Giáo trình: Xóa mù OOP

(Lập trình hướng đối tượng)

Ngôn ngữ lập trình: Java

Tác giả: Nguyên Nguyễn

Với sự hỗ trợ của AI

Mục lục

A,Các thứ tạo nên class cơ bản

I, Cấu trúc class cơ bản

```
public class Tenlop {
    // code
}
```

II, Các khoá thế hiện mức độ truy cập

1.Các mức truy cập

-**private:** Được ví như két sắt. Chỉ mình mới biết mật mã

-**public:** Cửa chính mở toang. Ai cũng có thể vào. Kể cả thằng trộm cũng có thể vào được.

-**protected:** Cửa chính chỉ được mở cho con cháu họ hàng.

2, Các mức truy cập được sử dụng như thế nào

-**Private**: chủ yếu sẽ để các biến thành viên

```
public class Nguoi {  no usages
    private String ten;       // Thuộc tính private  no usages
    private int tuoi;         // Thuộc tính private  no usages

    private void xuLyDuLieu() {  no usages
        // Phương thức private - chỉ gọi được bên trong class
    }
}
```

-**Public**: chủ yếu sẽ để các hàm xử lí.

```
public class Nguoi {  no usages
    public String  ten;  no usages
    public int lop;  no usages

    public void display(){  no usages

    }
}
```

Lưu ý: Trong hàm nhap() sẽ có 2 hàm làm cho việc viết hàm nhap() trở nên trơn tru hơn.

- **Scanner.nextLine()**: Dùng để xóa kí tự thừa trong bộ đệm

```
public class Demo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Nhập tuổi: ");
        int tuoi = scanner.nextInt();

        scanner.nextLine(); // Bỏ qua ký tự xuống dòng còn lại
        System.out.print("Nhập tên: ");
        String ten = scanner.nextLine();
    }
```

→Ta sẽ đặt hàm cin.igonre() giữa kiểu dữ liệu int, double, float với string

Scanner.nextLine(): dùng để nhập cả khoảng trắng.

```
Scanner scanner = new Scanner(System.in);  1 usage
String name = scanner.nextLine(); // Nhập nguyên dòng (bao gồm dấu cách)
```

III, Constructor: hàm khởi tạo

1) Constructor không tham số

Cú pháp:

```
public TenLop() {  no usages
    // Khối lệnh khởi tạo mặc định
}
```

Ví dụ:

```java
public class HocSinh {  no usages
    String ten;  1 usage
    int tuoi;  1 usage

    // Constructor không tham số
    public HocSinh() {  no usages
        ten = "Không rõ";
        tuoi = 0;
    }
}
```

2) Constructor có tham số

Cú pháp:

```java
public TenLop(KieuDuLieu1 thamSo1, KieuDuLieu2 thamSo2) {  no usages
    // Gán giá trị cho các biến thành viên
    this.bien1 = thamSo1;
    this.bien2 = thamSo2;
}
```

Ví dụ:

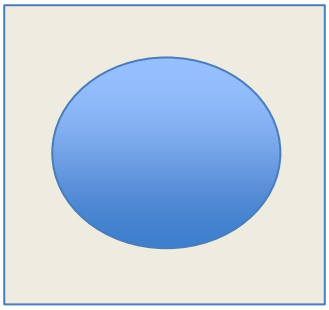
```java
public class HocSinh {  no usages
    String ten;  1 usage
    int tuoi;  1 usage

    // Constructor có tham số
    public HocSinh(String t, int a) {  no usages
        ten = t;
        tuoi = a;
    }
```

IV) Destructor(Không áp dụng bên java)

B) 4 tính chất của lập trình hướng đối tượng

1, Tính đóng gói

Ví dụ đơn giản về tính đóng gói: Chúng ta có một ô vuông khép kín. Ở trong đó có 1 dấu chấm. Nếu chúng ta muốn lấy dấu chấm đó ra, chúng ta sẽ dùng hàm getter() để lấy dấu chấm đó là và dùng setter() để cập nhật giá trị.



