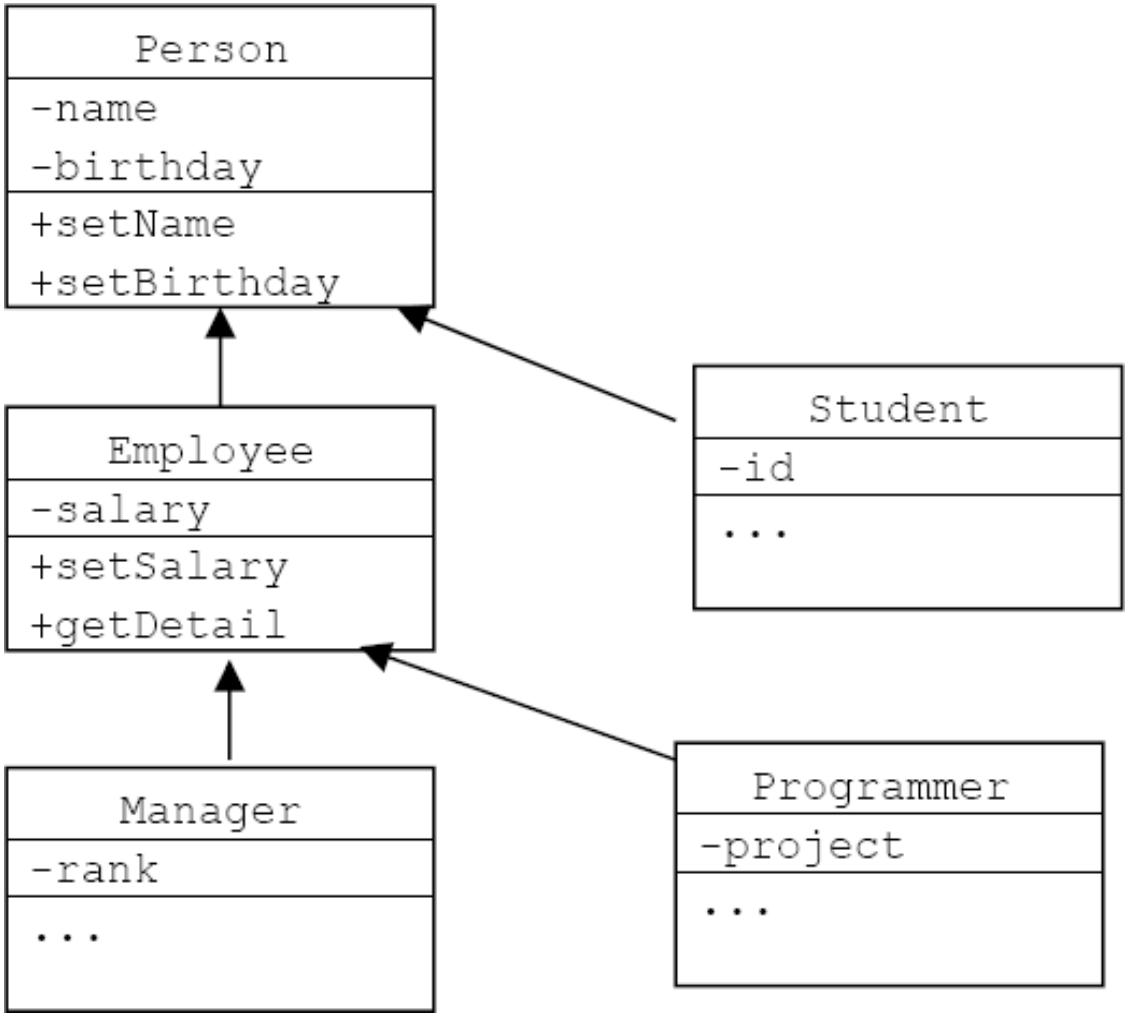
Cây phân cấp kế thừa

- Cấu trúc phân cấp hình cây, biểu diễn mối quan hệ kế thừa giữa các lớp.
- Dẫn xuất trực tiếp
  - B dẫn xuất trực tiếp từ A
- Dẫn xuất gián tiếp
  - C dẫn xuất gián tiếp từ A



