ĐẠI NAM
UNIVERSITY

CHƯƠNG 5
ĐA HÌNH

Giảng viên: Phạm Văn Tiệp

Nội dung

- Đa hình
- Lớp trừu tượng
- Interface

ĐẠI NAM
UNIVERSITY

ĐA HÌNH

http://dainam.edu.vn

Chồng phương thức

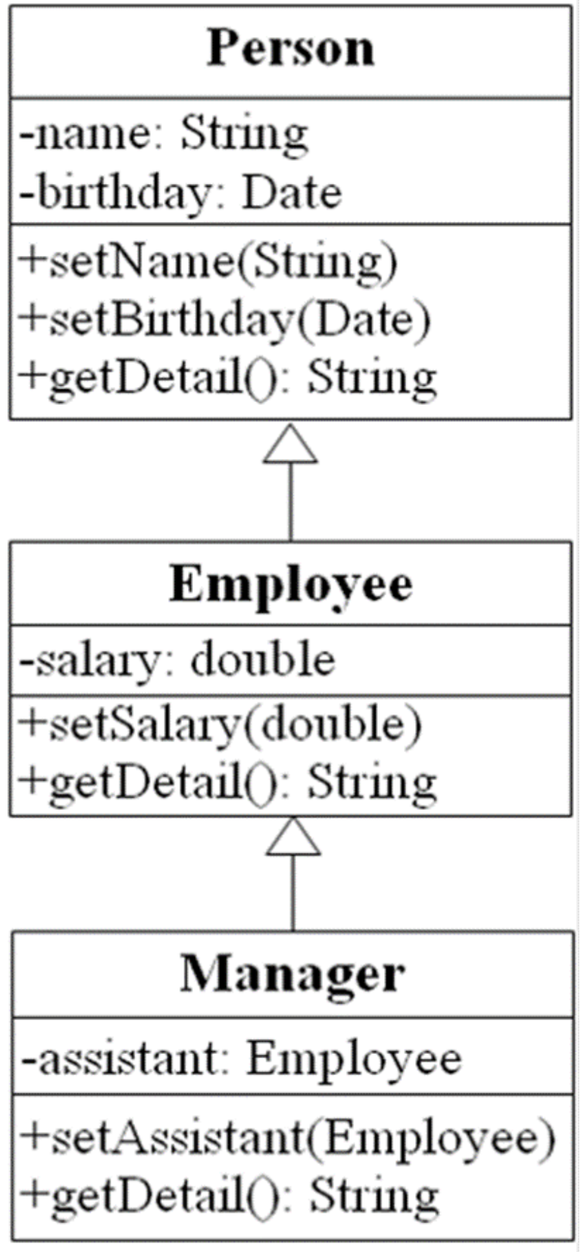
- Nhắc lại: lớp con có thể viết lại phương thức thừa kế từ lớp cha bằng 2 cách thức:
  - **Chồng phương thức** (Overloading): giữ tên và giá trị trả về, thay đổi đối số
  - **Ghi đè phương thức** (Overriding): giữ nguyên tên, giá trị trả về và đối số
- Chồng phương thức có thể thực hiện ngay trong chính 1 lớp:
  - Ví dụ: Viết các phương thức khởi tạo khác nhau

Upcasting và Downcasting

- **Upcasting**: đối tượng lớp con được nhìn nhận như đối tượng lớp cha
  - Dùng đối tượng của lớp con để truyền tham số.
  - Dùng đối tượng của lớp con làm thuộc tính.
  - Thực hiện tự động
- **Downcasting**: đối tượng lớp cha được nhìn nhận như đối tượng lớp con
  - Phải ép kiểu

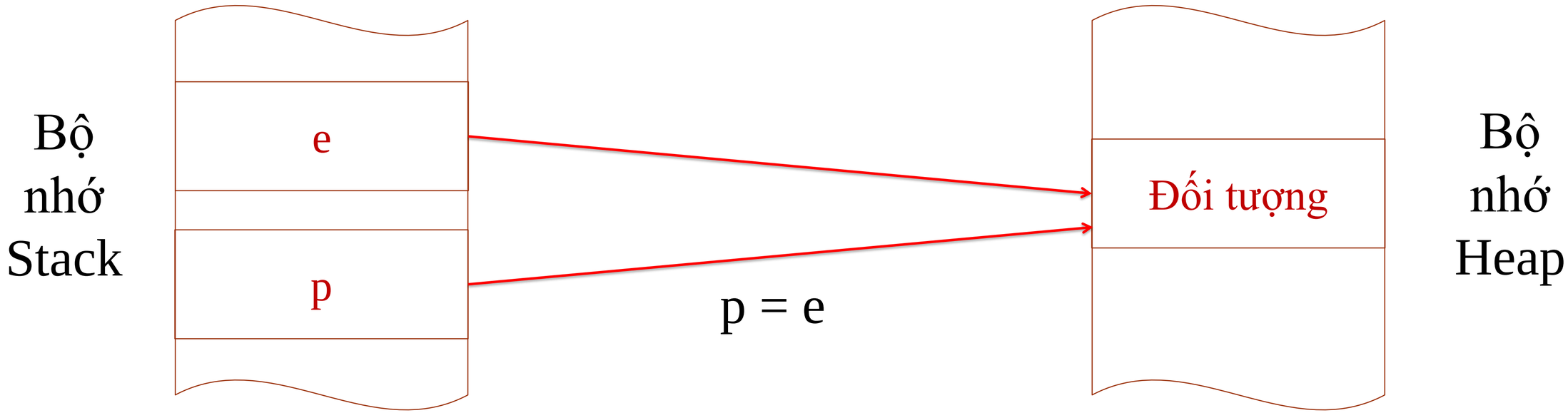
```
public static void main(string  args[])
{
    Person p1 = new Employee();
    Person p2 = new Manager();

    Employee e = (Employee) p1;
    Manager m = (Manager) p2;
}
```



