

Session 10: Tính Trừu Tượng trong Java

Khám phá sức mạnh của Abstraction - một trong bốn trụ cột của lập trình hướng đối tượng

1

Interface

Giới thiệu và khai báo Interface trong Java

2

Abstract Class

Giới thiệu và khai báo Abstract Class

3

So Sánh

Phân biệt Abstract Class và Interface

4

Anonymous Class

Giới thiệu và ứng dụng Anonymous Class

Interface Là Gì?

Interface là một "hợp đồng" định nghĩa các hành vi mà một lớp phải triển khai. Nó chỉ chứa khai báo phương thức trừu tượng, tạo ra sự linh hoạt trong thiết kế hệ thống.



Chỉ Abstract Methods

Không có implementation cụ thể



Không Có Constructor

Không thể khởi tạo trực tiếp



Đa Kế Thừa

Implement nhiều interface cùng lúc

Khai Báo Interface

Cú pháp cơ bản

```
public interface TenInterface {  
    void phuongThuc1();  
    String phuongThuc2(int thamSo);  
    int HANG_SO = 100;  
}
```

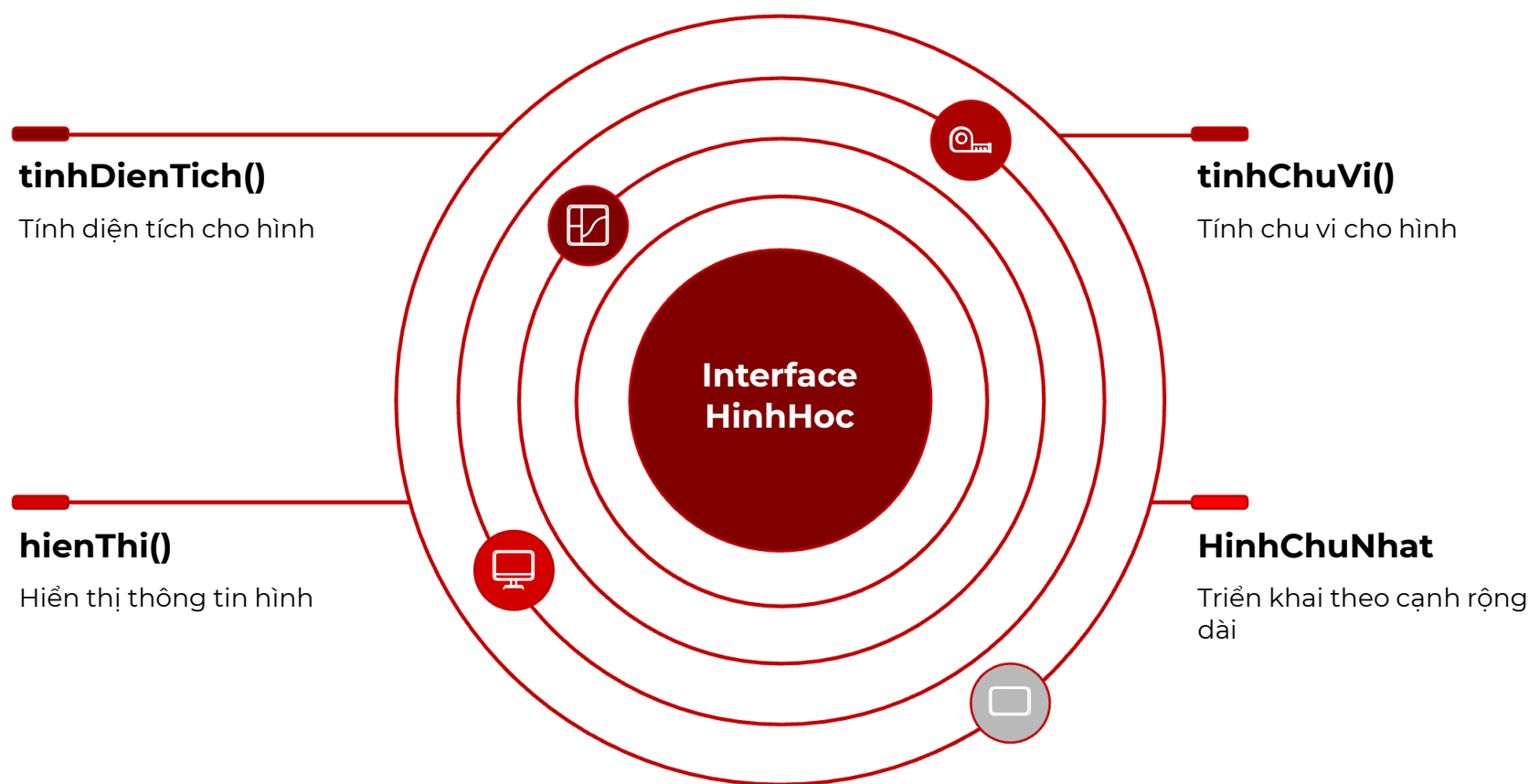
Implementation

```
class LopCuaToi implements TenInterface {  
    @Override  
    public void phuongThuc1() {  
        // Triển khai  
    }  
}
```

