



KẾ THỪA

Thời gian: 4 tiết

NỘI DUNG

- ✓ Tái sử dụng mã nguồn
- ✓ Kết tập
- ✓ Kế thừa (Inheritance)

- **Tái sử dụng mã nguồn:** Sử dụng lại các mã nguồn đã viết
 - **Lập trình cấu trúc:** Tái sử dụng hàm/chương trình con
 - **OOP:** Khi mô hình thế giới thực, tồn tại nhiều loại đối tượng có các thuộc tính và hành vi tương tự hoặc liên quan đến nhau
- Làm thế nào để tái sử dụng lớp đã viết?



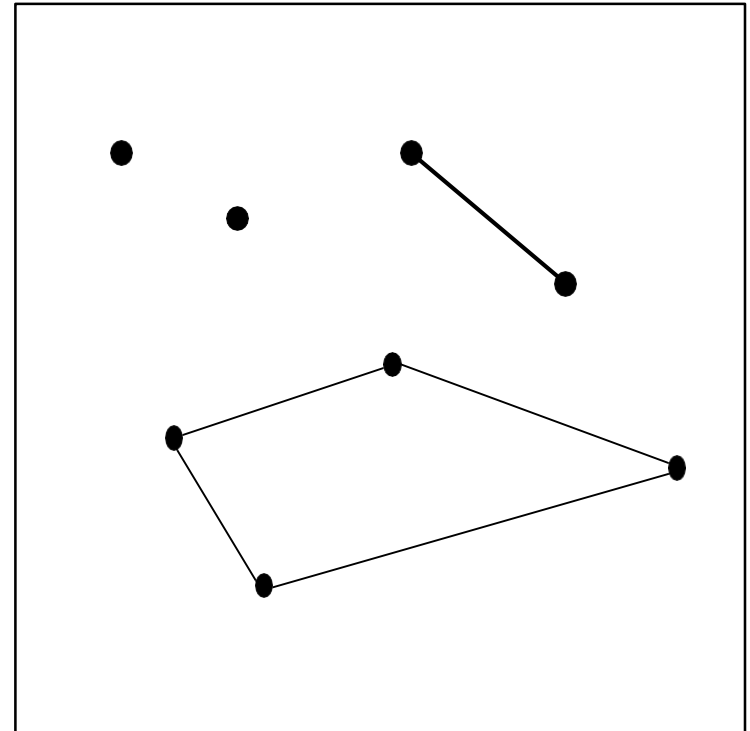
- Các cách sử dụng lại lớp đã có:
 - Sao chép lớp cũ thành 1 lớp khác => Dư thừa và khó quản lý khi có thay đổi
 - Tạo ra lớp mới là sự tập hợp hoặc sử dụng các đối tượng của lớp cũ đã có => Kết tập (Aggregation)
 - Tạo ra lớp mới trên cơ sở phát triển từ lớp cũ đã có => Kế thừa (Inheritance)

Ưu điểm của tái sử dụng mã nguồn:

- Giảm thiểu công sức, chi phí
- Nâng cao chất lượng phần mềm
- Nâng cao khả năng mô hình hóa thế giới thực
- Nâng cao khả năng bảo trì (maintainability)

- Khi xây dựng lớp, trường hợp các thuộc tính của một lớp là những đối tượng của một lớp khác.
- Hiện tượng đó gọi là kết tập (aggregation) trong Java.
- Khi một đối tượng được tạo mới, các thuộc tính của đối tượng đó đều phải được khởi tạo và gán những giá trị tương ứng.
- Các đối tượng thành phần được khởi tạo trước. Các phương thức khởi tạo các đối tượng của lớp thành phần được thực hiện trước.

- Ví dụ:
 - Điểm
 - Tứ giác gồm 4 điểm
→ Kết tập
- Kết tập
 - Quan hệ chứa/có ("has-a") hoặc là một phần (is-a-part-of)



- Kết tập (aggregation)
 - Tạo ra các đối tượng của các lớp có sẵn trong lớp mới → thành viên của lớp mới.
 - Kết tập tái sử dụng thông qua đối tượng
- Lớp mới
 - Lớp toàn thể (Aggregate/Whole),
- Lớp cũ
 - Lớp thành phần (Part).