Cây phân cấp kế thừa

Mọi lớp
đều kế thừa từ
lớp gốc Object



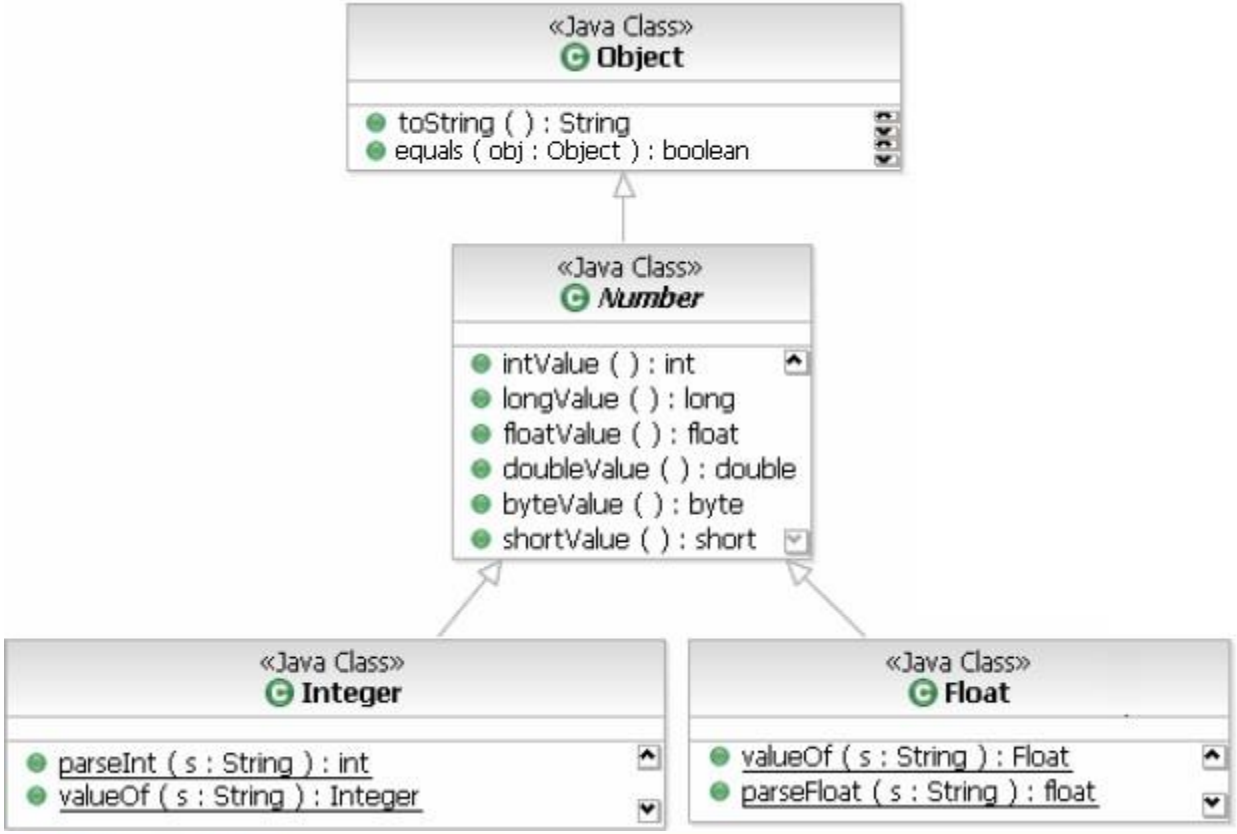
Lớp Object

- Trong gói java.lang
- Nếu một lớp không được định nghĩa là lớp con của một lớp khác thì mặc định nó là lớp con trực tiếp của lớp Object.

  → Lớp Object là lớp gốc trên cùng của tất cả các cây phân cấp kế thừa

Lớp Object

- Chứa một số phương thức hữu ích kế thừa lại cho tất cả các lớp, ví dụ: toString(), equals()...