Điểm Quan Trọng

- Dùng từ khóa **interface**
- Phương thức mặc định là **public abstract**
- Hằng số mặc định **public static final**
- Lớp con phải implement **TẤT CẢ** phương thức

Ví Dụ: Interface Hình Học



Interface **HìnhHoc** định nghĩa hợp đồng chung cho các hình học, mỗi lớp triển khai theo cách riêng.

Khai báo Interface

```
interface HìnhHoc {
    double tinhDienTich();
    double tinhChuVi();
    void hienThi();
}
```

Ví dụ Implementation

```
class HìnhChuNhat implements HìnhHoc
{
    private double chieuDai, chieuRong;

    public double tinhDienTich() {
        return chieuDai * chieuRong;
    }
}
```

Đa Kế Thừa Với Interface

Java không hỗ trợ đa kế thừa class, nhưng cho phép implement nhiều interface - đây là ưu điểm vượt trội của Interface.

1

Định nghĩa Interface

```
interface CoTheBay {
    void bay();
}

interface CoTheBoi {
    void boi();
}
```

2

Implement Đa Kế Thừa

```
class Vit implements
    CoTheBay, CoTheBoi {

    public void bay() { }
    public void boi() { }
}
```

3

Kết Quả

Đối tượng **Vit** có cả hai khả năng bay và bơi mà không bị giới hạn kế thừa

Abstract Class Là Gì?



Abstract Class là lớp đặc biệt không thể khởi tạo trực tiếp, được thiết kế làm lớp cha với cả phương thức abstract và concrete.

Từ Khóa **abstract**

Khai báo với abstract

Có Constructor

Khởi tạo cho lớp con

Phương Thức Đa Dạng

Abstract + concrete methods

Đơn Kế Thừa

Chỉ extends một class

Khai Báo Abstract Class

Cú pháp đầy đủ

```
public abstract class TenAbstractClass {  
    protected String thuocTinh;  
  
    public TenAbstractClass(String thuocTinh) {  
        this.thuocTinh = thuocTinh;  
    }  
  
    // Abstract method  
    public abstract void phuongThucAbstract();  
  
    // Concrete method  
    public void phuongThucThuong() {  
        // Implementation có sẵn  
    }  
}
```

Lưu Ý Quan Trọng

- Lớp con **BẮT BUỘC** override abstract methods
- Có thể chứa thuộc tính với mọi access modifier
- Constructor dùng để khởi tạo thuộc tính cho lớp con
- Có thể có phương thức concrete để chia sẻ code

Ví Dụ: Hệ Thống Nhân Viên

1

Abstract Class NhanVien

```
abstract class NhanVien {  
    protected String ten;  
    protected double luongCoBan;  
  
    public abstract double tinhLuong();  
  
    public void hienThi() {  
        // Code chung  
    }  
}
```

2

NhanVienVanPhong

```
class NhanVienVanPhong  
    extends NhanVien {  
  
    public double tinhLuong() {  
        return luongCoBan + phuCap;  
    }  
}
```

3

NhanVienSanXuat

```
class NhanVienSanXuat  
    extends NhanVien {  
  
    public double tinhLuong() {  
        return luongCoBan +  
            soSanPham * donGia;  
    }  
}
```

LESSON 03

So Sánh Abstract Class và Interface

Hiểu rõ sự khác biệt là chìa khóa thiết kế hệ thống hướng đối tượng hiệu quả



Bảng So Sánh Chi Tiết

Tiêu Chí	Abstract Class	Interface
Từ khóa	abstract class	interface
Constructor	Có	Không có
Thuộc tính	Mọi loại	Chỉ hằng số
Phương thức	Abstract + Concrete	Chỉ abstract
Kế thừa	Đơn (extends)	Đa (implements)
Mục đích	IS-A relationship	CAN-DO relationship
Tốc độ	Nhanh hơn	Chậm hơn

Khi Nào Dùng Abstract Class?