Upcasting – Ví dụ

```
Person p;
Employee e = new Employee();
p = e; //upcasting
p.setName();//OK
p.setSalary();//compile error
```



Upcasting – Ví dụ

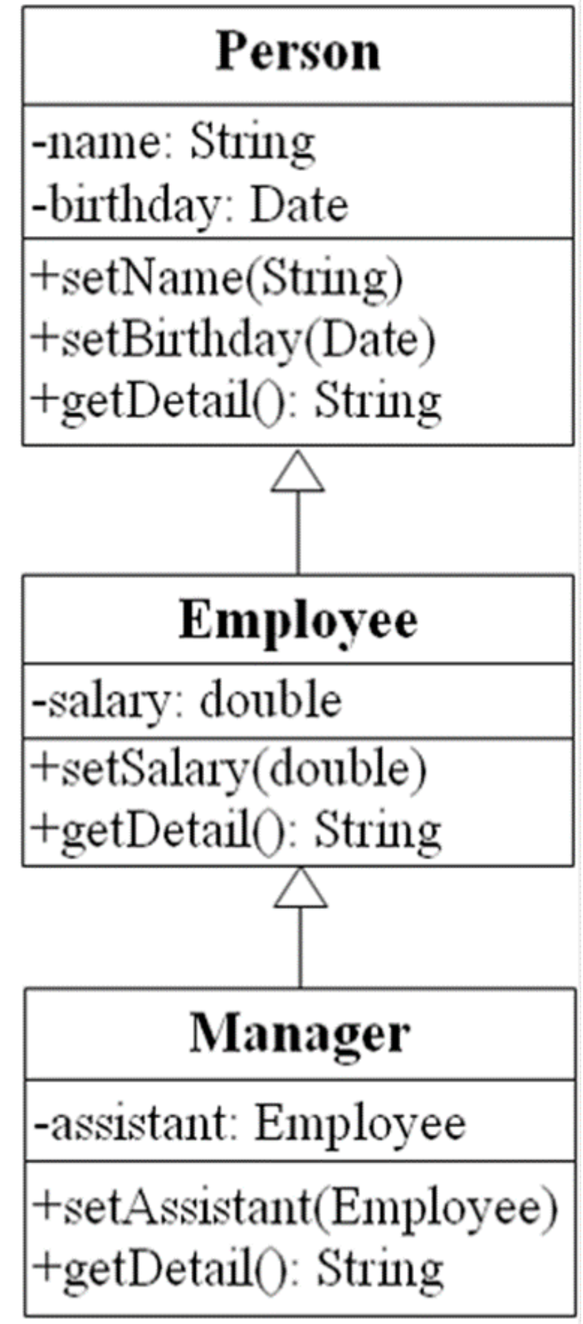
```
public class Test3 {
string teamInfo(Person p1, Person p2)
{
    return "Leader: " + p1.getName() + "; member: " + p2.getName();
}
public static void main(string arg[]) {
    Employee e1, e2;
    Manager  m1, m2;
    //…
    System.out.println(teamInfo(e1, e2));
    System.out.println(teamInfo(m1, m2));
    System.out.println(teamInfo(m1,e2));
  }
}
```

Upcasting – Ví dụ

```
class Manager : Employee
{
    Employee assistant;
    //...
    public void setAssistant(Employee e)
    {
        assistant = e;
    }
    //...
}
public class Test2 {
    public static void main(String arg[]) {
        Manager junior, senior;
        //...
        senior.setAssistant(junior);
    }
}
```

Downcasting – Ví dụ

```
public class Test2 {
    public static void main(String arg[]) {
        Employee e = new Employee();
        Person p = e;  // up casting
        Employee ee = (Employee)p;  // down casting
        Manager m = (Manager)ee;  //  run-time error
        Person p2 = new Manager();
        Employee e2 = (Employee) p2;
    }
}
```



