

Chương 4

KẾ THỪA - ĐA HÌNH

TRÊN JAVA

Mục tiêu



- Vẽ sơ đồ UML và hiện thực tính kế thừa trong Java
- Viết phương thức *override*
- Biết cách tạo và sử dụng lớp trừu tượng (*abstract class*)
- Hiện thực *Interface*
- Cài đặt được tính đa hình trong Java

Nội dung



- 4.1. Khái niệm kế thừa
- 4.2. Kỹ thuật phân cấp kế thừa
- 4.3. Hiện thực tính kế thừa trong Java
- 4.4. Lớp trừu tượng (Abstract class)
- 4.5. Interface
- 4.6. Đa hình (Polymorphism)

4.1. Khái niệm kế thừa

Vấn đề



- Ví dụ xét trường hợp bài toán quản lí nhân sự và sinh viên của một trường đại học

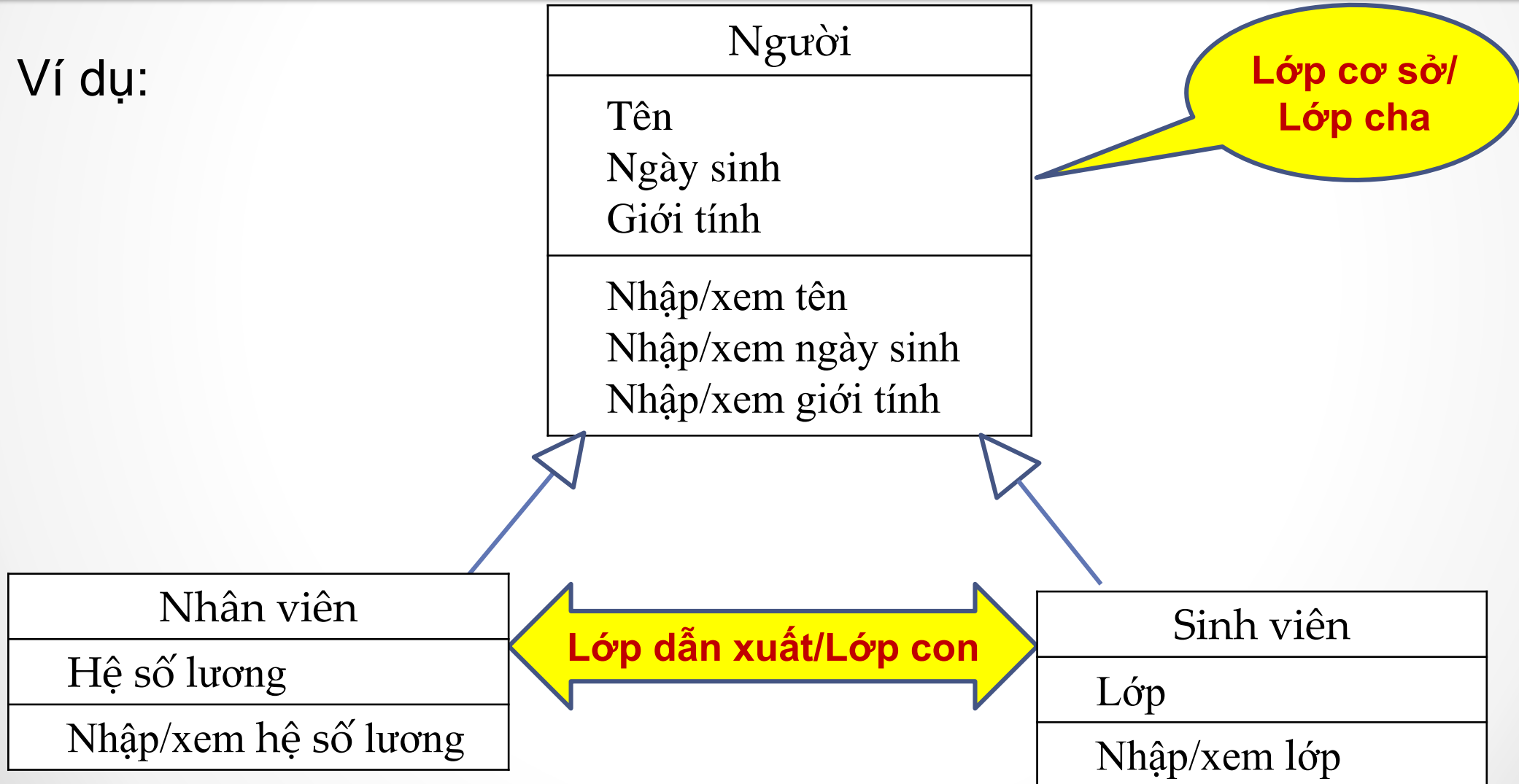
| Nhân viên |
|----------------------|
| Tên |
| Ngày sinh |
| Giới tính |
| Hệ số lương |
| Nhập/xem tên |
| Nhập/xem ngày sinh |
| Nhập/xem giới tính |
| Nhập/xem hệ số lương |

| Sinh viên |
|--------------------|
| Tên |
| Ngày sinh |
| Giới tính |
| Lớp |
| Nhập/xem tên |
| Nhập/xem ngày sinh |
| Nhập/xem giới tính |
| Nhập/xem lớp |

4.1. Khái niệm kế thừa

Giải quyết vấn đề

- Ví dụ:



4.1. Khái niệm kế thừa

Vấn đề



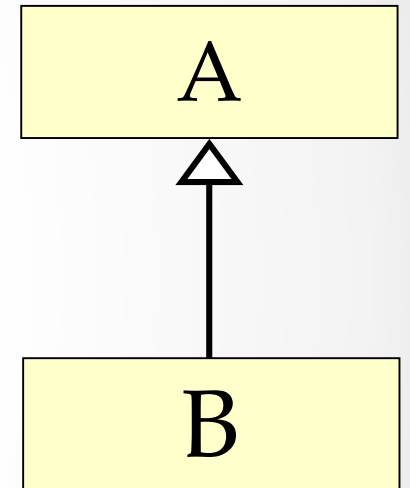
- Một số phân loại tồn tại trong thực tế

| Đối tượng | Phân loại |
|-------------|--|
| Student | GraduateStudent, UndergraduateStudent |
| Shape | Circle, Triangle, Rectangle |
| Loan | CarLoan, HomeImprovementLoan, MortgageLoan |
| Employee | Faculty, Staff |
| BankAccount | CheckingAccount, SavingsAccount |
| ... | |

4.1. Khái niệm kế thừa



- Kế thừa là việc xây dựng lớp mới dựa trên lớp đã có sẵn
 - Lớp có sẵn gọi là *parent class/super class/base class*
 - Lớp mới gọi là *child class/subclass/derived class*
- Là mối quan hệ “is-a”
- Lớp con **kế thừa** những thuộc tính và hành vi của lớp cha
- Lớp con có thể **thêm** những thuộc tính và hành vi của riêng mình và có thể **sửa đổi** những hành vi của lớp cha
- Kế thừa cho phép tái sử dụng mã → Tiết kiệm công sức xây dựng + kiểm thử
- Trong Java, mặc định mọi lớp kế thừa lớp Object



B kế thừa A

4.1. Khái niệm kế thừa

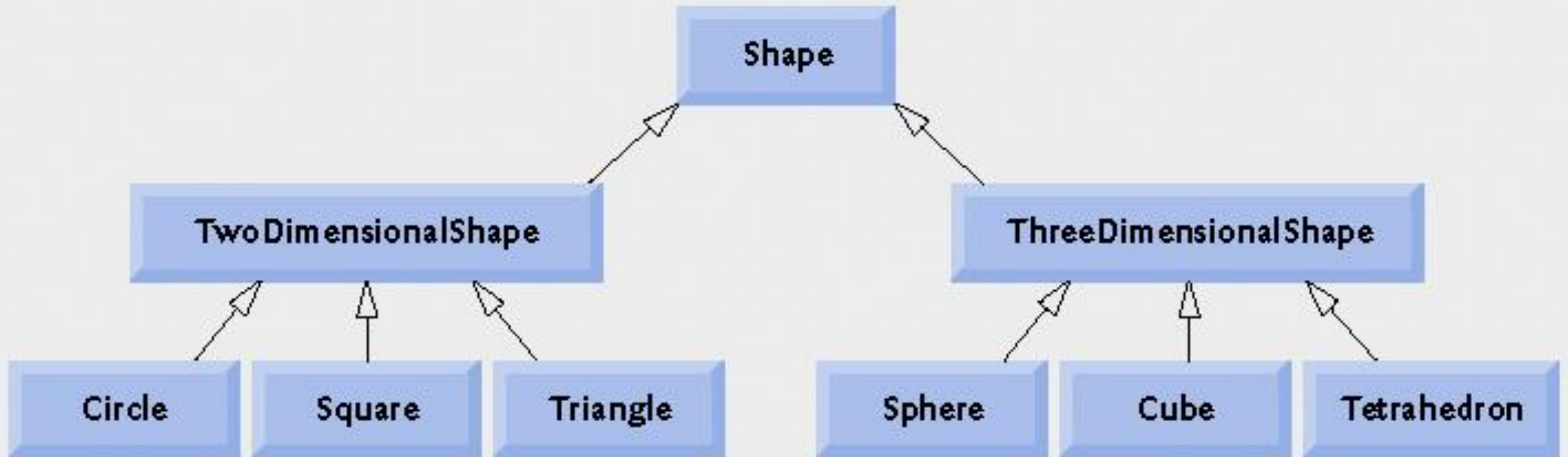
Ví dụ 1



- Sắp xếp các class sau theo một phân cấp phù hợp và vẽ lược đồ lớp:
 - Nhà phố
 - Ngôi nhà
 - Biệt thự
 - Xe