- Lớp toàn thể chứa đối tượng của lớp thành phần
 - Là một phần (is-a-part of) của lớp toàn thể
 - Tái sử dụng các thành phần dữ liệu và các hành vi của lớp thành phần thông qua đối tượng thành phần

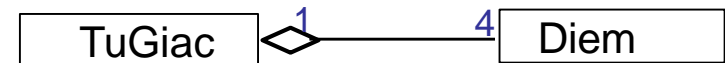
Thứ tự khởi tạo trong kết tập

- Khi một đối tượng được tạo mới, các thuộc tính của đối tượng đó đều phải được khởi tạo và gán những giá trị tương ứng.
- Các đối tượng thành phần được khởi tạo trước
 - Các phương thức khởi tạo của các lớp của các đối tượng thành phần được thực hiện trước

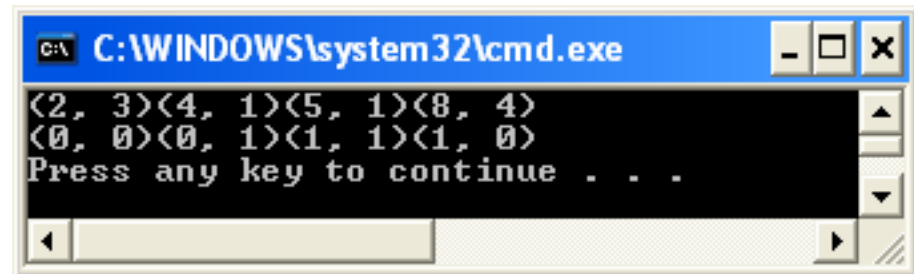
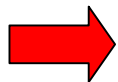
Minh
họa
trên
Java

```
class Diem {  
    private int x, y;  
    public Diem(){}  
    public Diem(int x, int y) {  
        this.x = x; this.y = y;  
    }  
    public void setX(int x) { this.x = x; }  
    public int getX() { return x; }  
    public void printDiem() {  
        System.out.print("(" + x + ", " + y + ")");  
    }  
}
```

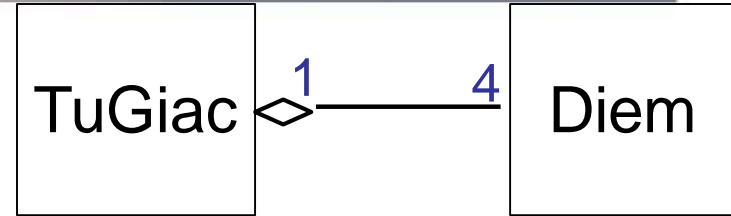
```
class TuGiac {  
  
    private Diem d1, d2;  
    private Diem d3, d4;  
  
    public TuGiac(Diem p1, Diem p2, Diem p3, Diem p4){  
        d1 = p1; d2 = p2; d3 = p3; d4 = p4;  
    }  
  
    public TuGiac(){  
        d1 = new Diem(); d2 = new Diem(0,1);  
        d3 = new Diem (1,1); d4 = new Diem (1,0);  
    }  
  
    public void printTuGiac(){  
  
        d1.printDiem();      d2.printDiem();  
        d3.printDiem();      d4.printDiem();  
  
        System.out.println();  
    }  
}
```



```
public class Test {  
    public static void main(String arg[]) {  
        Diem d1 = new Diem(2,3); Diem d2 = new Diem(4,1);  
        Diem d3 = new Diem (5,1); Diem d4 = new Diem (8,4);  
        TuGiac tg1 = new TuGiac(d1, d2, d3, d4);  
        TuGiac tg2 = new TuGiac();  
        tg1.printTuGiac();  
        tg2.printTuGiac();  
    }  
}
```



```
C:\WINDOWS\system32\cmd.exe  
<2, 3><4, 1><5, 1><8, 4>  
<0, 0><0, 1><1, 1><1, 0>  
Press any key to continue . . .
```



```
class TuGiac {
    private Diem[] diem = new Diem[4];
    public TuGiac(Diem p1, Diem p2,
                  Diem p3, Diem p4) {
        diem[0] = p1; diem[1] = p2;
        diem[2] = p3; diem[3] = p4;
    }
    public void printTuGiac() {
        diem[0].printDiem(); diem[1].printDiem();
        diem[2].printDiem(); diem[3].printDiem();
        System.out.println();
    }
}
```



Kế thừa

Khái niệm

- ❖ Kế thừa là một đặc điểm của ngôn ngữ dùng để biểu diễn mối quan hệ đặc biệt hóa – tổng quát hóa giữa các lớp. Các lớp được trừu tượng hóa và được tổ chức thành một sơ đồ phân cấp lớp.
- ❖ Sự kế thừa là một mức cao hơn của trừu tượng hóa, cung cấp một cơ chế gom chung các lớp có liên quan với nhau thành một mức khái quát hóa đặc trưng cho toàn bộ các lớp nói trên.