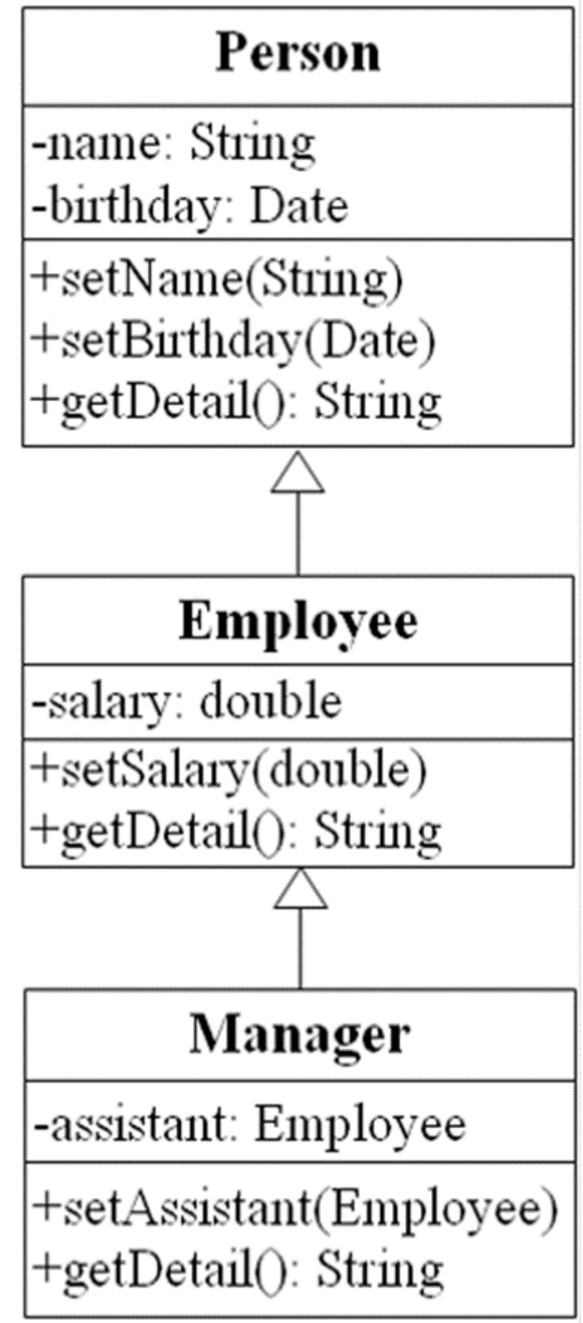
Liên kết tĩnh và liên kết động

Static and dynamic binding

- **Liên kết tĩnh**: Liên kết tại thời điểm biên dịch
  - Early Binding/Compile-time Binding
  - Lời gọi phương thức được quyết định khi biên dịch, do đó chỉ có một phiên bản của phương thức được thực hiện
  - Nếu có lỗi thì sẽ có lỗi biên dịch
  - Ưu điểm về tốc độ
- **Liên kết động**: Lời gọi phương thức được quyết định khi thực hiện (run-time)
  - Late binding/Run-time binding
  - Phiên bản của phương thức phù hợp với đối tượng được gọi.

Ví dụ

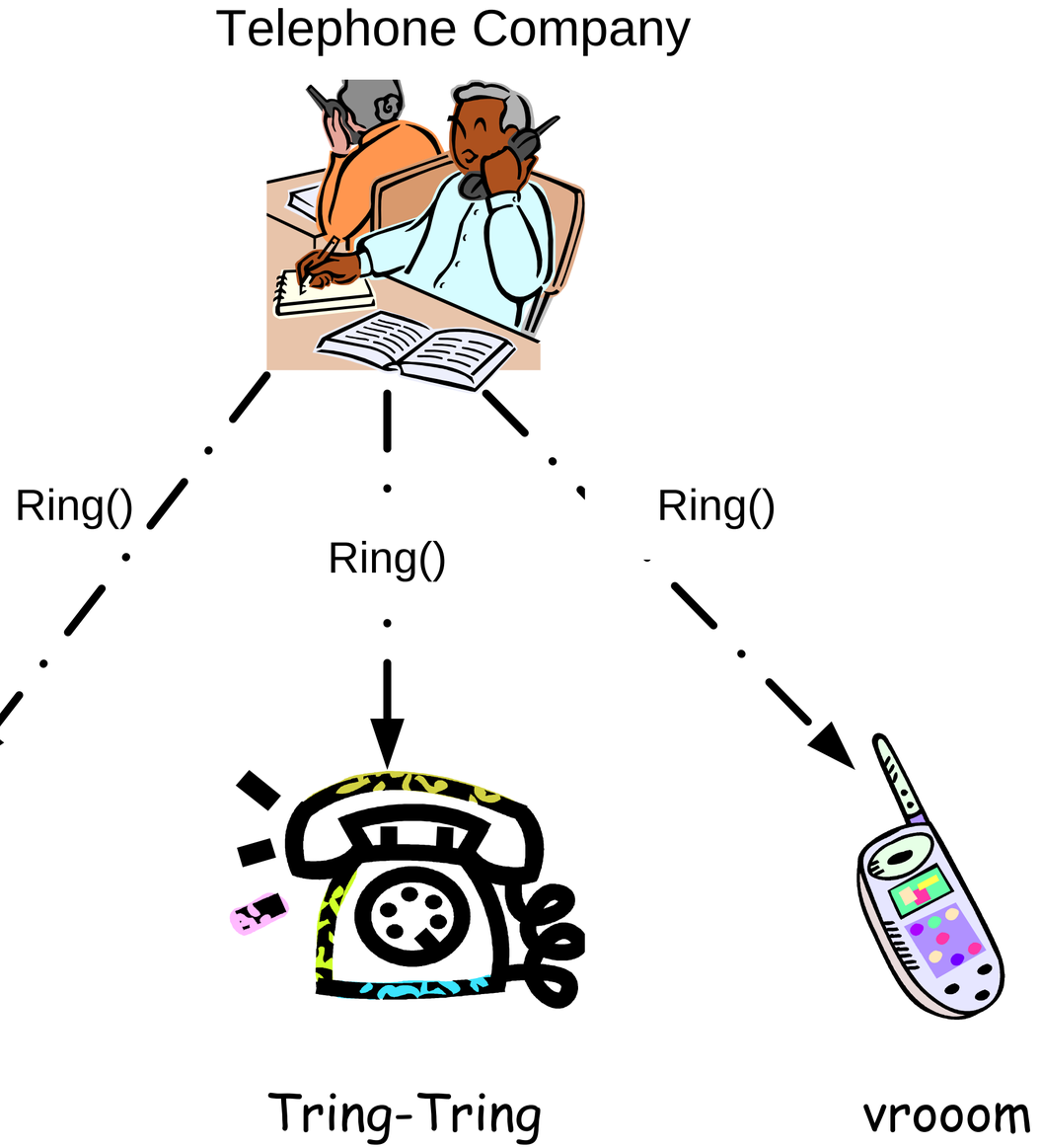
```
public class Test {
    public static void main(String arg[]){
        Person p = new Person();
        // ...
        Employee e = new Employee();
        // ...
        Manager m = new Manager();
        // ...
        Person pArr[] = {p, e, m};
        for (int i=0; i< pArr.length; i++){
            Console.WriteLine(pArr[i].getDetail());
        }
    }
}
```



Đa hình (Polumorphism) là gì?

- **Đa hình**: nhiều hình thức thực hiện một hành vi, nhiều kiểu tồn tại của một đối tượng
- Đa hình trong lập trình:
  - **Đa hình phương thức**: chồng phương thức, ghi đè phương thức
  - **Đa hình đối tượng**: nhìn nhận đối tượng theo nhiều kiểu khác nhau
- Ví dụ: Một bạn sinh viên là cán bộ lớp thì có thể nhìn nhận theo 2 góc nhìn

Đa hình là gì?