4.1. Khái niệm kế thừa

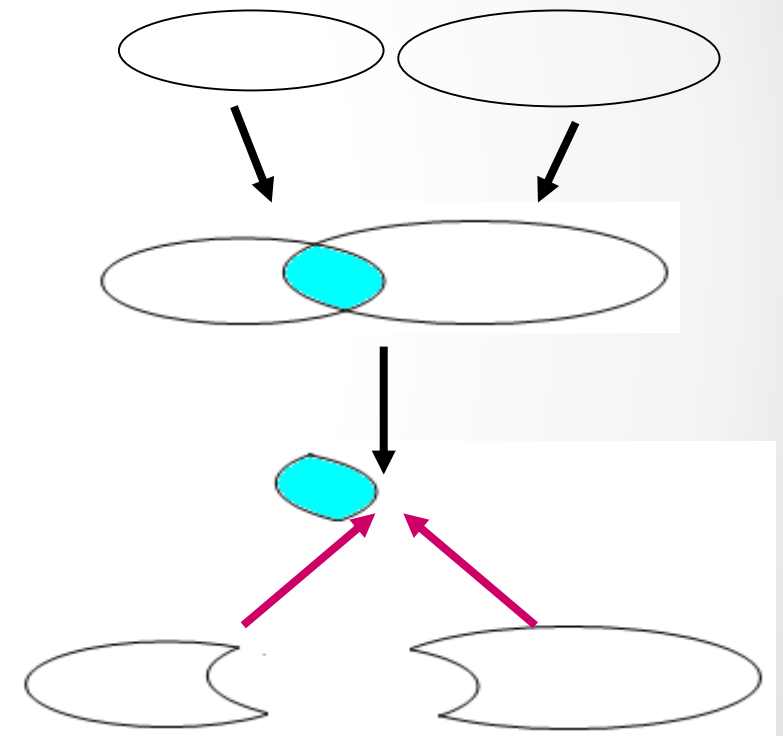
Ví dụ 2



4.2. Kỹ thuật phân cấp kế thừa



1. Liệt kê đặc điểm của các loại đối tượng cần quan tâm.
2. Tìm tập giao của các tính chất giữa các lớp, tách tập giao này để xây dựng lớp cha.
3. Đặt tên gọi có ý nghĩa cho lớp cha.
4. Phần còn lại sau khi tách tập giao là các lớp con.



4.2. Kỹ thuật phân cấp kế thừa

Ví dụ 1



- Cho lớp Book và Magazine được định nghĩa như sau, xác định sự tương đồng và khác nhau giữa chúng. Lập mô hình kế thừa.

| Book |
|------------------------------------|
| title author price copies |
| sellCopy() orderCopies() |

| Magazine |
|---|
| title price orderQty currIssue copies |
| sellCopy() adjustQty() recvNewIssue() |

4.2. Kỹ thuật phân cấp kế thừa

Ví dụ 2



Công ty du lịch X có quản lý thông tin chuyến xe. Có 2 loại chuyến xe:

- Chuyến xe nội thành: gồm *Mã số chuyến, Họ tên tài xế, số xe, số tuyến, số km đi được, doanh thu*.
- Chuyến xe ngoại thành: gồm *Mã số chuyến, Họ tên tài xế, số xe, nơi đến, số ngày đi được, doanh thu*.

Yêu cầu: Xây dựng các lớp với chức năng thừa kế (vẽ mô hình).

4.2. Kỹ thuật phân cấp kế thừa

Ví dụ 2: Giải



4.3. Hiện thực tính kế thừa trong Java

Cú pháp



- Cú pháp kế thừa trong Java:

```
public class Child_ClassName extends Parent_ClassName {  
    // derived class methods extend and possibly override  
    // those of the base class  
}
```

4.3. Hiện thực tính kế thừa trong Java

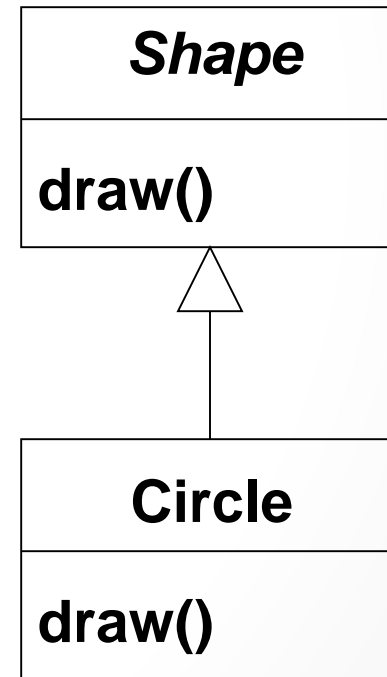
Ví dụ 1



- Cài đặt quan hệ kế thừa:

```
public class Shape
{
    public void draw() { ... }
    ...
}

public class Circle extends Shape
{
    public void draw() { ... }
    ...
}
```



4.3. Hiện thực tính kế thừa trong Java

Ví dụ 2



```
public class ChuyenXe {
    private String maSoChuyen, hoTenTaiXe, soXe;
    private double doanhThu;
    public ChuyenXe() { /* */ }
    public ChuyenXe(String maSoChuyen, String hoTenTaiXe,
                     String soXe, double doanhThu) { /* */ }
    public String toString() {
        return "MS chuyển:" + maSoChuyen + ", Tài xế: " + hoTenTaiXe +
            ", Số xe: " + soXe + ", Doanh thu:" + doanhThu;
    }
}
```


4.3. Hiện thực tính kế thừa trong Java

Ví dụ 2 (tt)



```
public class ChuyenXeNoiThanh extends ChuyenXe {
    private String soTuyen;
    private double soKmDiDuoc;
    public ChuyenXeNoiThanh() {
        super();
    }
    public ChuyenXeNoiThanh(String maSoChuyen, String hoTenTaiXe, String soXe,
                             double doanhThu, String soTuyen, double soKmDiDuoc) {
        super(maSoChuyen, hoTenTaiXe, soXe, doanhThu);
        this.soTuyen = soTuyen;
        this.soKmDiDuoc = soKmDiDuoc;
    }
    public String toString() {
        return "CX NoiThanh:" + super.toString() + ", số tuyến: " + this.soTuyen +
            ", số km đã đi: " + this.soKmDiDuoc;
    }
}
```

4.3. Hiện thực tính kế thừa trong Java

Từ khóa *super*



- Từ khóa **super** được dùng trong các lớp con, để:
 - truy xuất các thành phần của lớp cha (xem lại ví dụ 2)
 - gọi hàm khởi tạo của lớp cha (xem lại ví dụ 2)
- Sự thừa kế trong hàm khởi tạo - *Constructor inheritance*
 - Gọi tường minh hàm khởi tạo của lớp cha
 - Gọi ngầm định hàm khởi tạo của lớp cha


4.3. Hiện thực tính kế thừa trong Java

Gọi tường minh hàm khởi tạo



```
class Circle {  
    private double radius;  
    public Circle( double radius ) { //Circle's constructor  
        this.radius = radius;  
        System.out.println("This is constructor of class Circle");  
    }  
}
```

Cylinder b = new Cylinder(10, 8.6);



```
class Cylinder extends Circle {  
    private double height;  
    public Cylinder( double radius, double height ) { //Cylinder's constructor  
        super(radius); //gọi tường minh hàm khởi tạo của lớp Circle  
        this.height = height;  
        System.out.println("This is constructor of class Cylinder");  
    }  
}
```