Abstract Class phù hợp khi các lớp con có **mối quan hệ chặt chẽ** và cần **chia sẻ code chung**.

Sử dụng khi:

- Các lớp con có chung thuộc tính và phương thức
- Cần chia sẻ code giữa các lớp con
- Quan hệ IS-A mạnh (Dog **IS-A** Animal)
- Cần định nghĩa state với thuộc tính
- Muốn dùng access modifiers khác nhau

Ví dụ điển hình:

• **HìnhHoc** → HìnhTron, HìnhVuong

• **NhanVien** → NVVanPhong, NVSanXuat

• **DongVat** → ConCho, ConMeo

Khi Nào Dùng Interface?



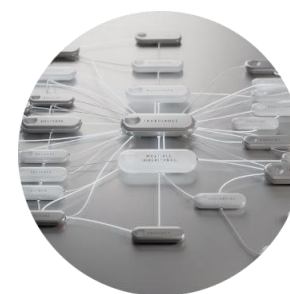
Định Nghĩa Hành Vi

Interface mô tả **khả năng** mà một lớp có thể thực hiện, không quan tâm cấu trúc bên trong



Lớp Không Liên Quan

Nhiều lớp hoàn toàn khác nhau có thể cùng chia sẻ một hành vi qua interface



Đa Kế Thừa

Một lớp cần **nhiều khả năng** từ các nguồn khác nhau



Quan Hệ CAN-DO

Chim **CAN-DO** bay, Cá **CAN-DO** bơi - mô tả khả năng hơn là bản chất

Cải Tiến Từ Java 8



Java 8 mở rộng Interface với **default** và **static methods**, làm mờ ranh giới với Abstract Class.

Default Methods

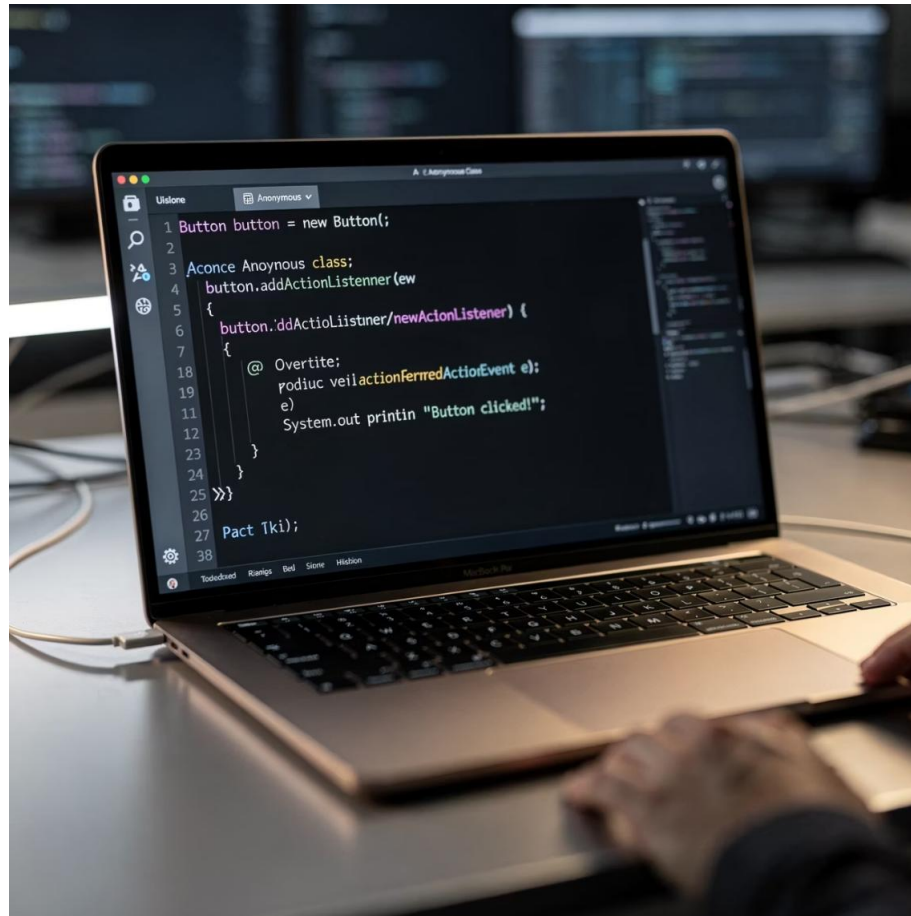
```
interface MayTinh {  
    int cong(int a, int b);  
  
    default int nhan(int a, int b) {  
        return a * b; // Có implementation  
    }  
}
```