Cú pháp kế thừa

- Cú pháp kế thừa trên Java:

```
<Lớp con> extends <Lớp cha>
```

- Lớp cha nếu được định nghĩa là `final` thì không thể có lớp dẫn xuất từ nó.
- Ví dụ:

```
class HinhVuong extends TuGiac {
  ...
}
```

Từ khóa *super*

- Tái sử dụng các đoạn mã của lớp cha trong lớp con
- Gọi phương thức khởi tạo

  `super(danh sách tham số);`

  – Bắt buộc nếu lớp cha không có phương thức khởi tạo mặc định
- Gọi các phương thức của lớp cha

  `super.tênPt(danh sách tham số);`

Ví dụ

```
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;
    }
}
```

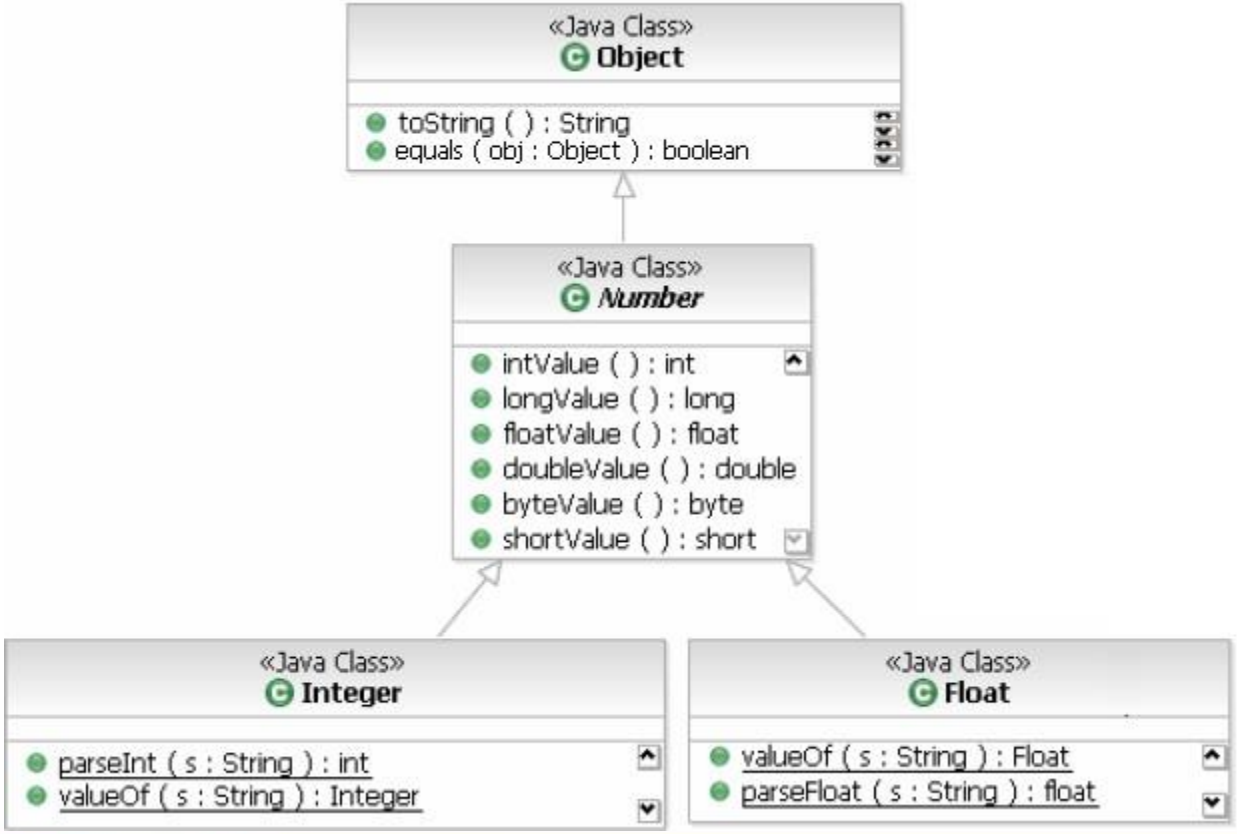
Lớp Object

- Trong gói java.lang
- Nếu một lớp không được định nghĩa là lớp con của một lớp khác thì mặc định nó là lớp con trực tiếp của lớp Object.

  → Lớp Object là lớp gốc trên cùng của tất cả các cây phân cấp kế thừa

Lớp Object

- Chứa một số phương thức hữu ích kế thừa lại cho tất cả các lớp, ví dụ: toString(), equals()...