OUTPUT

This is constructor of class Circle
This is constructor of class Cylinder

4.3. Hiện thực tính kế thừa trong Java

Gọi ngầm định hàm khởi tạo



- Ví dụ:

```
public class Parent
{
    public Parent()
    {
        System.out.println("Invoke parent default constructor");
    }
}

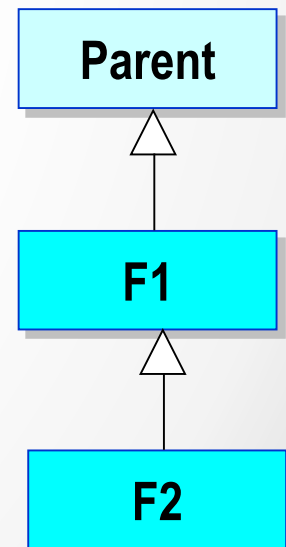
public class F1 extends Parent
{
    public F1()
    {
        System.out.println("Invoke F1 default constructor");
    }
}

public class F2 extends F1
{
    public F2()
    {
        System.out.println("Invoke F2 default constructor");
    }
}
```

```
public static void main(String [] args)
{
    new F2 ();
}
```

```
Invoke Parent default constructor
Invoke F1 default constructor
Invoke F2 default constructor
```

- | |
|---------------------------------|
| 1. Object |
| 2. Parent() call super() |
| 3. F1() call super() |
| 4. F2() call super() |
| 5. Main() call new F2() |



4.3. Hiện thực kế thừa trong Java

Phương thức *Override*

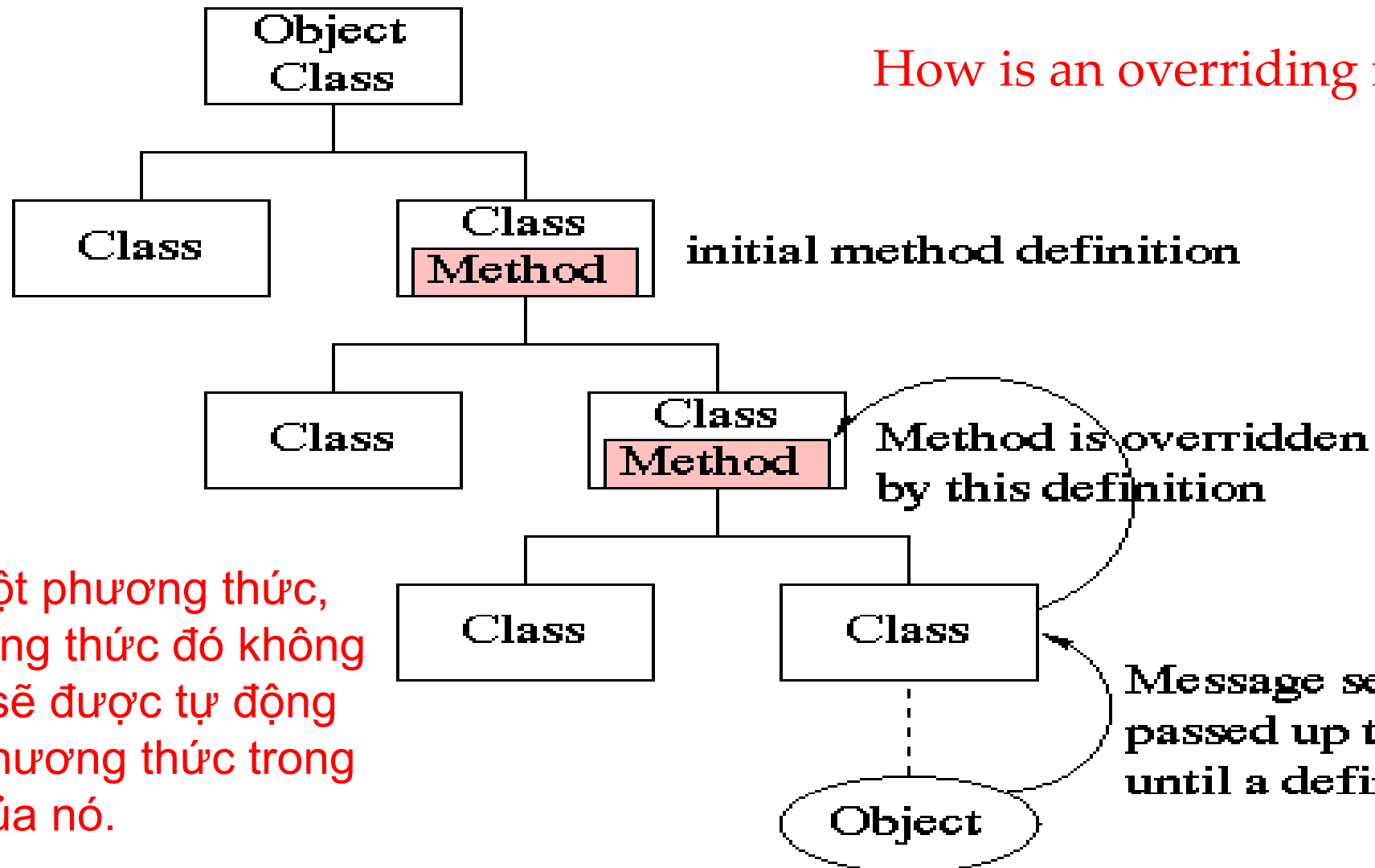


- Lớp con kế thừa tất cả các phương thức từ lớp cha của chúng. Tuy nhiên, nội dung thực thi cụ thể của phương thức có thể **được thay đổi ở lớp con**. Đó là khả năng *ghi đè* phương thức (*override*).
- Để ghi đè một phương thức, trong lớp con, viết phương thức có *signature* giống hệt như phương thức mà nó ghi đè (và *@Override*)
- Ví dụ: Override phương thức `toString()` của lớp `Object`:

```
public class Student {  
    /* ... */  
    @Override  
    public String toString() {  
    }  
}
```

4.3. Hiện thực tính kế thừa trong Java

Phương thức *Override* (cont.)



Khi gọi một phương thức, nếu phương thức đó không có thì nó sẽ được tự động gọi đến phương thức trong lớp cha của nó.

How is an overriding method called?

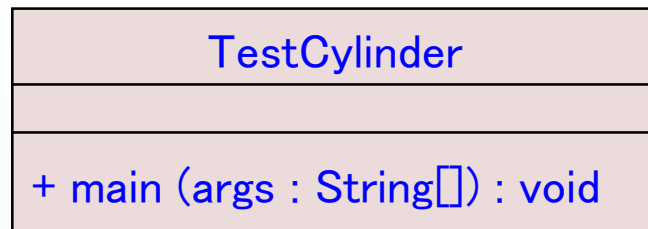
4.3. Hiện thực tính kế thừa trong Java

Case study [1/4]



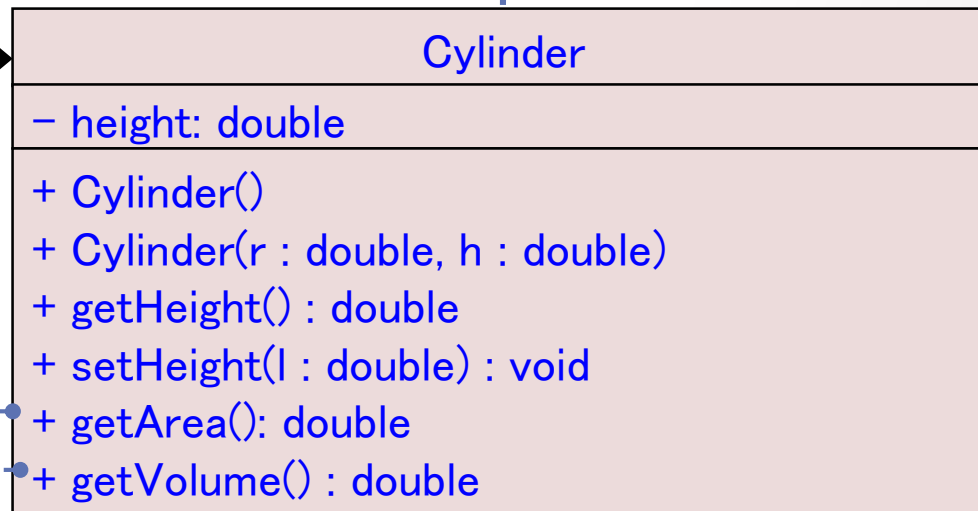
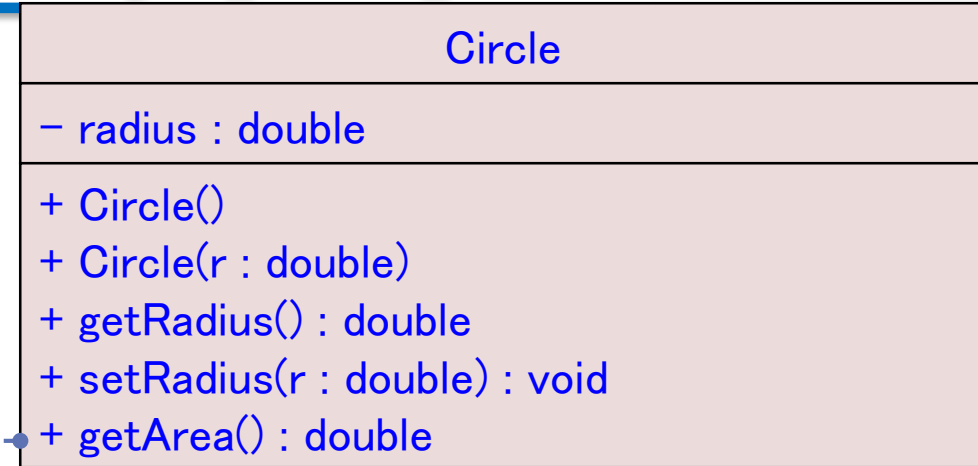
- Cài đặt cho mô hình lớp sau:

Diện tích hình tròn = $3.14 \times \text{bán kính}^2$



Diện tích xq hình trụ = chu vi đáy x chiều cao

Thể tích hình trụ = diện tích đáy x chiều cao



4.3. Hiện thực tính kế thừa trong Java

Case study [2/4]



```
public class Circle {  
    private double radius;  
    public Circle() {  
        radius = 0;  
    }  
    public Circle(double r) {  
        radius = r;  
    }  
    public double getRadius() {  
        return radius;  
    }  
    public void setRadius(double r){  
        radius = r;  
    }  
}
```

```
    public double getArea() {  
        return radius * radius * Math.PI;  
    }  
}
```


4.3. Hiện thực tính kế thừa trong Java

Case study [3/4]



```
public class Cylinder extends Circle {  
    private double height;  
    public Cylinder() {  
        super();  
        height = 0;  
    }  
    public Cylinder(double r, double h) {  
        super(r);  
        height = h;  
    }  
    public double getHeight() {  
        return height;  
    }  
    public void setHeight(double h) {  
        height = h;  
    }  
}
```

```
@Override  
public double getArea() {  
    return 2 * Math.PI * super.getRadius()  
        * height;  
}  
  
public double getVolume() {  
    return super.getArea() * height; //?  
}  
}
```

4.3. Hiện thực tính kế thừa trong Java

Case study [4/4]



```
public class Test {  
    public static void main(String[] args) {  
        Cylinder c = new Cylinder(5.0, 5.2);  
        System.out.println("The radius is " + c.getRadius()); // ?  
        System.out.println("The height is " + c.getHeight());  
        System.out.println("The area of the cylinder is " + c.getArea()); // ?  
        System.out.println("The volume of the cylinder is " + c.getVolume());  
    }  
}
```



Review questions 1

1. Lớp nào là lớp trên cùng của phân cấp lớp trong Java.
2. Constructor có được kế thừa không. Làm thế nào để sử dụng lại constructor của lớp cha.
3. Method overriding là gì. Khi nào thì cần sử dụng.

Bài tập



- Module 3

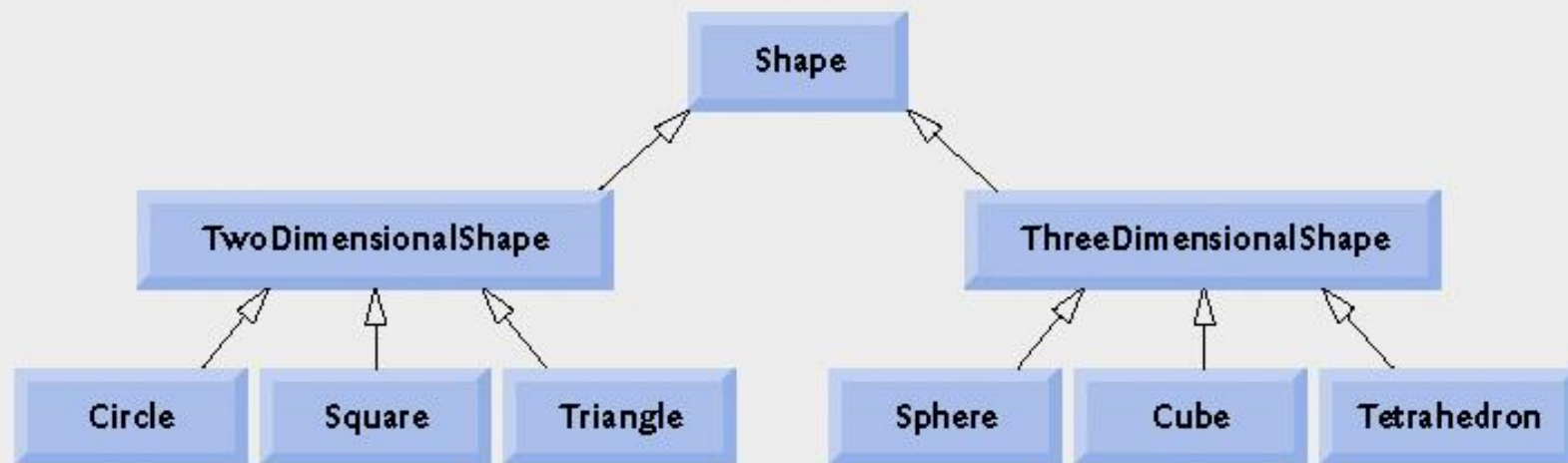
4.3. Lớp trừu tượng Vấn đề 1



Người

Tên
Ngày sinh
Giới tính

Nhập/xem tên
Nhập/xem ngày sinh
Nhập/xem giới tính



Làm sao để ngăn cản việc tạo ra đối tượng Người, đối tượng Shape (vì không mang ý nghĩa)

Nhân viên

Lương

Nhập/xem lương

Sinh viên

Lớp

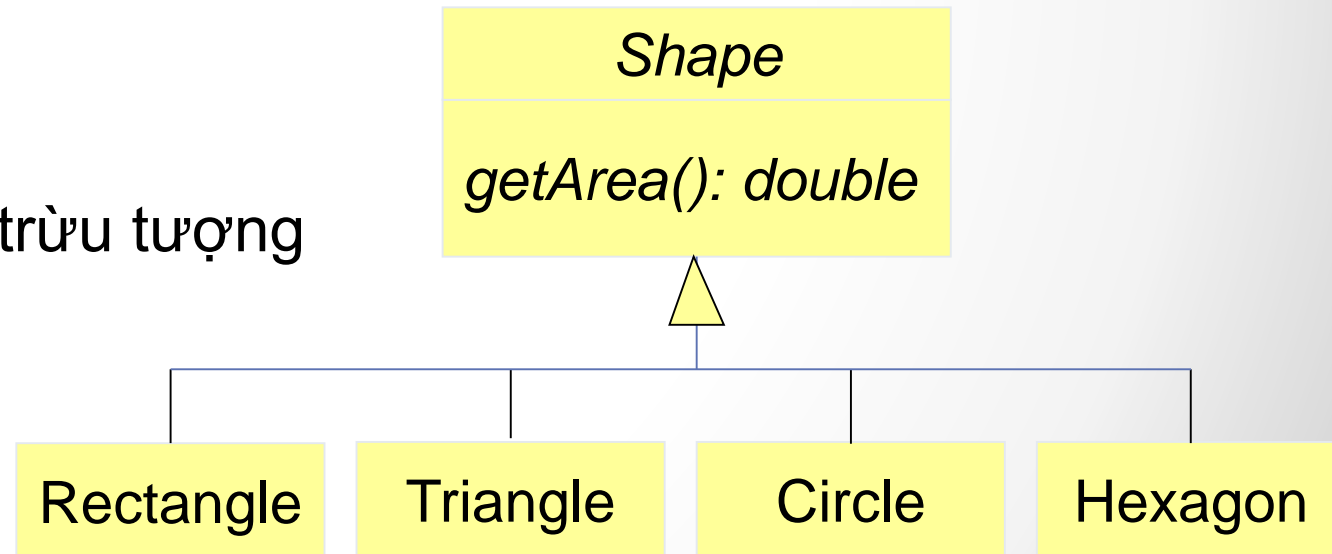
Nhập/xem lớp

4.3. Lớp trừu tượng

Vấn đề 2



- Giả sử có ứng dụng thao tác với các loại hình bằng phương pháp HĐT (xem hình), trong đó có công việc tính diện tích cho từng loại hình.
- Vấn đề:
 - Làm sao viết hàm *getArea()* cho lớp Shape
- Giải quyết:
 - Khai báo hàm *getArea()* là hàm trừu tượng
→ Shape là *lớp trừu tượng*



4.3. Lớp trừu tượng

Khái niệm – Đặc trưng



- *Lớp trừu tượng (abstract class)* là một lớp chung, không có thật, là cơ sở để tạo ra các lớp con khác có cùng tính chất, như là “giao kèo” mà bắt buộc tất cả các lớp con phải theo cùng tiêu chuẩn như nhau
- Đặc điểm của lớp trừu tượng:
 - Lớp trừu tượng được khai báo là **abstract**,
 - Có thể CÓ hoặc KHÔNG các *phương thức trừu tượng* (các lớp con phải hiện thực các phương thức trừu tượng này),
 - Không thể khởi tạo một đối tượng trực tiếp từ một lớp trừu tượng.