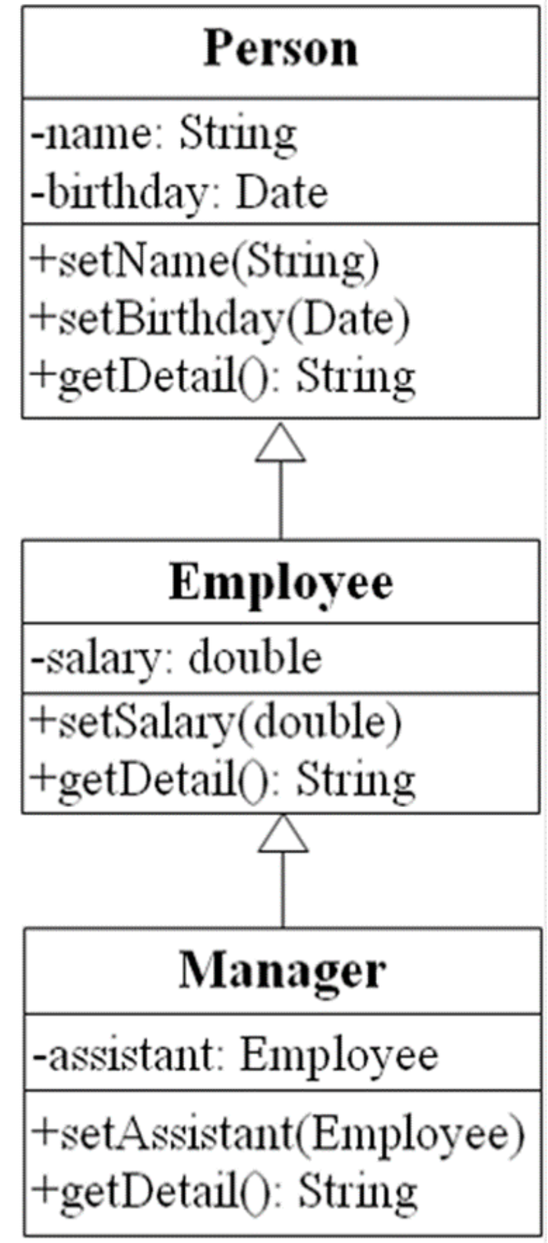


Đa hình (Polumorphism) là gì?

- Nhìn nhận đối tượng theo nhiều kiểu khác nhau → Upcasting và Downcasting

```
public static void main(string  args[])
{
   Person p1 = new Employee();
   Person p2 = new Manager();

   Employee e = (Employee) p1;
   Manager m = (Manager) p2;
}
```



Đa hình (Polumorphism) là gì?

- Các đối tượng khác nhau giải nghĩa các thông điệp theo các cách thức khác nhau → Liên kết động

```
Person p1 = new Person();
Person p2 = new Employee();
Person p3 = new Manager();
...
Console.WriteLine(p1.getDetail());
Console.WriteLine(p2.getDetail());
Console.WriteLine(p3.getDetail());
```

Một ví dụ khác

```
class EmployeeList {
    Employee list[];
    ...
    public void add(Employee e) {
        ...
    }
    public void print() {
        for (int i=0; i<list.length; i++) {
            Console.WriteLine(list[i].getDetail());
            ...
    }
...
EmployeeList list = new EmployeeList();
Employee e1;
Manager m1;
...
list.add(e1);
list.add(m1);
list.print();
```

Toán tử instanceof

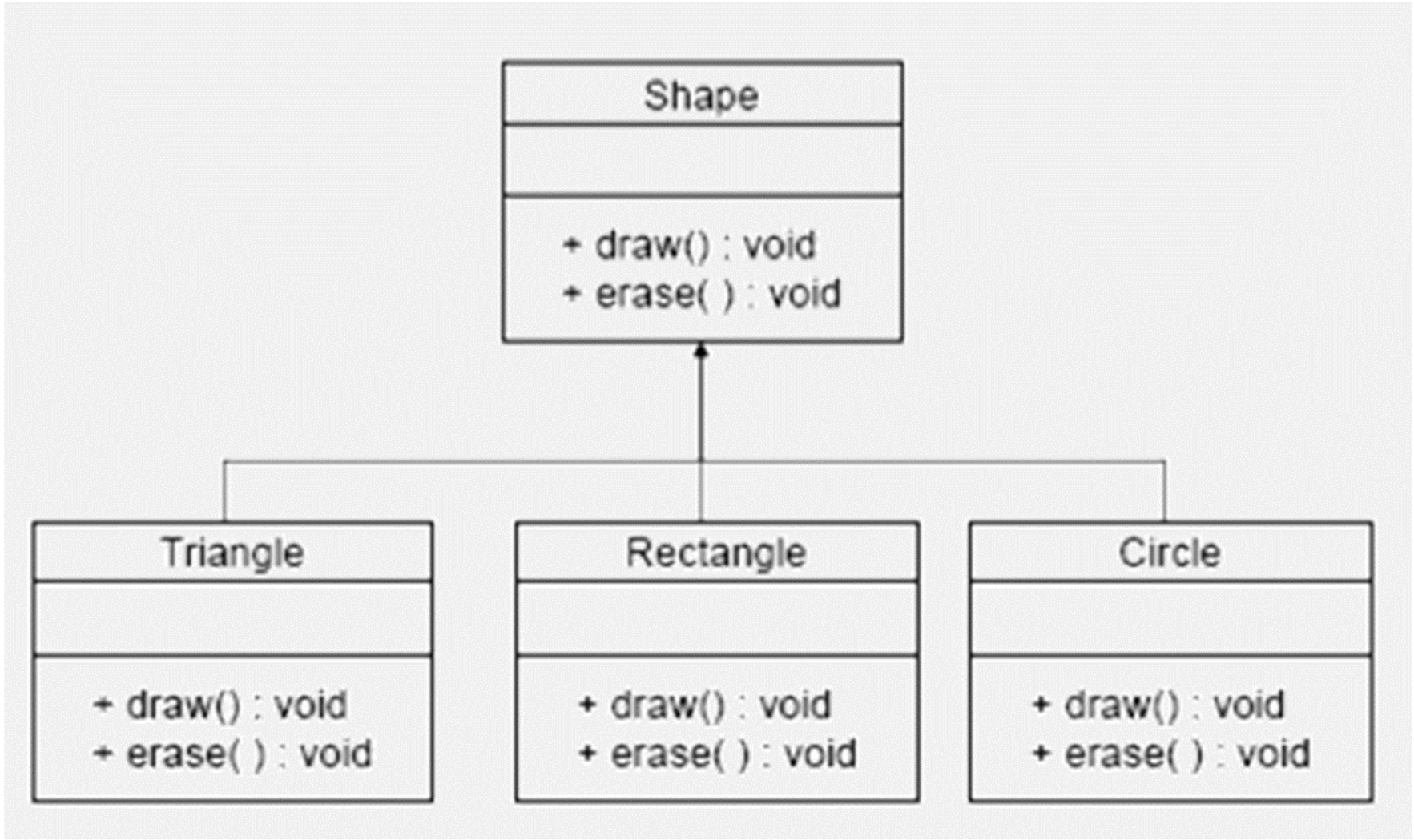
- Kiểm tra một đối tượng có phải đang là thể hiện của lớp hoặc giao diện nào đó không
- **Cú pháp**: objectName instanceof Class