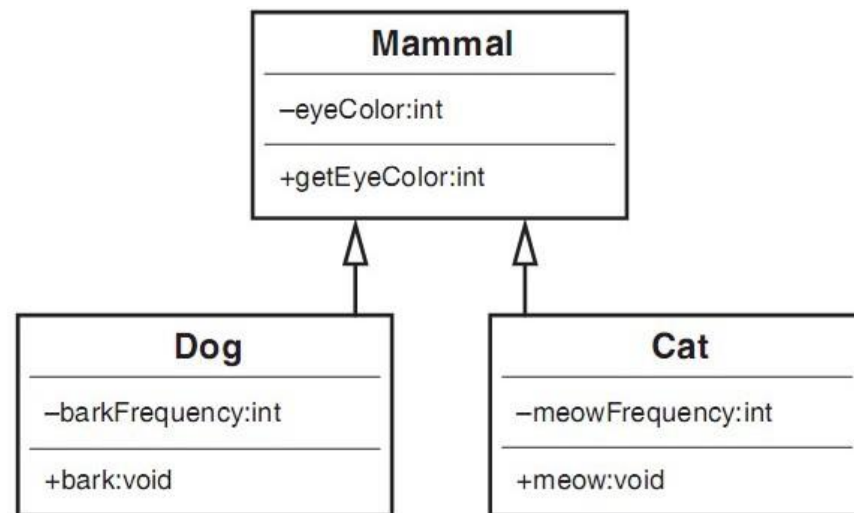
- ❖ Các lớp với các đặc điểm tương tự nhau có thể được tổ chức thành một sơ đồ phân cấp kế thừa (cây kế thừa).
- ❖ Kế thừa được sử dụng thông dụng nhất để biểu diễn mối quan hệ 1-1.
 - Một sinh viên là một người
 - Một hình tròn là một hình ellipse
 - Một tam giác là một đa giác
 - ...

- ❖ Kế thừa cho phép xây dựng lớp mới từ lớp đã có.
- ❖ Kế thừa cho phép tổ chức các lớp chia sẻ mã chương trình chung, nhờ vậy có thể dễ dàng sửa chữa, nâng cấp hệ thống.

Biểu diễn kế thừa biểu đồ lớp

❖ Cho phép định nghĩa lớp mới từ lớp đã có.

- Lớp mới gọi là lớp con (subclass) hay lớp dẫn xuất (derived class)
- Lớp đã có gọi là lớp cha (superclass) hay lớp cơ sở (base class).

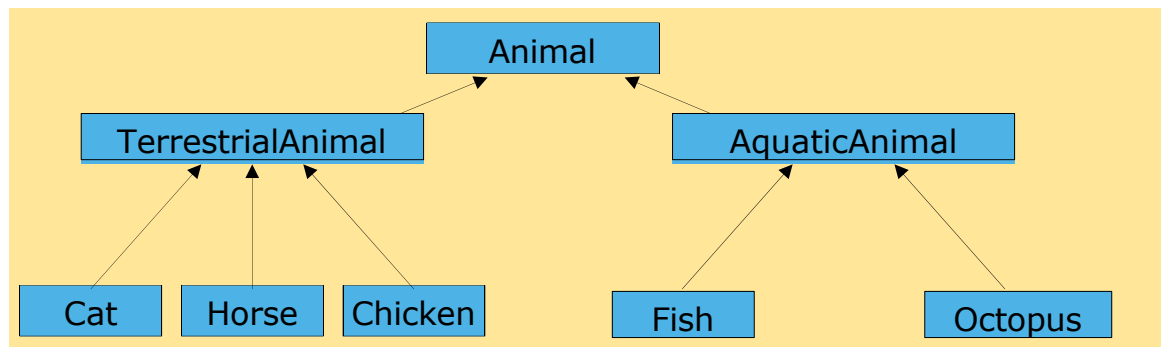


Biểu diễn kế thừa biểu đồ lớp

❖ Thừa kế cho phép:

- Nhiều lớp có thể dẫn xuất từ một lớp cơ sở
- Một lớp có thể là dẫn xuất của nhiều lớp cơ sở

❖ Thừa kế không chỉ giới hạn ở một mức: Một lớp dẫn xuất có thể là lớp cơ sở cho các lớp dẫn xuất khác



- 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 HìnhVuong **extends** TuGiac {
 ...
 }

Từ khóa *super*

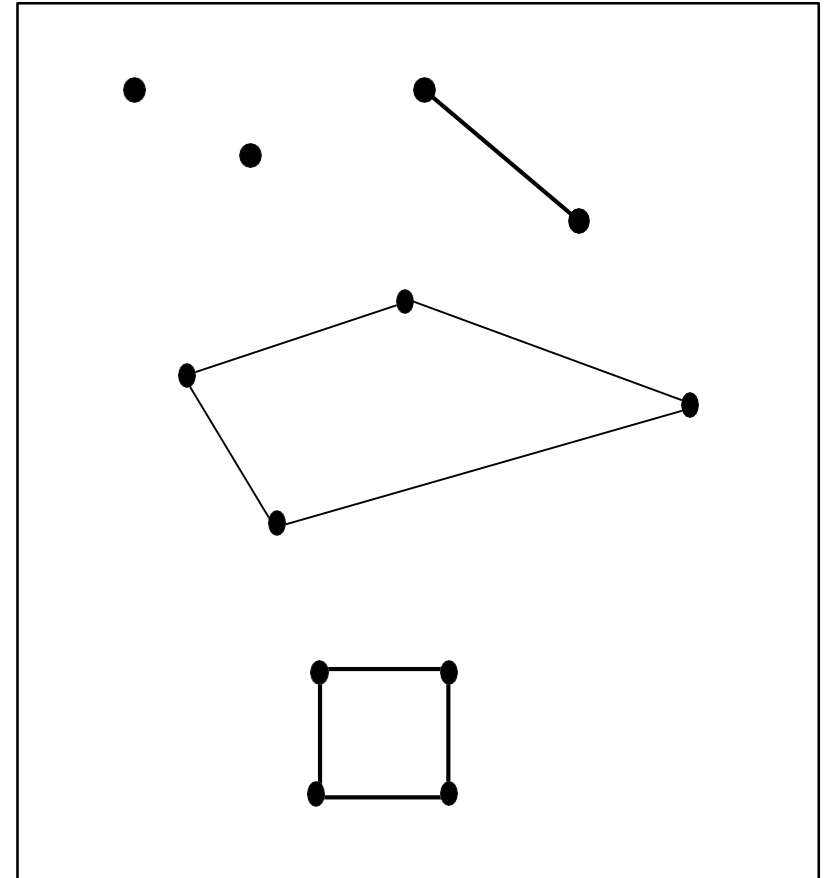
- Tái sử dụng các đoạn mã của lớp cha trong lớp con
- Gọi phương thức khởi tạo *super* (danh sách tham số);
 - Bắt buộc nếu lớp cha không có phương thức khởi tạo mặc định
- Gọi các phương thức của lớp cha *super.tênPt*(danh sách tham số);

```
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;
    }
}
```

- Ví dụ:
 - Điểm
 - Tứ giác gồm 4 điểm
→ Kết tập
 - Tứ giác
 - Hình vuông
→ Kế thừa



- Kế thừa (Inherit, Derive)
 - Tạo lớp mới bằng cách phát triển lớp đã có.
 - Lớp mới kế thừa những gì đã có trong lớp cũ và phát triển những tính năng mới.
- Lớp cũ:
 - Lớp cha (parent, superclass), lớp cơ sở (base class)
- Lớp mới:
 - Lớp con (child, subclass), lớp dẫn xuất (derived class)