Cú pháp hàm getter():

```
public class TenLop {  no usages
    private KieuDuLieu tenThuocTinh;  1 usage

    public KieuDuLieu getTenThuocTinh() {  no usages
        return tenThuocTinh;
    }
}
```

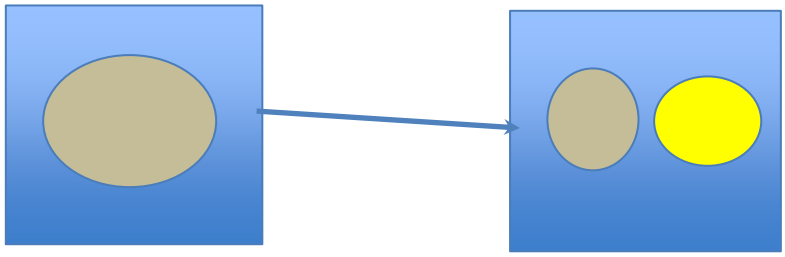
Cú pháp hàm setter():

```
public class TenLop {  no usages
    private KieuDuLieu tenThuocTinh;  1 usage

    public void setTenThuocTinh(KieuDuLieu value) {  no usages
        this.tenThuocTinh = value;
    }
}
```

Ghi chú: Nếu bạn là người mới học lập trình hoặc đang học lại từ đầu, thì có thể tạm thời sử dụng **public** cho tất cả các thuộc tính để dễ quan sát và thử nghiệm. Tuy nhiên, khi đã hiểu rõ hơn về lập trình hướng đối tượng, bạn nên sử dụng **private** để đảm bảo an toàn cho dữ liệu – đây là nguyên tắc quan trọng trong **tính đóng gói (Encapsulation)**.

2,Tính kế thừa

Ví dụ về tính kế thừa: Từ tính đóng gói, ta sẽ vẽ thêm ô thứ 2. Sau đó ta sẽ lấy chấm tròn từ ô thứ nhất sang ô thứ 2 và sẽ tô thêm 1 chấm tròn nữa. Đó chính là Tính kế thừa.



Cú pháp để khai báo tính kế thừa:

```
class LopCha {  1 usage  1 inheritor
    // thuộc tính và phương thức của lớp cha
}

class LopCon extends LopCha {  no usages
    // thêm các thuộc tính và phương thức riêng
}
```

—-> Sử dụng từ khóa extends để kế thừa.

3, Tính đa hình.

Có 2 loại đa hình chính:

Nạp chồng (Overloading): cùng tên, khác tham số (compile time).

Ghi đè (Overriding): lớp con định nghĩa lại phương thức lớp cha (runtime).

Ví dụ về nạp chồng hàm:

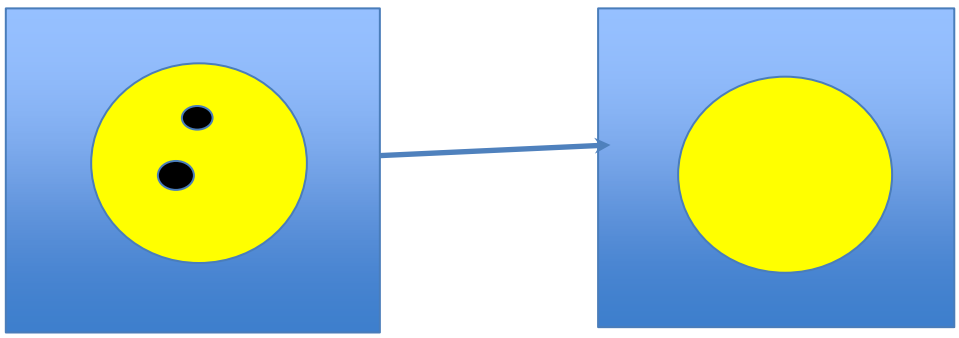
```
class MayTinh {  2 usages
    int cong(int a, int b) {  1 usage
        return a + b;
    }

    double cong(double a, double b) {  1 usage
        return a + b;
    }
}

public class Main {
    public static void main(String[] args) {
        MayTinh mt = new MayTinh();
        System.out.println(mt.cong( a: 2,  b: 3));         // 5
        System.out.println(mt.cong( a: 2.5,  b: 3.5));     // 6.0
    }
}
```

Ví dụ về ghi đè:

```java
class DongVat {  2 usages  1 inheritor
    void keu() {  1 usage  1 override
        System.out.println("Động vật kêu...");
    }
}

class Meo extends DongVat {  1 usage
    @Override  1 usage
    void keu() {
        System.out.println("Meo meo");
    }
}

public class Main {
    public static void main(String[] args) {
        DongVat dv = new Meo();    // đa hình
        dv.keu();                  // "Meo meo"
    }
}
```

4, Tính trừu tượng:

Tính trừu tượng: Giúp **ẩn giấu logic phức tạp**, chỉ hiển thị phần cần thiết.