  objectName instanceof Interface
- Kết quả:
  - true: đúng
  - false: sai

```
if (pObj  instanceof  Person)
    System.out.println(“pObj is a Person");
if (pObj  instanceof  Student)
    System.out.println(“pObj is a Student");
```

Ví dụ

- Các đối tượng Triangle, Rectangle, Circle đều là các đối tượng Shape



Ví dụ

```
...
public static void handleShapes(Shape[] shapes){
  // Vẽ các hình theo cách riêng của mỗi hình
  for( int i = 0; i < shapes.length; ++i) {
    shapes[i].draw();
  }
  ...
  // Gọi đến phương thức xóa, không cần quan tâm
  // đó là hình gì
  for( int i = 0; i < shapes.length; ++i) {
    shapes[i].erase();
  }
}
...
```

Ví dụ

Xây dựng chương trình quản lý việc tính lương cho nhân viên.

- Nhân viên có các thuộc tính là: mã số nhân viên, họ và tên, ngày sinh, số ngày làm việc.
- Lương của nhân viên được tính bằng công thức: số ngày làm việc nhân với 10 USD
- Nhân viên bán hàng còn có thêm thuộc tính số sản phẩm đã bán trong tháng.
- Lương nhân viên bán hàng được tính theo công thức: số ngày làm việc nhân với 5 USD + số sản phẩm nhân với 2 USD

Về lưu trữ: làm cách nào có thể lưu trữ tất cả các nhân viên thuộc các loại khác nhau vào trong một mảng

ĐẠI NAM
UNIVERSITY

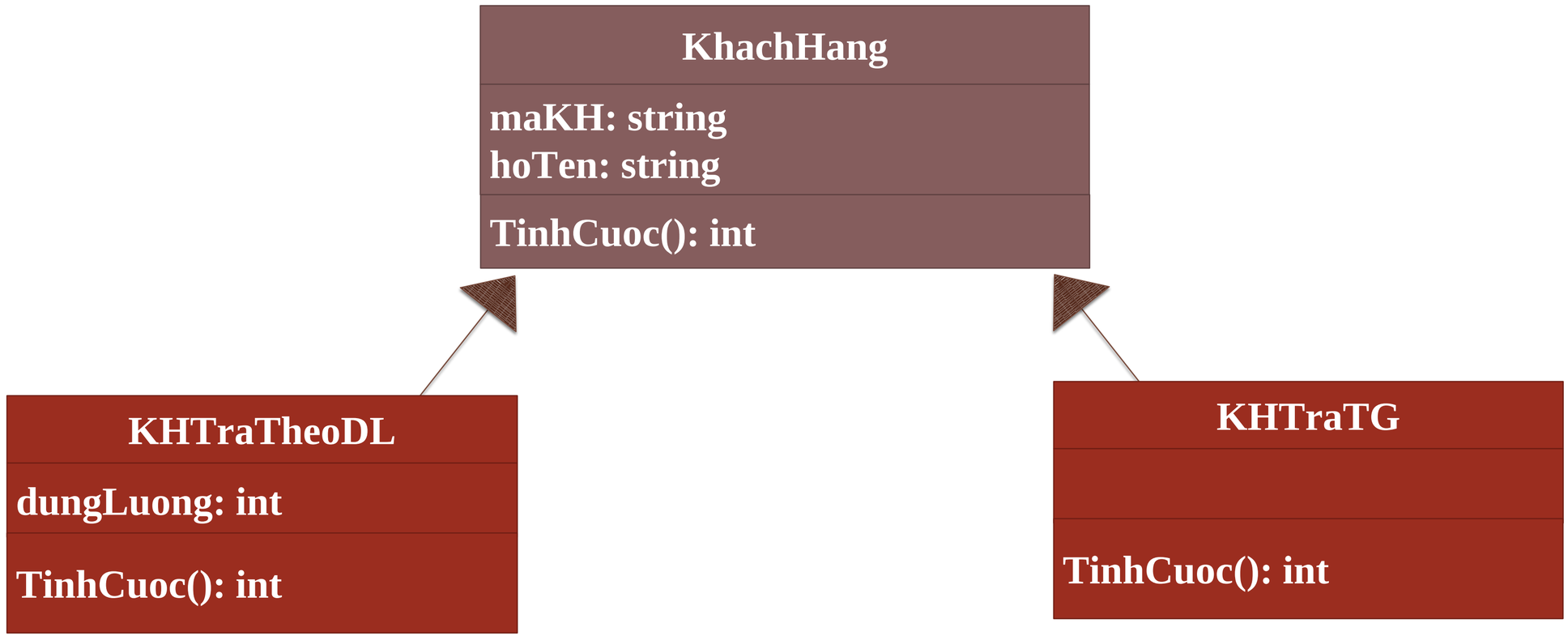
LỚP TRỪU TƯỢNG

Bài toán cần giải quyết

Tính cước phí của khách hàng sử dụng dịch vụ internet ADSL.