Static Methods

```
static void thongTin() {  
    System.out.println("Máy tính v1.0");  
}
```

📌 **Lợi ích:** Thêm chức năng mới mà không phá vỡ code cũ

Anonymous Class Là Gì?



Anonymous Class (lớp nặc danh) là lớp không tên, được định nghĩa và khởi tạo đồng thời - dùng một lần cho code ngắn gọn.

Không Có Tên

Lớp nặc danh

Định Nghĩa + Khởi Tạo

Cùng lúc một chỗ

Dùng Một Lần

Không tái sử dụng

Code Ngắn

Đơn giản, súc tích

Cú Pháp Anonymous Class

1

Với Interface

```
Interface obj = new Interface()
{
    @Override
    public void phuongThuc() {
        // Implementation
    }
}; // Chú ý dấu ;
```

2

Với Abstract Class

```
AbstractClass obj = new
AbstractClass() {
    @Override
    public void
    phuongThucAbstract() {
        // Override method
    }
};
```

3

Với Class Thông Thường

```
ClassCha obj = new ClassCha() {
    @Override
    public void phuongThuc() {
        // Ghi đè
    }
};
```

📌 **Quan trọng:** Dấu chấm phẩy (;) ở cuối là bắt buộc!

Ví Dụ 1: Anonymous Class Với Interface

Interface ClickListener

```
interface ClickListener {  
    void onClick();  
    void onDoubleClick();  
}
```

Cách Truyền Thống

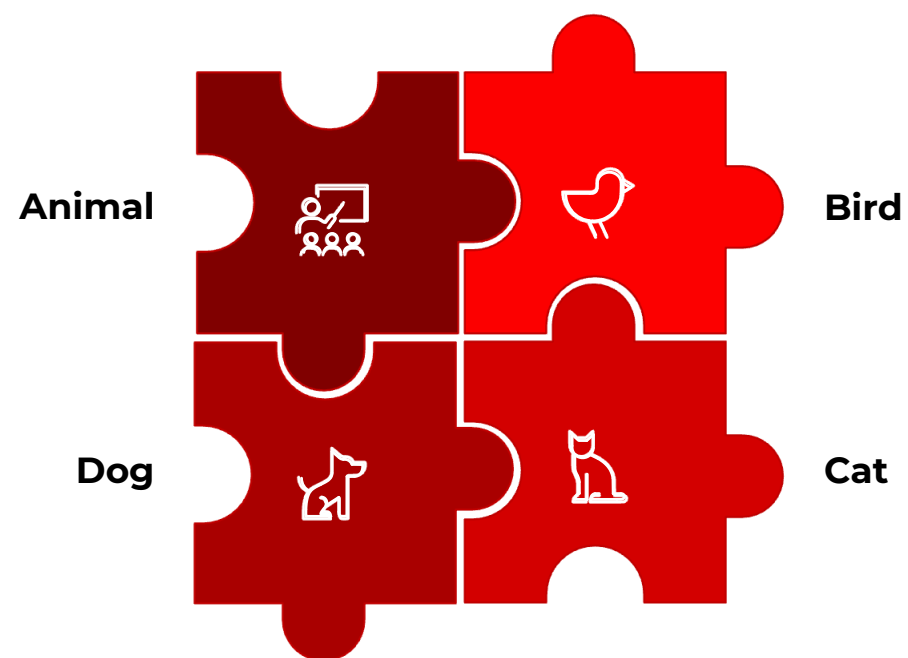
```
class MyClickListener  
    implements ClickListener {  
    public void onClick() { }  
    public void onDoubleClick() { }  
}  
  
ClickListener listener =  
    new MyClickListener();
```

Anonymous Class - Ngắn Gọn!

```
ClickListener listener =  
    new ClickListener() {  
  
        @Override  
        public void onClick() {  
            System.out.println("Click!");  
        }  
  
        @Override  
        public void onDoubleClick() {  
            System.out.println("Double!");  
        }  
    };  
  
listener.onClick();
```

Lợi ích: Không cần tạo file class riêng!