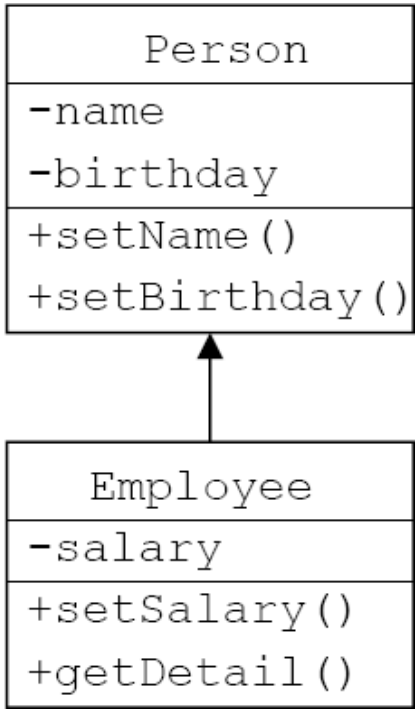


```
public class TuGiac {
        protected Diem d1, d2, d3, d4;
        public void printTuGiac() {...}
        public TuGiac() {...}
        public TuGiac(Diem d1, Diem d2, Diem d3, Diem d4) {...}
}
public class HinhVuong extends TuGiac {
        public HinhVuong(){
                d1 = new Diem(0,0); d2 = new Diem(0,1);
                d3 = new Diem(1,0); d4 = new Diem(1,1);
        }
}
public class Test {
        public static void main(String args[]) {
                HinhVuong hv = new HinhVuong();
                hv.printTuGiac();
        }
}
```

Ví dụ

protected

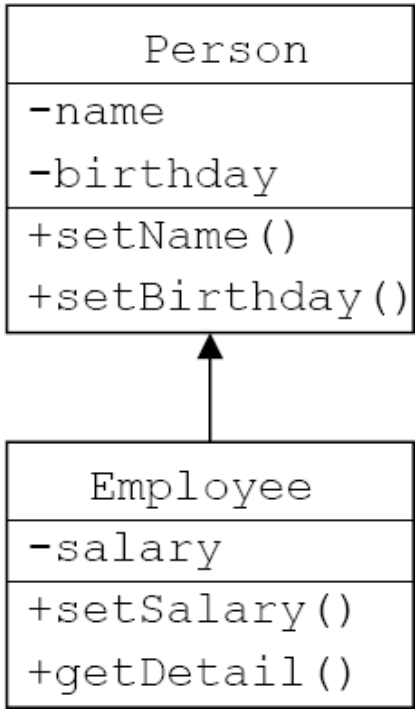
```
class Person {
   private String name;
   private Date birthday;
   public String getName() {return name;}
   ...
}
class Employee extends Person {
   private double salary;
   public boolean setSalary(double sal){
    salary = sal;
    return true;
   }
   public String getDetail(){
    String s = name+", "+birthday+", "+salary; //Loi
    }
}
```



Ví dụ

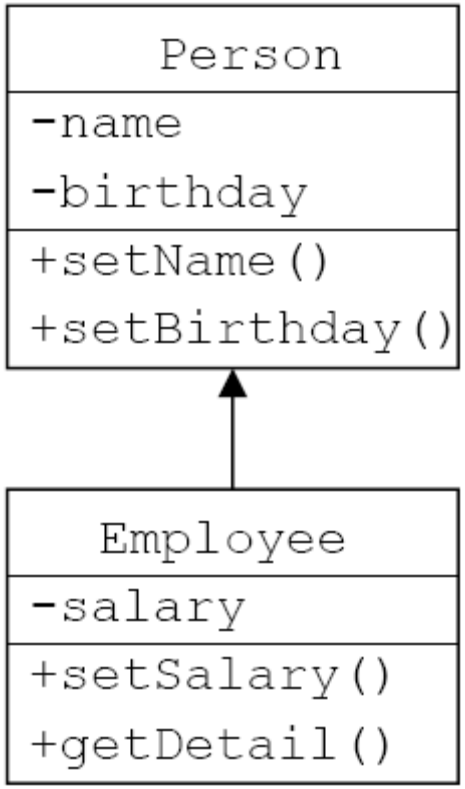
protected

```
class Person {
   protected String name;
   protected Date bithday;
   public String getName() {return name;}
   ...
}
class Employee extends Person {
   private double salary;
   public boolean setSalary(double sal){
    salary = sal;
    return true;
   }
   public String getDetail(){
    String s = name+", "+birthday+", "+salary;
   }
}
```