Có 2 loại khách hàng: khách hàng trả trọn gói và khách hàng trả theo dung lượng sử dụng.

Một khách hàng chỉ thuộc 1 trong 2 loại khách hàng trên.



Đoạn code minh hoạ

```
class KhachHang
{
    ……….
    public virtual int TinhCuoc()
    {
    }
}
```

Nên viết code như thế nào trong hàm TinhCuoc của lớp KhachHang

```
class KHTraTheoDL : KhachHang
{
    ……
    public int dungLuong;
    public override int TinhCuoc()
    {
        return dungLuong * 3;
    }
}
class KHTraTG : KhachHang
{
    ……..
    public override int TinhCuoc()
    {
        return 200000;
    }
}
```

Lớp trừu tượng

- Khi chưa thể định nghĩa rõ ràng nội dung của một phương thức → cần xây dựng phương thức đó như là **phương thức trừu tượng (abstract method).**
- Lớp chứa phương thức trừu tượng bắt buộc phải khai báo như lớp trừu tượng.

```
public abstract class  Person {
  public abstract void displayPerson();
}
```

Lớp trừu tượng

- Khi chưa thể định nghĩa rõ ràng nội dung của một phương thức → cần xây dựng phương thức đó như là phương thức trừu tượng
- Lớp chứa phương thức trừu tượng bắt buộc phải khai báo như lớp trừu tượng

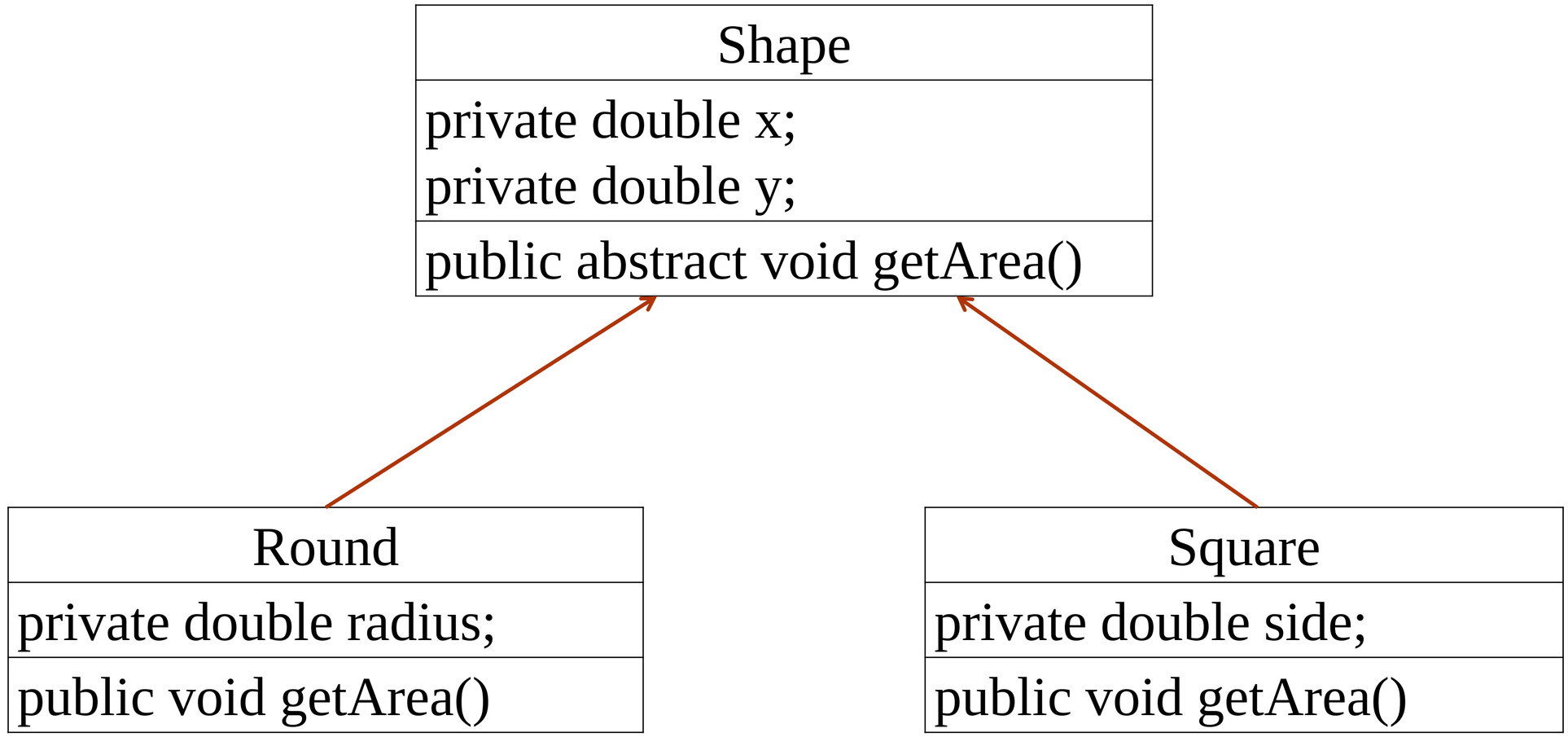
```
package  java.oop.person;
/**The Person class contains some information of someone
*/
public abstract class  Person {
  public abstract void displayPerson();
}
```

Các quy tắc khi sử dụng lớp trừu tượng

- Phương thức trừu tượng không được phép định nghĩa cụ thể tại lớp cha
- Lớp con kế thừa từ lớp trừu tượng phải định nghĩa nội dung của phương thức trừu tượng
  - Chỉ định truy cập không được chặt hơn lớp cha
  - Nhắc lại mức độ chặt của các chỉ định truy cập:

    public > protected > không chỉ định > private
- Không được tạo đối tượng từ lớp trừu tượng
  - Nhưng lớp trừu tượng vẫn có phương thức khởi tạo