C++ dùng **abstract class**

```java
abstract class DongVat {  1 usage  1 inheritor
    abstract void keu();  // phương thức trừu tượng

    void an() {  no usages
        System.out.println("Đang ăn...");
    }
}

class Cho extends DongVat {  no usages
    @Override  no usages
    void keu() {
        System.out.println("Gâu gâu");
    }
}
```

C) Các đơn vị kiến thức khác

I) Nạp chồng toán tử:(Không áp dụng bên java)

II) Hàm bạn , lớp bạn. (Không áp dụng bên)

III) Biến static

Dùng chung cho tất cả các đối tượng. Không phụ thuộc vào từng instance.

Biến static: dùng để đếm số đối tượng chẳng hạn.

```
static <kiểu_dữ_liệu> <tên_biến>;
```

Hàm static: gọi trực tiếp qua class.

```
static <kiểu_trả_về> <tên_hàm>() {
        // nội dung
        }
```

Ví dụ:

```
class SinhVien {  4 usages
    String ten;  2 usages
    static String truong = "Đại học Bách Khoa";  1 usa

    public SinhVien(String ten) {  2 usages
        this.ten = ten;
    }

    void hienThi() {  2 usages
        System.out.println(ten + " - " + truong);
    }
}

public class Main {
    public static void main(String[] args) {
        SinhVien sv1 = new SinhVien( ten: "An");
        SinhVien sv2 = new SinhVien( ten: "Bình");

        sv1.hienThi(); // An - Đại học Bách Khoa
        sv2.hienThi(); // Bình - Đại học Bách Khoa
    }
}
```

Lưu ý: `main()` trong Java cũng là một phương thức static!

D) Phần Main()

I) Khai báo một đối tượng

1) Tính đóng gói

a) Cấu trúc

```
public class Main {
    public static void main(String[] args) {
        HocSinh hs1 = new HocSinh();        // khai báo 1 đối tượng
        hs1.setTen("Nguyen Van A");         // cập nhật tên
        System.out.println(hs1.getTen());   // in ra tên
    }
```

b) Ví dụ:

```
class HocSinh {  2 usages
    private String ten; // thuộc tính private  2 usages

    // setter
    public void setTen(String ten) {  1 usage
        this.ten = ten;
    }

    // getter
    public String getTen() {  1 usage
        return ten;
    }
}

public class Main {
    public static void main(String[] args) {
        HocSinh hs1 = new HocSinh();        // khai báo 1 đối tượng
        hs1.setTen("Nguyen Van A");         // cập nhật tên
        System.out.println(hs1.getTen());   // in ra tên
    }
}
```

2) Tính kế thừa

a) Cấu trúc

```java
public class Main {
    public static void main(String[] args) {
        Chim chim1 = new Chim();      // khai báo 1 đối tượng từ lớp kế thừa
        chim1.keu();                  // dùng phương thức từ lớp cha
        chim1.bay();                  // dùng phương thức của lớp con
    }
```

b) Vídụ:

```java
class DongVat {  3 usages  2 inheritors
    public void keu() {  1 usage  1 override
        System.out.println("Động vật kêu...");
    }
}

class Chim extends DongVat {  2 usages
    public void bay() {  1 usage
        System.out.println("Chim đang bay...");
    }
}

public class Main {
    public static void main(String[] args) {
        Chim chim1 = new Chim();      // khai báo 1 đối tượng từ lớp kế thừa
        chim1.keu();                  // dùng phương thức từ lớp cha
        chim1.bay();                  // dùng phương thức của lớp con
    }
}
```

3) Tính trừu tượng:

a)Cấu trúc:

```java
public class Main {
    public static void main(String[] args) {
        DongVat dv = new Cho(); // đối tượng cha trỏ đến con
        dv.keu();               // đa hình: chạy phương thức của lớp con
    }
```

b)Ví dụ:

```java
class DongVat {  3 usages  2 inheritors
    public void keu() {  1 usage  1 override
        System.out.println("Động vật phát ra âm thanh...");
    }
}

class Cho extends DongVat {  1 usage
    @Override  1 usage
    public void keu() {
        System.out.println("Gâu gâu!");
    }
}

public class Main {
    public static void main(String[] args) {
        DongVat dv = new Cho(); // đối tượng cha trỏ đến con
        dv.keu();               // đa hình: chạy phương thức của lớp con
    }
}
```

4) Tính trừu tượng

a)Cấutrúc:

```java
public class Main {
    public static void main(String[] args) {
        Hinh h = new HinhTron(); // đối tượng cha trỏ tới lớp con
        h.ve();                  // gọi phương thức cụ thể đã triển khai
    }
```

b)Ví dụ:

```java
abstract class Hinh {  2 usages  1 inheritor
    public abstract void ve(); // phương thức trừu tượng  1 usage  1 implement
}

class HinhTron extends Hinh {  1 usage
    @Override  1 usage
    public void ve() {
        System.out.println("Vẽ hình tròn");
    }
}

public class Main {
    public static void main(String[] args) {
        Hinh h = new HinhTron(); // đối tượng cha trỏ tới lớp con
        h.ve();                  // gọi phương thức cụ thể đã triển khai
    }
```