4.3. Lớp trừu tượng

Ví dụ



```
abstract class Shape
{
    protected int x, y;
    public Shape(int _x, int _y)
    {
        x = _x;
        y = _y;
    }
}
```

```
class Circle extends Shape
{
    int r;
    public Circle(int _x, int _y, int _r)
    {
        super(_x, _y);
        r = _r;
    }
}
```

Lớp trừu tượng được khai báo với từ khóa abstract

```
class ShapTest
{
    public static void main(String[] args)
    {
        Shape s1 = new Circle(5, 5, 6);
        Shape s = new Shape(10, 10); // compile error
    }
}
```

Lớp trừu tượng không thể được khởi tạo

4.3. Lớp trừu tượng

Phương thức trừu tượng



- Là phương thức chỉ có khai báo mà không có phần hiện thực:
 - Có từ khóa *abstract* trong phần khai báo
 - Phần khai báo được kết thúc bởi dấu ;

- Ví dụ:

```
public abstract class Shape
{
    public abstract double getArea();
}
```

- Phương thức trừu tượng bắt buộc phải được **ĐỊNH NGHĨA LẠI** tại lớp con

4.3. Lớp trừu tượng

Phương thức trừu tượng: Ví dụ

★

```
public abstract class Shape {  
    final static int BLACK = 0;  
    private int colour;  
  
    public Shape() {  
        colour = BLACK;  
    }  
  
    public void setColour(int c) {  
        this.colour = c;  
    }  
  
    public abstract double getArea();  
}
```

Abstract methods
have no body

Phương thức trừu tượng bắt buộc phải được
định nghĩa lại tại lớp con

Shape

getArea(): double
setColour(int)

Circle

```
public class Circle extends Shape {  
    final static double PI = 3.1419;  
    private int radius;  
  
    public Circle(int r) {  
        radius = r;  
    }  
  
    public double getArea() {  
        return (radius^2)*PI;  
    }  
}
```

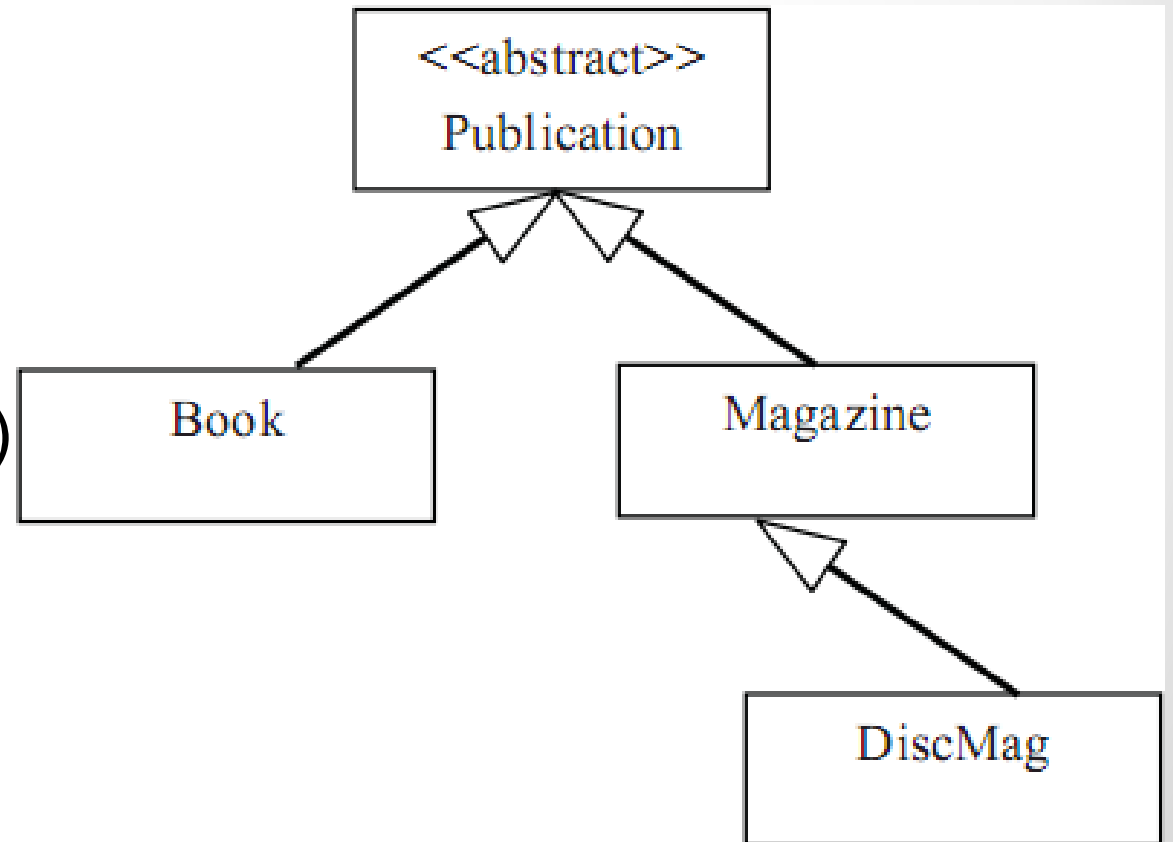
4.3. Lớp trừu tượng

Quiz 1



Cho mô hình lớp sau, câu lệnh nào sau đây là hợp lệ:

- a) `Publication p = new Book(...);`
- b) `Publication p = new DiscMag(...);`
- c) `Magazine m = new DiscMag(...);`
- d) `DiscMag dm = new Magazine(...);`
- e) `Publication p = new Publication(...);`



4.3. Lớp trừu tượng

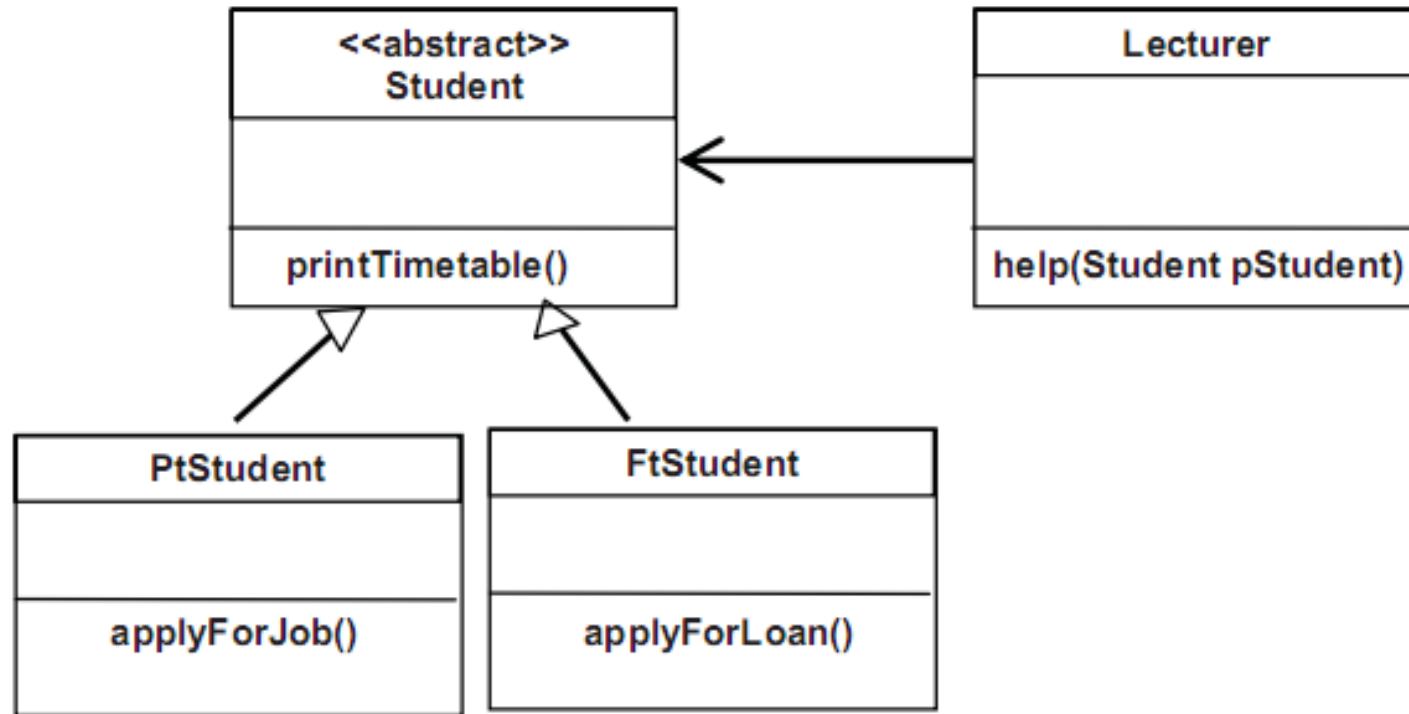
Quiz 2



Cho mô hình lớp sau, cho biết đoạn lệnh nào sau đây là hợp lệ:

a. `Student s = new Student();`
`Lecturer l = new Lecturer();`
`l.help(s);`

b. `Student s = new FtStudent();`
`Lecturer l = new Lecturer();`
`l.help(s);`



4.3. Lớp trừu tượng Employee

★ Case study

- Thiết kế và cài đặt mô hình lớp với mô tả sau:

Salaried-
Employee

Hourly-
Employee

Commission-
Employee

BasePlus-
Commission-
Employee

abstract

firstName lastName
social security number: *SSN*

weeklySalary

salaried employee: *firstName lastName*
social security number: *SSN*
weekly salary: *weeklysalar*

if hours <= 40
 wage * hours
else if hours > 40
 40 * wage +
 (hours - 40) *
 wage * 1.5

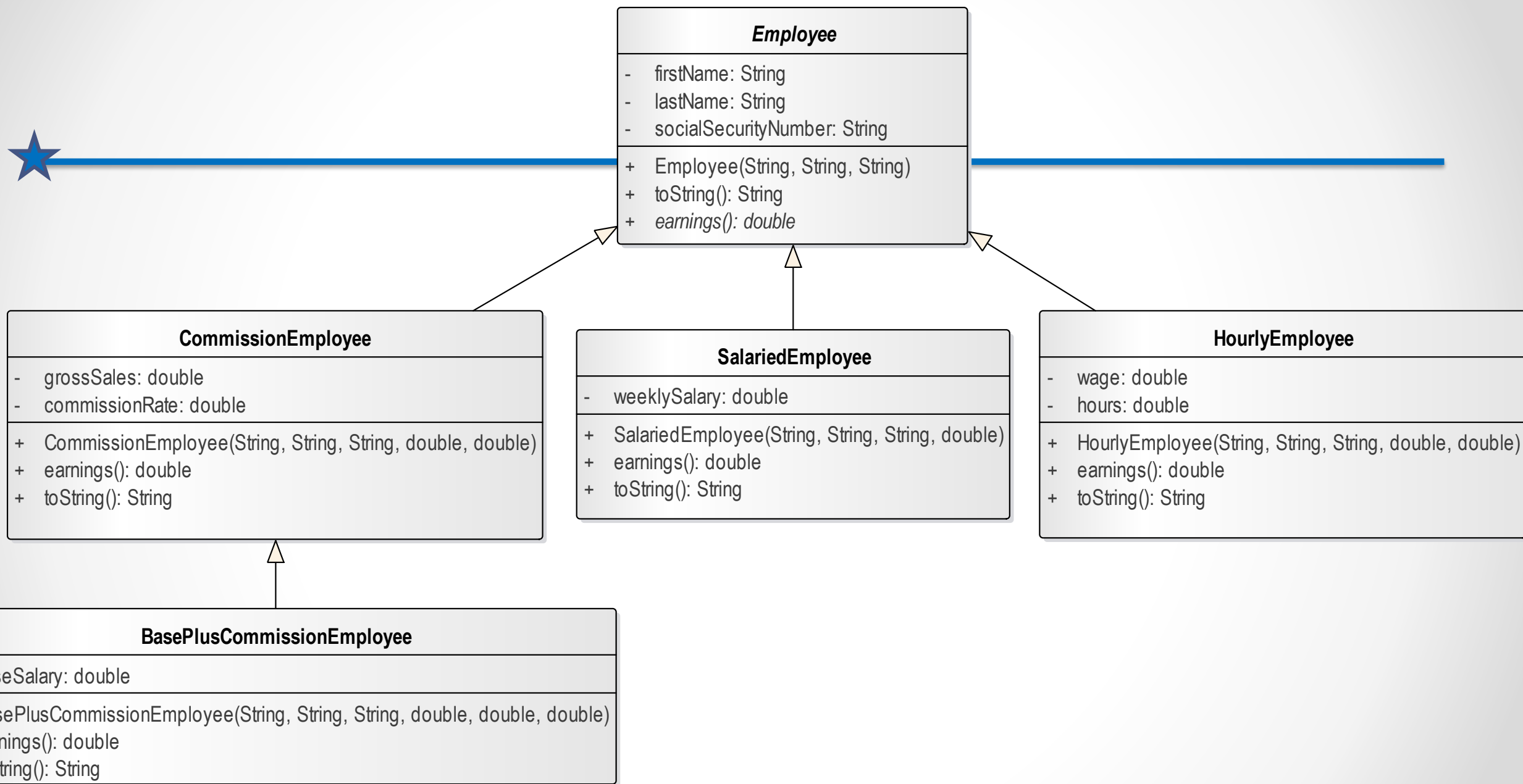
hourly employee: *firstName lastName*
social security number: *SSN*
hourly wage: *wage*; hours worked: *hours*

commissionRate *
grossSales

commission employee: *firstName lastName*
social security number: *SSN*
gross sales: *grossSales*;
commission rate: *commissionRate*

(commissionRate *
grossSales) +
baseSalary

base salaried commission employee:
 firstName lastName
social security number: *SSN*
gross sales: *grossSales*;
commission rate: *commissionRate*;
base salary: *baseSalary*



4.4. Interface



- Tại sao sử dụng Interface?
 - Trong Java chỉ có thừa kế đơn (có nghĩa là lớp con chỉ kế thừa từ duy nhất một lớp cha)
 - Interface cho phép Java thực hiện đa thừa kế

4.4. Interface

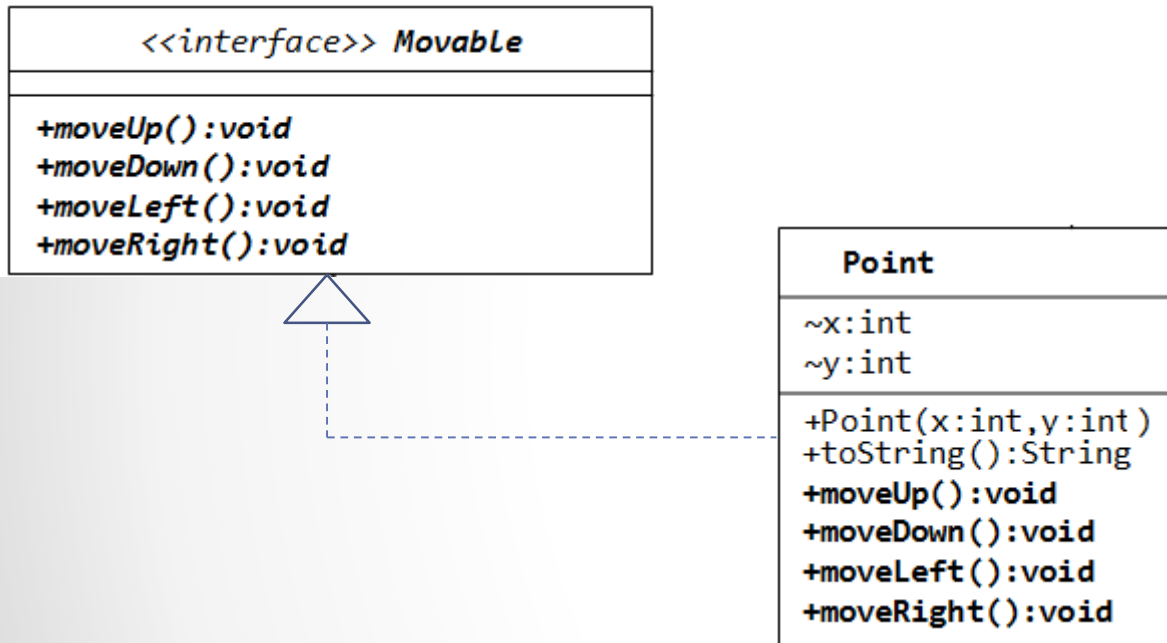
Khái niệm – Đặc trưng



- Trong Java, Interface là một kiểu dữ liệu tham chiếu tương tự như class nhưng chỉ có thể chứa **hằng số** và **phương thức trừu tượng**.
- Đặc điểm của Interface:
 - Interface là type (không phải class)
 - Không thể khởi tạo đối tượng từ Interface
- Là mối quan hệ “can-do”
- Một lớp khi đã “**implement**” interface nào thì bắt buộc phải viết lại tất cả các phương thức trong interface đó (nếu không, lớp đó là abstract)