Ví dụ

```
public class Test {
  public static void main(String args[]){
      Employee e = new Employee();
      e.setName("John");
      e.setSalary(3.0);
  }
}
```



Ví dụ – Cùng gói


```
public class Person {
        Date birthday;
        String name;
        ...
}
public class Employee extends Person {
        ...
        public String getDetail() {
                String s;
                s = name + "," + birthday;
                s += ", " + salary;
                return s;
        }
}
```

Ví dụ – Khác gói

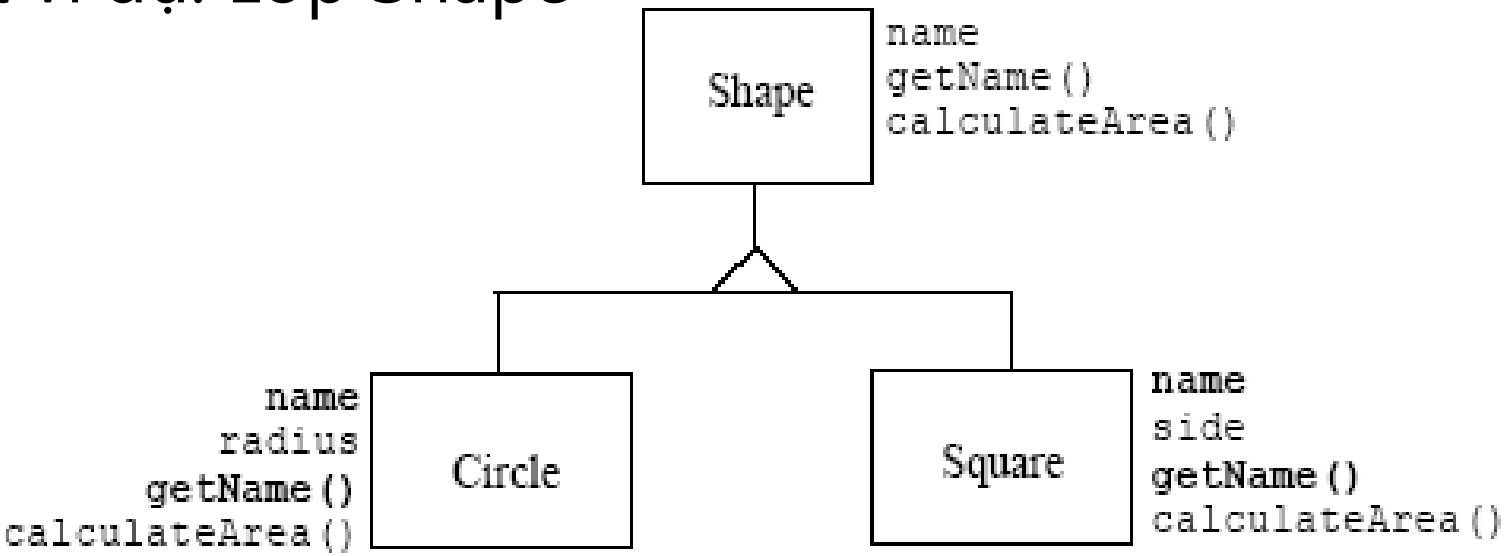
```
package abc;
public class Person {
        protected Date birthday;
        protected String name;
        ...
}
import abc.Person;
public class Employee extends Person {
        ...
        public String getDetail() {
                String s;
                s = name + "," + birthday + "," + salary;
                return s;
        }
 }
```

Khởi tạo và huỷ bỏ đối tượng

- Khởi tạo đối tượng:
  - Lớp cha được khởi tạo trước lớp con.
  - Các phương thức khởi tạo của lớp con luôn gọi phương thức khởi tạo của lớp cha ở câu lệnh đầu tiên
    - Tự động gọi (không tường minh - implicit): Khi lớp cha có phương thức khởi tạo mặc định
    - Gọi trực tiếp (tường minh - explicit)
- Hủy bỏ đối tượng:
  - Ngược lại so với khởi tạo đối tượng

Lớp trừu tượng

- Xét ví dụ: Lớp Shape



- Là một lớp "không rõ ràng", khó hình dung ra các đối tượng cụ thể
- Không thể thể hiện hóa (instantiate – tạo đối tượng của lớp) trực tiếp