Ví Dụ 2: Anonymous Class Với Abstract Class



Abstract Class

```
abstract class Animal {  
    protected String name;  
  
    public abstract void makeSound();  
  
    public void eat() {  
        System.out.println(name + " ăn");  
    }  
}
```

Tạo Nhiều Đối Tượng

```
Animal dog = new Animal() {  
    { name = "Chó"; }  
    public void makeSound() {  
        System.out.println("Gâu gâu!");  
    }  
};
```

```
Animal cat = new Animal() {  
    { name = "Mèo"; }  
    public void makeSound() {  
        System.out.println("Meo meo!");  
    }  
};
```

Anonymous Class vs Lambda Expression

Anonymous Class (Java 7)

```
Runnable runnable = new Runnable() {  
    @Override  
    public void run() {  
        System.out.println("Chạy");  
    }  
};  
  
Collections.sort(list,  
    new Comparator() {  
        public int compare(String s1,  
                           String s2) {  
            return s1.compareTo(s2);  
        }  
    }  
);
```

Lambda Expression (Java 8+)

```
Runnable runnable = () -> {  
    System.out.println("Chạy");  
};  
  
Collections.sort(list,  
    (s1, s2) -> s1.compareTo(s2)  
);  
  
// Hoặc ngắn hơn:  
Collections.sort(list,  
    String::compareTo  
);
```

Khi nào dùng? Lambda cho functional interface đơn giản.
Anonymous Class khi cần nhiều phương thức.

TỔNG KẾT

Tổng Kết Session 10

1

Interface

- Định nghĩa hành vi
- Đa kế thừa
- Quan hệ CAN-DO
- Java 8: default/static methods

2

Abstract Class

- Lớp trừu tượng
- Có constructor
- Đơn kế thừa
- Quan hệ IS-A

3

So Sánh

- Abstract: IS-A, chia sẻ code
- Interface: CAN-DO, linh hoạt
- Tốc độ vs Flexibility

4

Anonymous Class

- Lớp không tên
- Dùng một lần
- Code ngắn gọn
- Thay bằng Lambda (Java 8+)

Lợi Ích Của Tính Trừu Tượng



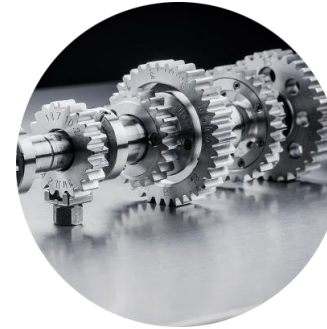
Che Giấu Chi Tiết

Ẩn implementation phức tạp, chỉ hiển thị những gì cần thiết



Tăng Tính Linh Hoạt

Thay đổi implementation mà không ảnh hưởng code sử dụng



Dễ Bảo Trì

Code rõ ràng, có cấu trúc, giảm bug và tăng chất lượng



Dễ Mở Rộng

Thêm tính năng mới không sửa code cũ - Open/Closed Principle



Hỗ Trợ Đa Hình

Một interface tham chiếu nhiều đối tượng khác nhau



Teamwork Hiệu Quả

Định nghĩa rõ contract, nhiều người làm song song

"Abstraction is not about hiding complexity, it's about managing complexity by providing a simpler interface."

- | Đã nắm vững 4 tính chất cốt lõi của OOP
- | Có thể thiết kế và triển khai hệ thống có sử dụng kế thừa
- | Biết cách sử dụng đa hình để code linh hoạt
- | Hiểu rõ khi nào dùng Interface vs Abstract Class
- | Có thể áp dụng vào project thực tế



KẾT THÚC

HỌC VIỆN ĐÀO TẠO LẬP TRÌNH CHẤT LƯỢNG NHẬT BẢN