4.4. Interface

Ví dụ 1



```
interface Movable {
    // "public" and "abstract" are optional
    void moveUp();
    void moveDown();
    void moveLeft();
    void moveRight();
}
```

```
class Point implements Movable {
    private int x, y;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
    @Override
    public String toString() {
        return "(" + x + "," + y + ")";
    }
    @Override
    public void moveUp() { y--; }
    @Override
    public void moveDown() { y++; }
    @Override
    public void moveLeft() { x--; }
    @Override
    public void moveRight() { x++; }
}
```

4.4. Interface

Ví dụ 2



```
interface IRunnable {  
    void Run();  
}
```

```
interface IWorkable {  
    void Work();  
}
```

```
interface IEatable {  
    void Eat();  
}
```

```
interface IFlyable {  
    void Fly();  
}
```

```
class Person implements IRunnable, IWorkable, IEatable {  
    protected string _name;  
    protected DateTime _birthDate;  
    public void Run() {  
        Console.WriteLine("Person runing...");  
    }  
    public void Work() {  
        Console.WriteLine("Person working...");  
    }  
    public void Eat() {  
        Console.WriteLine("Person eating...");  
    }  
}
```

```
class Bird implements IEatable, IFlyable {  
    public void Eat() {  
        Console.WriteLine("Bird eating...");  
    }  
    public void Fly() {  
        Console.WriteLine("Bird flying...");  
    }  
}
```

4.4. Interface

Lớp trừu tượng và Interface



| Lớp trừu tượng | Interface |
|---|---|
| Có thể chứa thuộc tính | Chỉ có thể chứa hằng |
| Các phương thức có thể được hiện thực hoặc không | Các phương thức không được hiện thực |
| Trong kế thừa, quan hệ là “is-a” | Trong kế thừa, quan hệ là “can-do” |
| Một lớp không thể kế thừa từ nhiều lớp trừu tượng | Có thể thừa kế từ 1 hoặc nhiều interfaces khác. Một lớp có thể hiện thực nhiều interface |

4.4. Interface

Case study 1




- Biết một sinh viên gồm các thuộc tính: mã, tên, tuổi, địa chỉ. Sắp xếp mảng sinh viên theo tên tăng dần, *sử dụng interface Comparable*:

```
public interface Comparable<T>
{
    public int compareTo(T other);
}
```

- Nếu sắp tăng dần:
 - Trả về 0 nếu this = other
 - Trả về một số dương nếu this > other
 - Trả về một số âm nếu this < other

4.4. Interface

Case study 1: Solution



```
class Student implements Comparable<Student>
{
    //...
    public int compareTo(Student o) {
        // sort student's name by ASC
        return this.getName().compareToIgnoreCase(o.getName());
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        Student[] listStudents = new Student[4];
        listStudents[0] = new Student(1, "Vinh", 19, "Hanoi");
        listStudents[1] = new Student(2, "Hoa", 19, "Hanoi");
        listStudents[2] = new Student(3, "Phu", 20, "Hanoi");
        listStudents[3] = new Student(4, "Quy", 22, "Hanoi");
        Arrays.sort(listStudents);
        for (Student student : listStudents) {
            System.out.println(student.toString());
        }
    }
}
```

4.4. Interface

Case study 2



- Sắp xếp mảng sinh viên theo từng tiêu chí mã, tên, tuổi. *Sử dụng interface Comparator.*

| |
|--|
| <<interface>> Comparator<T> |
| +compare(T o1, T o2): int |

Ghi chú: Sử dụng trong trường hợp không thể Comparable, hoặc muốn định nghĩa thêm các thứ tự khác.

4.4. Interface

Case study 2: Solution



```
public class Test {  
    public static void main(String[] args) {  
        Student[] listStudents = new Student[4];  
        listStudents[0] = new Student(1, "Vinh", 19, "Hanoi");  
        listStudents[1] = new Student(2, "Hoa", 19, "Hanoi");  
        listStudents[2] = new Student(3, "Phu", 20, "Hanoi");  
        listStudents[3] = new Student(4, "Quy", 22, "Hanoi");  
        Arrays.sort(listStudents, new Comparator<Student>() {  
            @Override  
            public int compare(Student o1, Student o2) {  
                return o1.getName().compareToIgnoreCase(o2.getName());  
            }  
        });  
        for (Student student : listStudents) {  
            System.out.println(student.toString());  
        }  
    }  
}
```



Review questions 2

1. Interface khác class như thế nào.
2. Một lớp có thể hiện thực được bao nhiêu Interface.
3. Một lớp muốn hiện thực một Interface thì phải làm thế nào.
4. Phát biểu nào đúng/sai, vì sao:
 - a. Interface chỉ bao gồm các phương thức trừu tượng.
 - b. Phương thức trừu tượng là phương thức không có hiện thực.
 - c. Tất cả các phương thức trong Interface phải là trừu tượng.
 - d. Một lớp implement Interface nào thì chỉ chứa những phương thức của Interface đó.
 - e. Nhiều lớp có thể implement cùng một Interface.
6. Làm thế nào để sắp xếp danh sách các phần tử.
7. Khi nào sử dụng Comparable, Comparator.



4.5. Đa hình (Polymorphism)

- Java cung cấp các hình thức đa hình:
 - **Override**: cho phép phương thức lớp con định nghĩa lại phương thức của lớp cha, phương thức lớp con có thể được gọi từ tham chiếu của lớp cha
 - **Overloading**: cho phép các phương thức trong cùng một lớp có cùng tên nhưng khác kiểu và tham số
 - **Dynamic binding**: lời gọi phương thức được quyết định khi chương trình thực hiện (run-time), phiên bản của phương thức phù hợp với đối tượng được gọi

4.5. Đa hình

Ví dụ



```
class OverloadingDemo
{
    public int add(int x, int y)
    { //method 1
        return x+y;
    }
    public int add(int x, int y, int z)
    { //method 2
        return x+y+z;
    }
    public int add(double x, int y)
    { //method 3
        return (int)x+y;
    }
}
```

4.5. Đa hình

Ví dụ



```
class Shape {  
    void draw() {  
        System.out.println("drawing...");  
    }  
}  
  
class Rectangle extends Shape {  
    void draw() {  
        System.out.println("drawing rectangle...");  
    }  
}  
  
class Circle extends Shape {  
    void draw() {  
        System.out.println("drawing circle...");  
    }  
}
```

```
class TestPolymorphism2 {  
    public static void main(String args[]) {  
        Shape s;  
        s = new Rectangle();  
        s.draw();  
        s = new Circle();  
        s.draw();  
        s = new Triangle();  
        s.draw();  
    }  
}
```

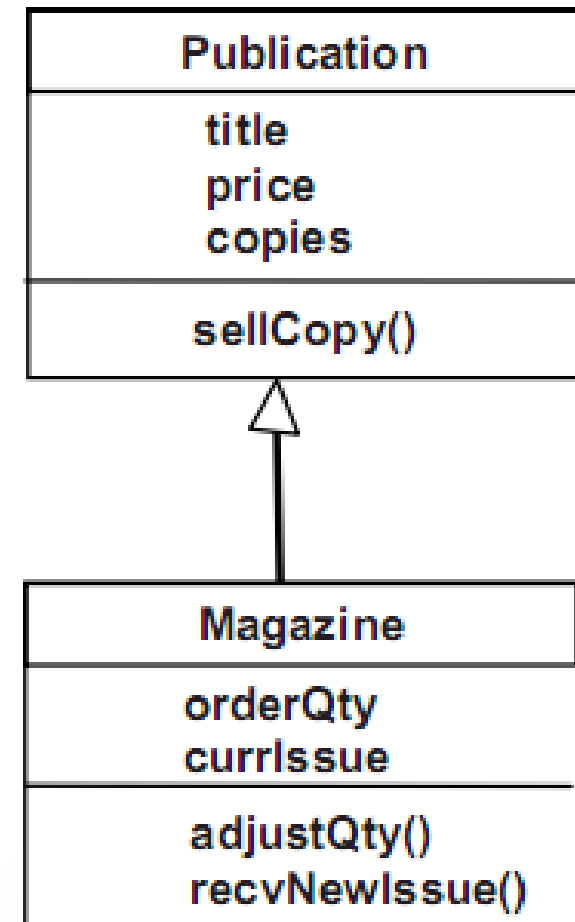
4.5. Đa hình

Quiz 1



- Cho mô hình lớp sau, cho biết cặp lệnh nào là hợp lệ.