■ Lớp con

- Là một loại (is-a-kind-of) của lớp cha
- Tái sử dụng bằng cách kế thừa các thành phần dữ liệu và các hành vi của lớp cha
- Chi tiết hóa cho phù hợp với mục đích sử dụng mới
 - Extension: Thêm các thuộc tính/hành vi mới
 - Redefinition (Method Overriding): Chỉnh sửa lại các hành vi kế thừa từ lớp cha

- Chỉ định truy cập protected
- Thành viên protected trong lớp cha được truy cập trong:
 - Các thành viên lớp cha
 - **Các thành viên lớp con**
 - Các thành viên các lớp cùng thuộc 1 package với lớp cha
- Lớp con có thể kế thừa được gì?
 - Kế thừa được các thành viên được khai báo là public và protected của lớp cha.
 - Không kế thừa được các thành viên private.
 - Các thành viên có chỉ định truy cập mặc định nếu lớp cha cùng gói với lớp con

	public	Không có	protected	private
Cùng lớp cha	Yes	Yes	Yes	Yes
Lớp con cùng gói	Yes	Yes	Yes	No
Lớp con khác gói	Yes	No	Yes	No
Khác gói, non-inher	Yes	No	No	No

- Các trường hợp không được phép kế thừa:
 - Các phương thức khởi tạo và hủy
 - Làm nhiệm vụ khởi đầu và gỡ bỏ các đối tượng
 - Chúng chỉ biết cách làm việc với từng lớp cụ thể
 - Toán tử gán =
 - Làm nhiệm vụ giống như phương thức khởi tạo

Kết tập và kế thừa

- So sánh kết tập và kế thừa?
 - Giống nhau
 - Đều là kỹ thuật trong OOP để tái sử dụng mã nguồn
 - Khác nhau?

Kế thừa

- Kế thừa **tái sử dụng** thông qua **lớp**.
 - Tạo lớp mới bằng cách phát triển lớp đã có
 - Lớp con kế thừa dữ liệu và hành vi của lớp cha
- Quan hệ "**là một loại**" ("is a kind of")
- Ví dụ: Ô tô là một loại phương tiện vận tải

Kết tập

- Kết tập **tái sử dụng** thông qua **đối tượng**.
 - Tạo ra lớp mới là tập hợp các đối tượng của các lớp đã có
 - Lớp toàn thể có thể sử dụng dữ liệu và hành vi thông qua các đối tượng thành phần
- Quan hệ "**là một phần**" ("is a part of")
- Ví dụ: Bánh xe là một phần của Ô tô

- Các lớp được nhóm lại thành package
 - Package bao gồm một tập hợp các lớp có quan hệ logic với nhau
- Gói (package) giống như thư mục giúp:
 - Tổ chức và xác định vị trí lớp dễ dàng và sử dụng các lớp một cách phù hợp.
 - Tránh cho việc đặt tên lớp bị xung đột (trùng tên)
 - Các package khác nhau có thể chứa các lớp có cùng tên
 - Bảo vệ các lớp, dữ liệu và phương thức ở mức rộng hơn so với mối quan hệ giữa các lớp.
- Còn được gọi là *không gian tên (namespace)* trong một số ngôn ngữ lập trình (C++...)

- Một số package có sẵn của Java: java.lang, javax.swing, java.io...
- Package có thể do ta tự đặt
 - Cách nhau bằng dấu "."
 - Quy ước sử dụng ký tự thường để đặt tên package
 - Tên gói phải được viết trên cùng của file mã nguồn
- Chỉ được phép có 1 câu khai báo gói trong mỗi file mã nguồn, và khai báo này sẽ được áp dụng cho tất cả các dữ liệu trong file đó.
- Một gói có thể được đặt trong một gói khác
 - Phân cách bằng dấu .
 - Ví dụ `package laptrinhhdt.oop.sv;`

- Cú pháp khai báo:

```
package tenpackage;  
chi_dinh_truy_cap class TenLop {  
    // Than lop  
}
```

- **chi_dinh_truy_cap:**

- public: Lớp có thể được truy cập từ bất cứ đâu, kể cả bên ngoài package chứa lớp đó.
- Không chỉ định: Lớp chỉ có thể được truy cập từ bên trong package chứa lớp đó.

```
package oop.sinhvien;  
    public class Student {  
        ...  
    }
```



Xin cảm ơn!

TRẦN QUÝ NAM

*Giảng viên Khoa CNTT
namtq@dainam.edu.vn*