Ví dụ về lớp trừu tượng



Lớp Shape

```
package java.oop.shape;
/** The Shape class illutrating a shape has x and y
coordinate and an abstract method */
public  abstract  class  Shape {
    private double x;
    private double y;
    /**
    * Constructs a new shape
    */
    public Shape(double initX, double initY){
        this.x = initX;
        this.y = initY;
    }
    public abstract void getArea();
}
```

Lớp Round

```
package java.oop.shape;
/** The Round class presents a round */
public class Round extends Shape {
    private double radius;
    /**
    * Constructs a new round
    */
    public Round(double initX, double initY, double  initRadius){
        super(initX,initY);
        this.radius = initRadius;
    }
    public void getArea(){
        return Math.PI*radius*radius;
    }
}
```

Lớp Square

```
package java.oop.shape;
/** The Square class presents a square */
public class Square extends Shape{
    private double side;
    /**
    * Constructs a new square
    */
    public Square(double initX, double initY, double  initSide){
        super(initX,initY);
        this.side = initSide;
    }
    public void getArea(){
        return side*side;
    }
}
```

Lớp ShapeTest

```
package java.oop.shape;
/** The Square class presents a square */
public  class  ShapeTest{
   public static void main(String arg[]){
      Shape shapeObj = new Shape(1,1); //wrong

      Round roundObj = new Round(1,1,2); //OK
      System.out.println("The area of this round" +  roundObj.getArea());

      Square squareObj = new Square(0,1,1); //OK
      System.out.println("The area of this square" +  squareObj.getArea());
   }
}
```

ĐẠI NAM
UNIVERSITY

INTERFACE

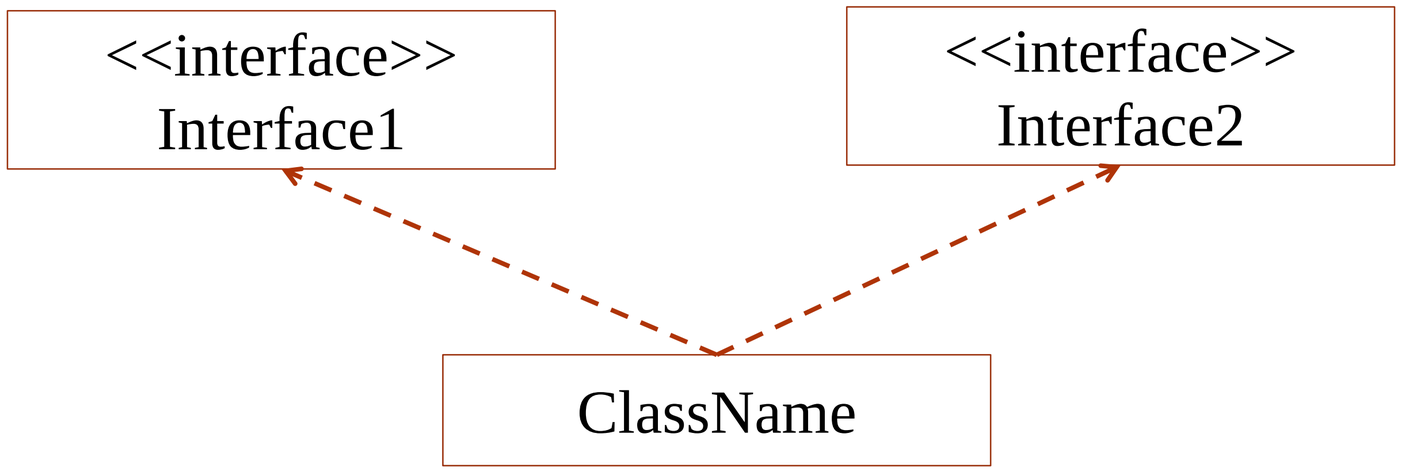
http://dainam.edu.vn

Giao diện (Interface)

- Java không cho phép đa kế thừa từ nhiều lớp
- Để thực hiện đa kế thừa, Java sử dụng khái niệm giao diện (interface)
- Giao diện chỉ quy định các phương thức phải có, nhưng không định nghĩa cụ thể
  - Cho phép tách rời đặc tả mức trừu tượng và triển khai cụ thể
  - Đảm bảo tính cộng tác trong phát triển phân mềm
- Các giao diện có thể kế thừa nhau
- Cú pháp

```
Modifier  interface  InterfaceName {
  //Declare constants
  //Declare methods
}
```

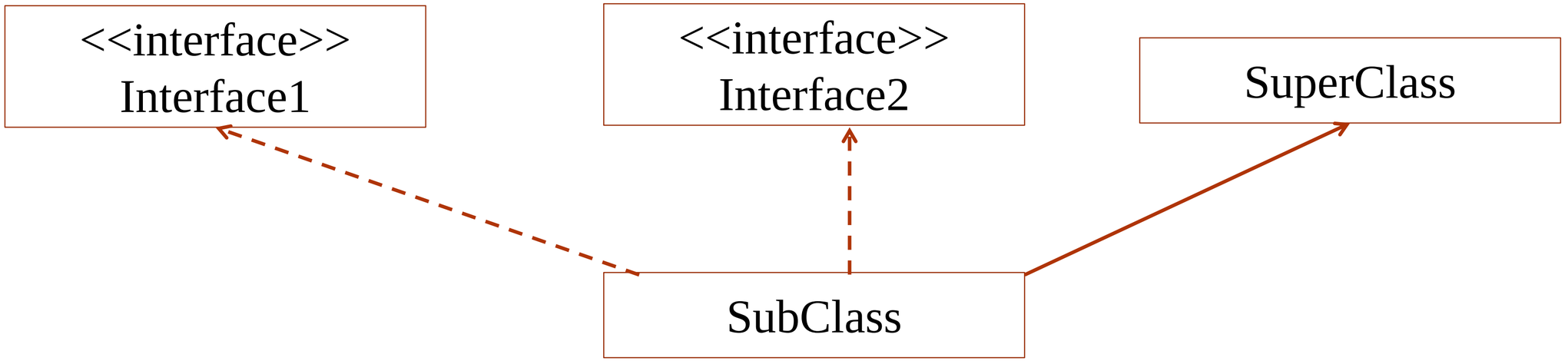
Triển khai giao diện




```
Modifier class ClassName  implements Interface1, Interface2
{
   //class's body
}
```