- a) `Publication p = new Publication(...);`
`p.sellCopy();`
- b) `Publication p = new Publication(...);`
`p.recvNewIssue();`
- c) `Publication p = new Magazine(...);`
`p.sellCopy();`
- d) `Publication p = new Magazine(...);`
`p.recvNewIssue();`
- e) `Magazine m = new Magazine(...);`
`m.recvNewIssue();`

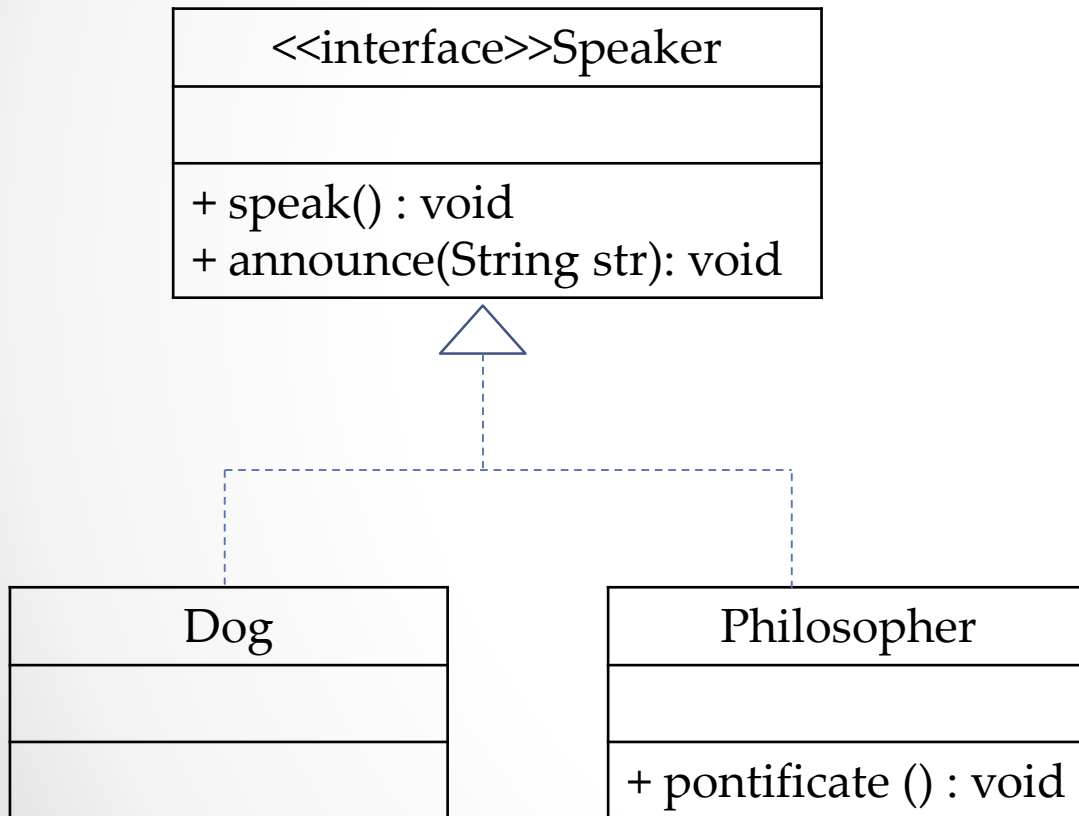


4.5. Đa hình

Quiz 2



Câu nào sau đây là đúng, giải thích.

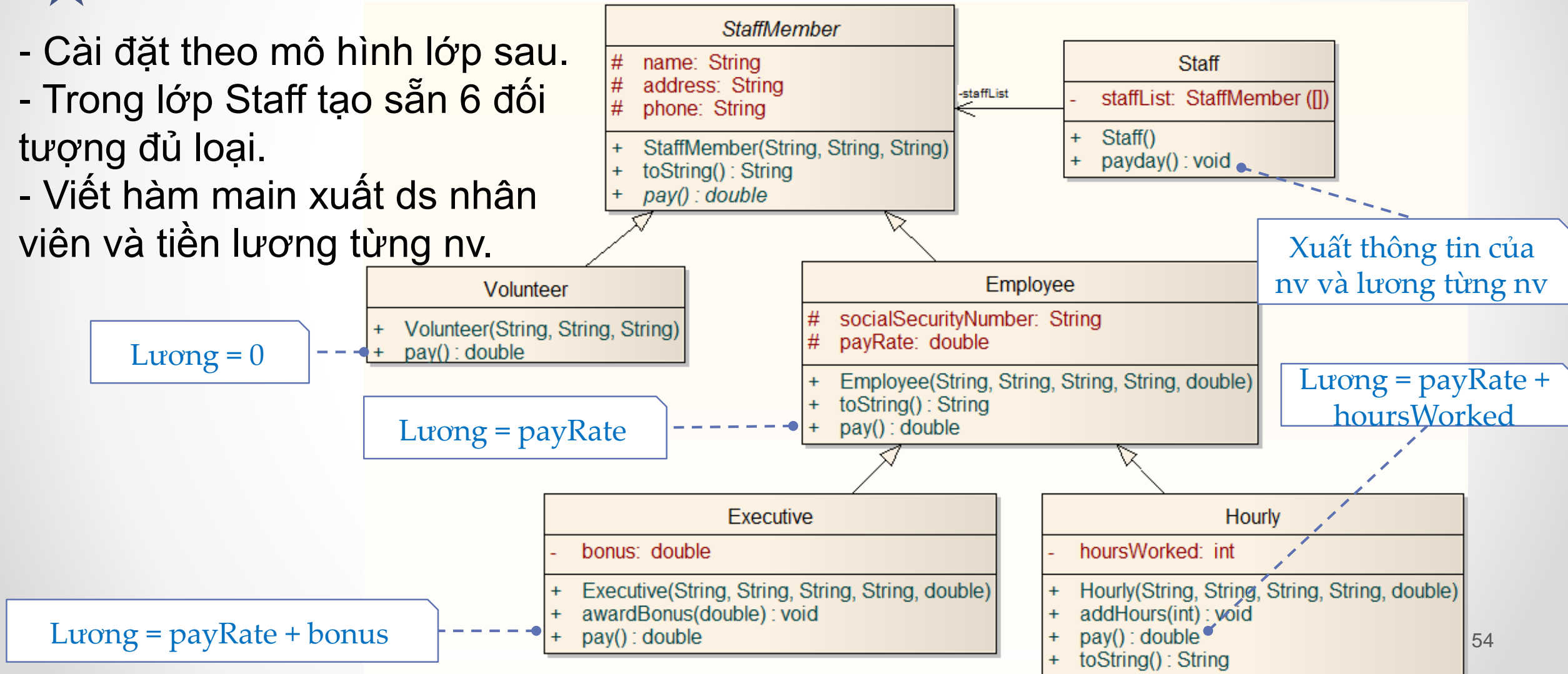


- a. `Speaker current = new Speaker();`
- b. `Speaker current = new Dog();`
- c. `Speaker first, second;`
`first = new Dog();`
`second = new Philosopher();`
`first.speak();`
`first = second;`
- d. `Speaker first = new Dog();`
`Philosopher second = new Philosopher();`
`second.pontificate();`
`first = second;`
- e. `Speaker first = new Dog();`
`Philosopher second = new Philosopher();`
`first = second;`
`second.pontificate();`
`first.pontificate();`

Case Study

(From Java Software Solution)

- Cài đặt theo mô hình lớp sau.
- Trong lớp Staff tạo sẵn 6 đối tượng đủ loại.
- Viết hàm main xuất ds nhân viên và tiền lương từng nv.



Case Study

Staff.java



```
public Staff() {  
    staffList = new StaffMember[6];  
    staffList[0] = new Executive("Sam", "123 Main Line", "555-0469", "123-45-6789", 2423.07);  
    staffList[1] = new Employee("Carla", "456 Off Line", "555-0101", "987-65-4321", 1246.15);  
    staffList[2] = new Employee("Woody", "789 Off Rocker", "555-0000", "010-20-3040",  
        1169.23);  
    staffList[3] = new Hourly("Diane", "678 Fifth Ave.", "555-0690", "958-47-3625", 10.55);  
    staffList[4] = new Volunteer("Norm", "987 Suds Blvd.", "555-8374");  
    staffList[5] = new Volunteer("Cliff", "321 Duds Lane", "555-7282");  
    ((Executive) staffList[0]).awardBonus(500.00);  
    ((Hourly) staffList[3]).addHours(40);  
}
```

Case Study

Staff.java



```
public void payday() {  
    double amount;  
    for (int count = 0; count < staffList.length; count++) {  
        System.out.println( staffList[count] );  
        amount = staffList[count].pay();    // polymorphic  
        if (amount == 0.0)  
            System.out.println("Thanks!");  
        else  
            System.out.println("Paid: " + amount);  
        System.out.println("-----");  
    }  
}
```