Lớp trừu tượng

- Đặc điểm của lớp trừu tượng
  - Không thể tạo đối tượng trực tiếp từ các lớp trừu tượng
  - 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.
  - Chưa đầy đủ, thường được sử dụng làm lớp cha.
  - Lớp con kế thừa nó sẽ hoàn thiện nốt.
  - Lớp trừu tượng thường chứa các phương thức trừu tượng không được định nghĩa

Phương thức trừu tượng

- Là các phương thức "không rõ ràng", khó đưa ra cách cài đặt cụ thể
- Chỉ có chữ ký mà không có cài đặt cụ thể
- Các lớp dẫn xuất có thể làm rõ - định nghĩa lại (overriding) các phương thức trừu tượng này

Cú pháp

- Lớp trừu tượng
  - Khai báo với từ khóa abstract

```
public abstract class Shape {
    // Nội dung lớp
}
```

- Phương thức trừu tượng
  - Khai báo với từ khóa abstract

```
public abstract float calculateArea();
// CHÚ Ý
Shape s = new Shape(); //Compile error
```

Ví dụ

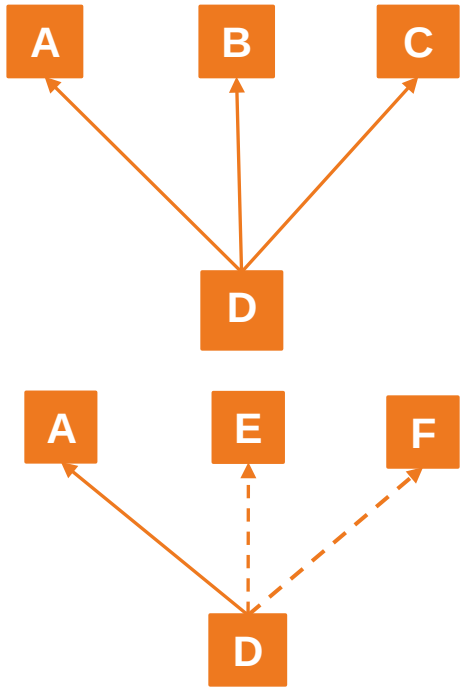
```
abstract class Shape {
    protected String name;
    Shape(String n) { name = n; }
    public String getName() { return name; }
    public abstract float calculateArea();
}

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 cha

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)



Giao diện

- Giao diện (interface)
  - Là kiểu dữ liệu trừu tượng, được dùng để đặc tả các hành vi mà các lớp phải thực thi
  - Tương tự như giao thức (protocols)
  - Chứa các chữ ký phương thức (Mọi phương thức đều là phương thức trừu tượng) và các hằng
  - Giải quyết bài toán đa thừa kế, tránh các rắc rối nhập nhằng ngữ nghĩa
- Giao diện trong Java
  - Được định nghĩa với từ khóa interface
  - Từ Java 8: có thêm phương thức default, static. Từ Java 9, có thêm phương thức private và private static

Giao diện

- 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)
- Cú pháp khai báo giao diện trên Java:

```
interface <Tên giao diện>
<Giao diện con> extends <Giao diện cha>
```

- Ví dụ:

```
public  interface DoiXung {…}
public  interface Can extends DoiXung {…}
public  interface DiChuyen {…}
```

Lớp thực thi giao diện

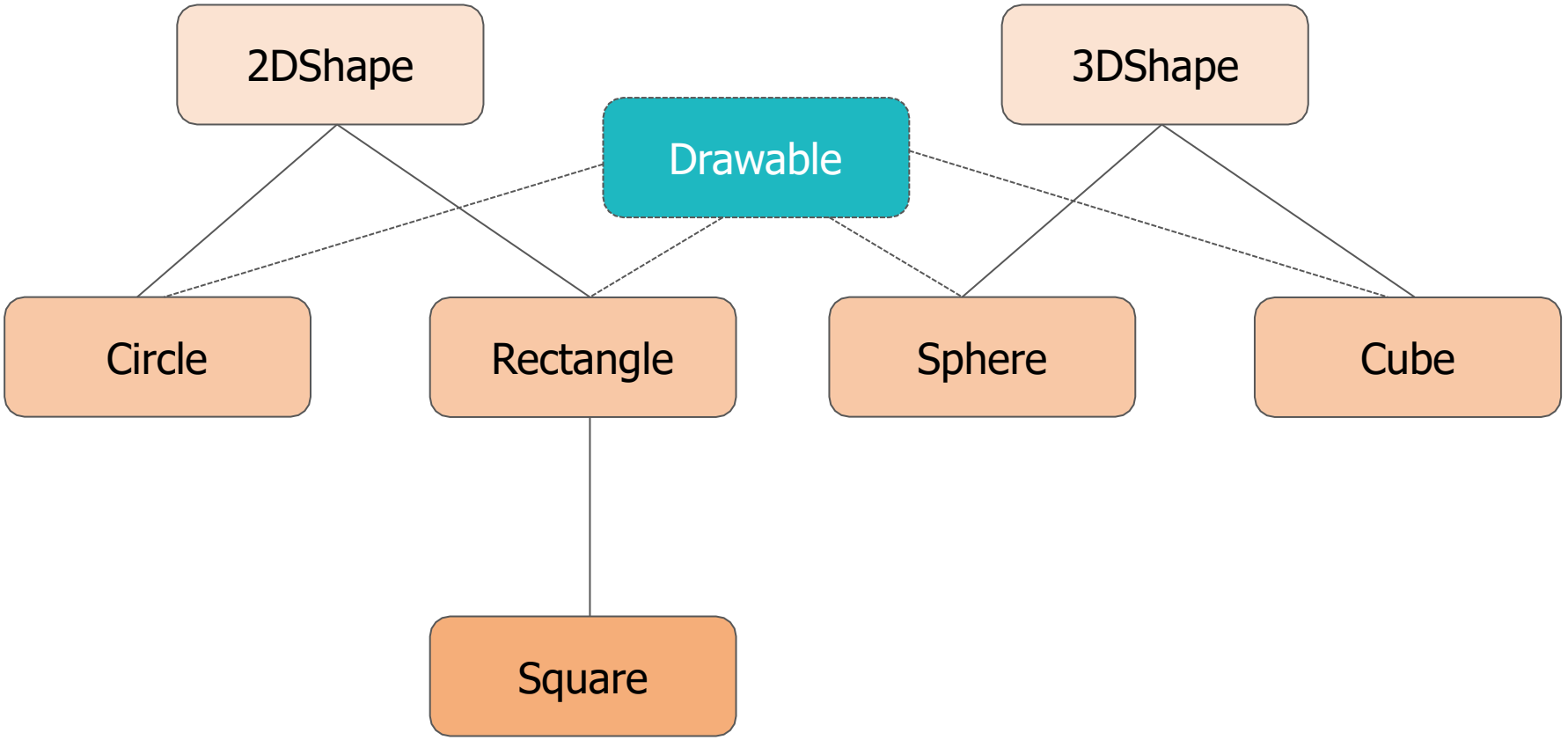
- Lớp thực thi giao diện
  - Hoặc là lớp trừu tượng (abstract class)
  - Hoặc là bắt buộc phải cài đặt chi tiết toàn bộ các phương thức trong giao diện nếu là lớp instance.
- Cú pháp thực thi giao diện

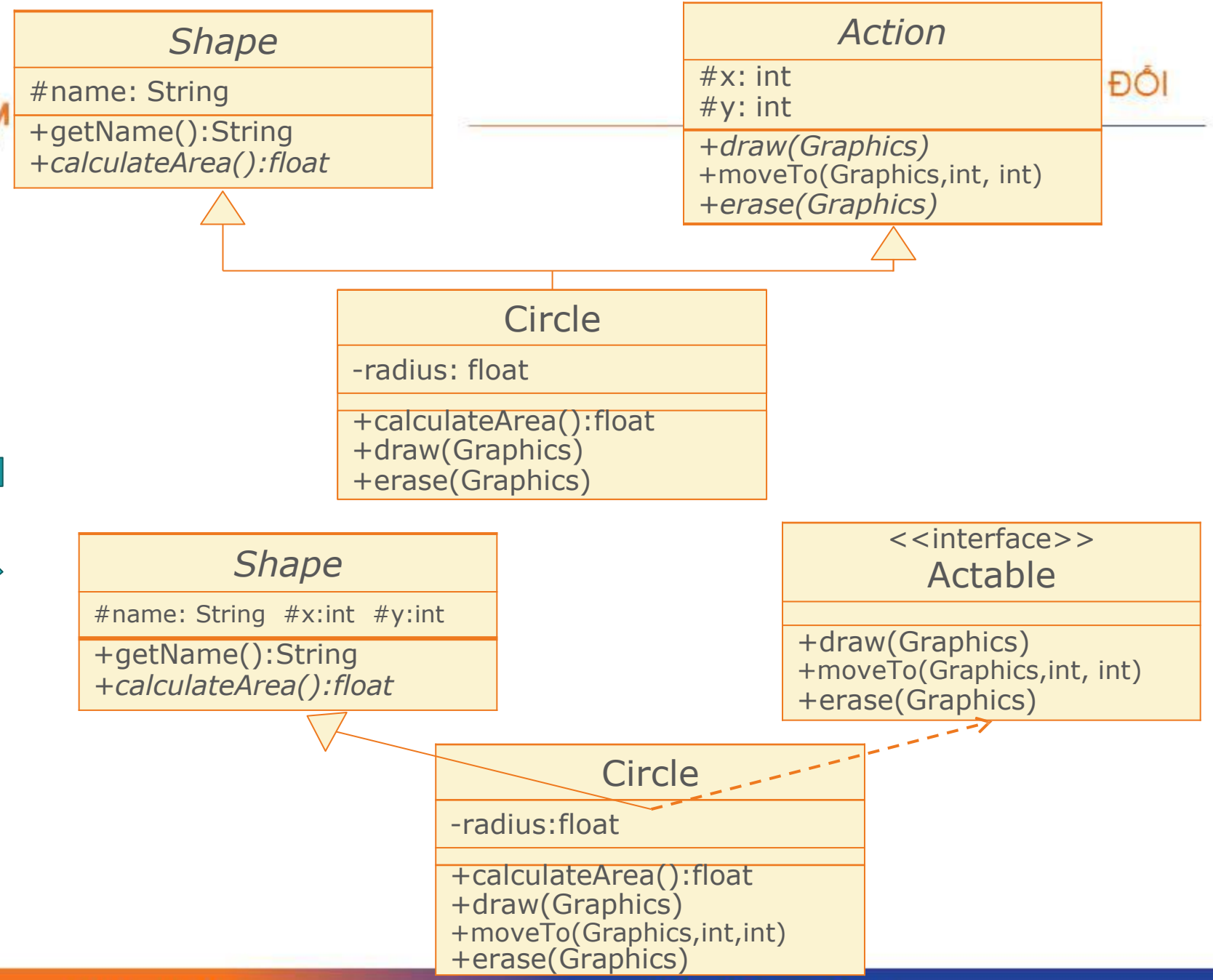
```
<Lớp con> [extends <Lớp cha>] implements <Danh sách giao diện>
```

- Ví dụ:

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

Interface


2DShape
3DShape
Drawable
Circle
Rectangle
Sphere
Cube
Square


Shape
#name: String
+getName():String
+calculateArea():float
Action
#x: int
#y: int
+draw(Graphics)
+moveTo(Graphics,int, int)
+erase(Graphics)
ĐỐI
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)

Ví dụ

```
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...
    }
}
```

Giao diện

- 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
- Không thể thể hiện hóa (instantiate) trực tiếp
- Một lớp có thể thực thi nhiều giao diện

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 – 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 cài đặt chi tiết
  - 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.

Đa thừa kế

```
interface Interface1 {
    default void doSomething() {
        System.out.println("doSomething1");
    }
}

interface Interface2 {
    default void doSomething() {
        System.out.println("doSomething2");
    }
}
public class MultiInheritance implements Interface1, Interface2 {
    @Override
    public void doSomething() {
        Interface1.super.doSomething();
    }
}
```

Đa thừa kế

```
interface Interface3 {
    default void doSomething() {
        System.out.println("Execute in Interface3");
    }
}

class Parent {
    public void doSomething() {
        System.out.println("Execute in Parent");
    }
}

public class MultiInheritance2 extends Parent implements
Interface3 {
    public static void main(String[] args) {
        MultiInheritance2 m = new MultiInheritance2();
        m.doSomething(); // Execute in Parent
    }
}
```

ĐẠI NAM
UNIVERSITY

Xin cảm ơn!

TRẦN QÚY NAM

Giảng viên Khoa CNTT
namtq@dnu.edu.vn