- ClassName phải định nghĩa mọi phương thức của Interface1và Interface2

Kế thừa và triển khai giao diện đồng thời



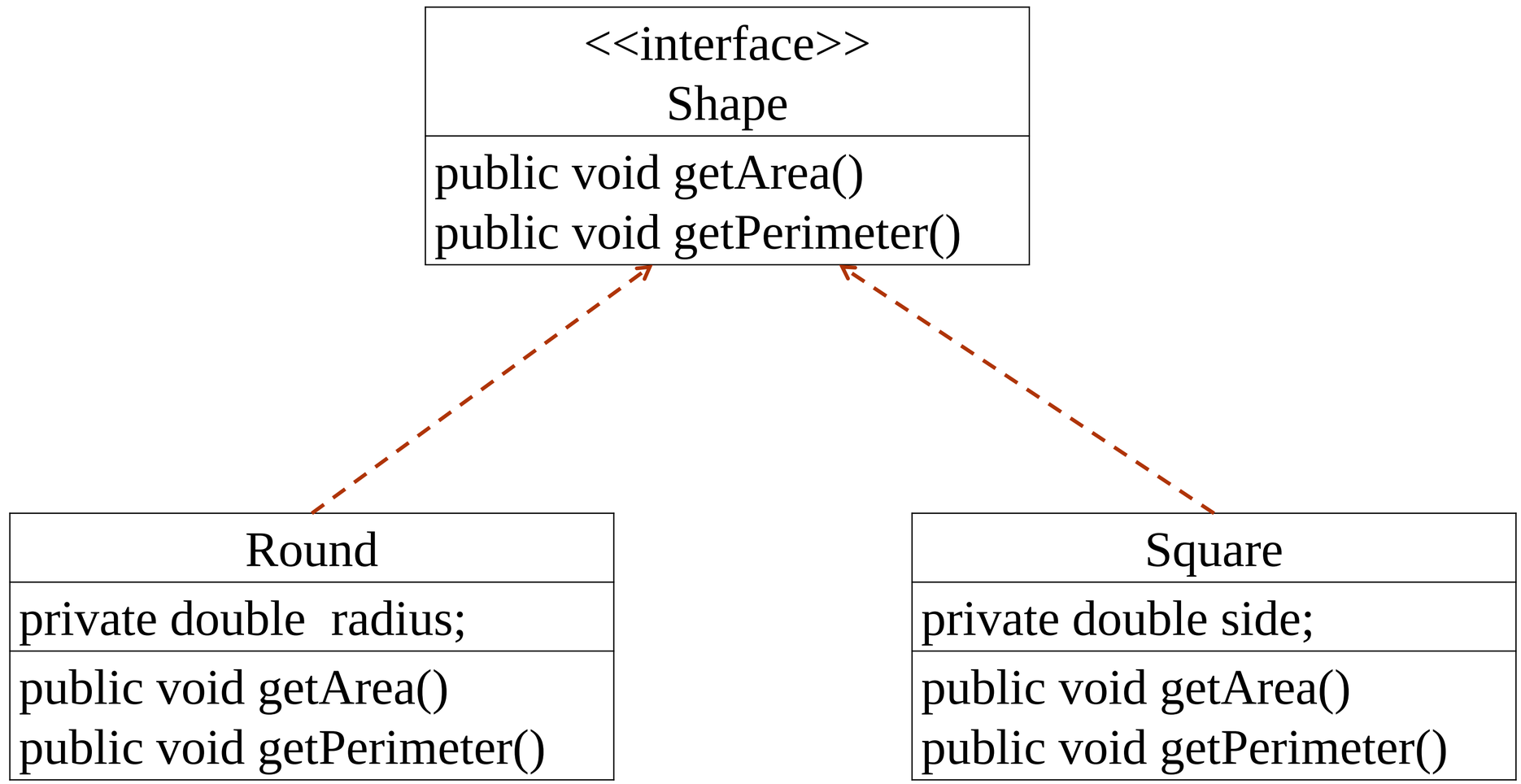

```
Modifier class SubClass extends SuperClass  implements
                              Interface1, Interface2{
   //class's body
}
```

- SubClass kế thừa các phương thức, thuộc tính của SuperClass và phải định nghĩa mọi phương thức của Interface1và Interface2

So sánh Giao diện với Lớp trừu tượng

Giao diện	Lớp trừu tượng
• Chỉ được phép có thành viên hằng • Mọi phương thức là trừu tượng với chỉ định truy cập public • Không có phương thức khởi tạo • Một lớp có thể triển khai nhiều giao diện • Không tái sử dụng mã nguồn	• Có thể có thuộc tính • Ngoài phương thức trừu tượng, có thể có phương thức riêng • Có phương thức khởi tạo • Một lớp chỉ có thể kế thừa từ một lớp trừu tượng • Có tái sử dụng mã nguồn

Ví dụ về interface



Giao diện Shape

```
package  java.oop.shape;
/** The Shape interfce illutrating a shape */
public  interface  Shape {
   public void  getArea();
   public void  getPerimeter();
}
```

Lớp Round

```
package java.oop.shape;
/** The Round class presents a round */
public class Round  implements Shape {
    private double radius;
    /** Constructs a new round */
    public Round(double initRadius){
        this.radius = initRadius;
    }
    public void getArea(){
        return Math.PI * radius * radius;
    }
    public void getPerimeter(){
        return 2 * Math.PI * radius;
    }
    public double getRadius(){return this.radius;}
}
```