Typo:

II) Danh sách đối tượng

1)Khai báo:

a)**Khai báo dùng mảng (Array):**

```java
TenLop[] ds = new TenLop[soLuong];
// sau đó gán từng phần tử
ds[0] = new TenLop(...);
```

Cố định số lượng phần tử

Phải gán từng phần tử thủ công

—>Duyệt được bằng cả for thường và for-each.

b) **Khai báo dùng ArrayList (linh hoạt hơn).**

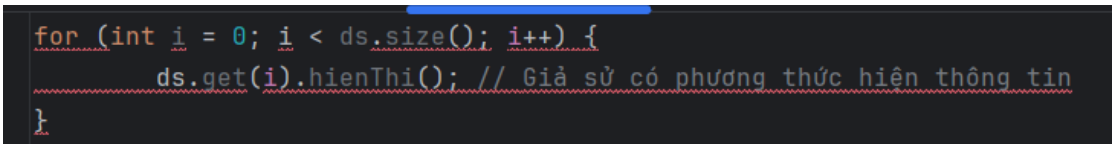
```java
ArrayList<TenLop> ds = new ArrayList<>();
ds.add(new TenLop(...));
        ds.add(new TenLop(...));
```

- Tự động mở rộng kích thước
- Thêm/xóa dễ dàng

→ Cũng duyệt được bằng for thường (với ds.get(i)) và for-each.

2) for thường:

a) Cấu trúc:

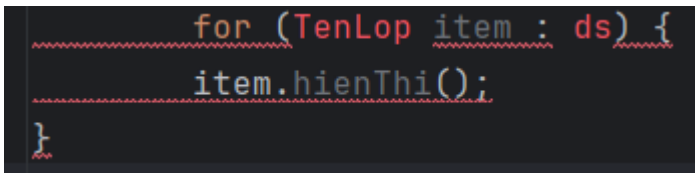


```
for (int i = 0; i < ds.size(); i++) {
        ds.get(i).hienThi(); // Giả sử có phương thức hiện thông tin
}
```

b) Lưu ý:

"Khung **for** từ min đến max giúp dễ điều chỉnh số lượng đối tượng cần nhập mà không bị giới hạn cứng. Đây là cách tổ chức linh hoạt, phù hợp với nhiều bài toán quản lý."

3) for- each



```
        for (TenLop item : ds) {
        item.hienThi();
}
```

E) Bài tập tổng hợp (dùng java)

Viết chương trình quản lý sinh viên với các yêu cầu sau:

-Tạo lớp SinhVien gồm các thuộc tính:

-Mã sinh viên,họ tên, điểm toán, điểm văn, điểm anh.

Các thuộc tính để **private.**

-Viết đầy đủ getter/setter, constructor, destructor

-Tạo hàm tinhDiemTrungBinh() cho sinh viên.

-Viết hàm in ra thông tin sinh viên.

-Tạo lớp SinhVienCNTT kế thừa từ **SinhVien**, thêm điểm lập trình.

-Ghi đè hàm tinhDiemTrungBinh() trong lớp **SinhVienCNTT.**

-Trong hàm main(), thực hiện:

Nhập danh sách sinh viên (tối thiểu 3 sinh viên).

In danh sách sinh viên ra màn hình.

Tìm sinh viên có điểm TB cao nhất.

Sắp xếp sinh viên theo điểm TB giảm dần.

Lời kết:

Giáo trình "Xóa mù OOP" được biên soạn với tâm huyết nhằm giúp các bạn dễ dàng tiếp cận lập trình hướng đối tượng bằng Java, theo cách đơn giản – dễ hiểu – dễ áp dụng.

Một phần nội dung trong giáo trình có sự hỗ trợ của trí tuệ nhân tạo (AI) để đảm bảo tính hệ thống, khoa học và rõ ràng. Tuy nhiên, toàn bộ cấu trúc, phong cách, ví dụ minh họa, bài tập... đều được thiết kế theo cách riêng – gần gũi với người học Việt Nam, đặc biệt là các bạn mới bắt đầu.

Hy vọng giáo trình này sẽ là người bạn đồng hành hữu ích trên hành trình học lập trình của bạn.
Đừng quên rằng, học lập trình không khó – quan trọng là học đúng cách và đúng lộ trình.

Chúc bạn học tốt và thành công!