I. Tính đóng gói (Encapsulation) trong Java

Tính đóng gói là một trong **bốn tính chất cơ bản của lập trình hướng đối tượng (OOP)**. Nó giúp **ẩn đi các chi tiết bên trong của một đối tượng**, và chỉ cho phép người dùng **tương tác thông qua các phương thức công khai (public methods)**.

Bạn có thể hình dung tính đóng gói giống như **một chiếc hộp kín**:

- Bạn **không thể nhìn thấy hoặc tác động trực tiếp vào bên trong** chiếc hộp.
- Thay vào đó, bạn **sử dụng các nút bấm (tức là các phương thức)** để điều khiển chức năng bên trong hộp.

Cụ thể, tính đóng gói trong Java được thực hiện qua 3 điểm chính:

1. Ẩn dữ liệu (Data Hiding)

- Các **thuộc tính (biến)** của một lớp thường được khai báo là `private`.
- Điều này nghĩa là: chỉ **các phương thức bên trong lớp đó mới có quyền truy cập** đến các thuộc tính này.
- Việc này giúp **ngăn chặn truy cập hoặc thay đổi trái phép từ bên ngoài**.

2. Cung cấp các phương thức truy cập (Accessors - Getters và Mutators - Setters)

- Để cho phép các đối tượng bên ngoài **tương tác với dữ liệu**, bạn cung cấp **các phương thức `public`**:
    - **Getter**: dùng để **lấy giá trị** của thuộc tính.
    - **Setter**: dùng để **gán hoặc thay đổi giá trị** của thuộc tính.
- Nhờ đó, bạn có thể **cho phép truy cập có kiểm soát**, thay vì mở hoàn toàn.

3. Bảo vệ tính toàn vẹn của dữ liệu

- Thông qua việc **kiểm soát truy cập bằng getter và setter**, bạn có thể:
    - **Giới hạn các giá trị hợp lệ** được gán cho thuộc tính.
    - **Ngăn các hành động sai lệch hoặc bất hợp pháp** từ bên ngoài đối tượng.
- Nhờ đó, bạn đảm bảo rằng **dữ liệu bên trong đối tượng luôn hợp lệ**, ổn định và đúng như thiết kế.

II. Kế thừa (Inheritance) trong Java

Kế thừa là một tính năng của **lập trình hướng đối tượng**, cho phép **một lớp (class) mới tiếp nhận các thuộc tính và phương thức** từ một lớp khác.

- Lớp mới được gọi là **lớp con (subclass)**.
- Lớp mà nó kế thừa được gọi là **lớp cha (superclass)**.

Kế thừa giúp:

- **Tái sử dụng mã nguồn**
- **Giảm lặp lại code**
- **Tăng tính mô-đun** và khả năng mở rộng của chương trình

Các thuật ngữ quan trọng trong kế thừa Java:

- **Lớp cha (Superclass):** Lớp có các thuộc tính và phương thức được lớp con kế thừa.
- **Lớp con (Subclass):** Lớp kế thừa các thuộc tính và phương thức từ lớp cha.
- **`extends`:** Từ khóa dùng để chỉ rõ rằng một lớp kế thừa từ lớp khác.
- **`super`:** Từ khóa dùng để truy cập các thuộc tính và phương thức của lớp cha từ lớp con.

Khi nào nên sử dụng kế thừa trong Java?

Bạn nên sử dụng kế thừa trong các trường hợp sau:

- Khi có **nhiều lớp có các thuộc tính và phương thức chung** → tạo một **lớp cha chung** để các lớp con kế thừa, giúp **tránh lặp mã**.
- Khi cần **mở rộng chức năng** của một lớp hiện có → tạo lớp con và **thêm thuộc tính hoặc phương thức mới**.
- Khi muốn **ghi đè phương thức của lớp cha** để thực hiện hành vi riêng trong lớp con (dùng `@Override`).
- Khi muốn tạo **nhiều lớp con có điểm chung** bằng cách kế thừa từ cùng một lớp cha — giúp tổ chức mã tốt hơn và tái sử dụng hiệu quả.

```java
public class Person {
    private String name, birth;
    public Person(String name, String birth){
        this.name = name;
        this.birth = birth;
    }
    public String getName(){
         return this.name
    }
}

public class Student extends Person{
    private String lop;
    private double gpa;
    public Student(String lop, double gpa, String name,
                                            String birth){
        super(name, birth);
        this.lop = lop;
        this.gpa = gpa;
    }
}

public class Main {
    public static void main(String[] args) {
        Student s = new Student("CNTT1", 3.2, "Nguyen Van A", "22/12/2002");
        System.out.println(s.getName());
    }
}
```

OUTPUT

Nguyen Van A

III. Đa Hình

- Đa hình cho phép bạn tham chiếu một biến thuộc kiểu dữ liệu của lớp cơ sở tới đối tượng của một lớp con
- ví dụ: Lớp Student kế thừa từ lớp Person thì tất cả các thực thể của lớp Student đều là thực thể của lớp Person nhưng ngược lại thì không

```java
public class Person {
    public void display(){
        System.out.println("Person !");
    }
}

public class Student extends Person{
    public void display(){
        System.out.println("Student !");
    }
}

public class Staff extends Person {
    public void display(){
        System.out.println("Staff !");
    }
}
```

```java
public class Lecturer extends Person {
    public void display(){
        System.out.println("Lecturer !");
    }
}

public class Main {
    public static void main(String[] args) {
        Person p1 = new Student();
        Person p2 = new Staff();
        Person p3 = new Lecturer();
        p1.display();
        p2.display();
        p3.display();
    }
}
```

OUTPUT

```
Student !
Staff !
Lecturer !
```

- Ép kiểu đối tượng và toán tử instanceof
  - + Một biến tham chiếu của đối tượng có thể được ép sang tham chiếu của một đối tượng thuộc lớp khác, đây gọi là ép kiểu đối tượng

```java
public class Main {
    public static void main(String[] args) {
        Object ob = new Student(); //Implicit casting
        Student s = (Student)ob; // Explicit casting
    }
}
```

+ Ta có thể ép một thực thể của lớp con sang một biến đối tượng của lớp cha, vì một đối tượng của lớp con bao giờ cũng là một đối tượng của lớp cha

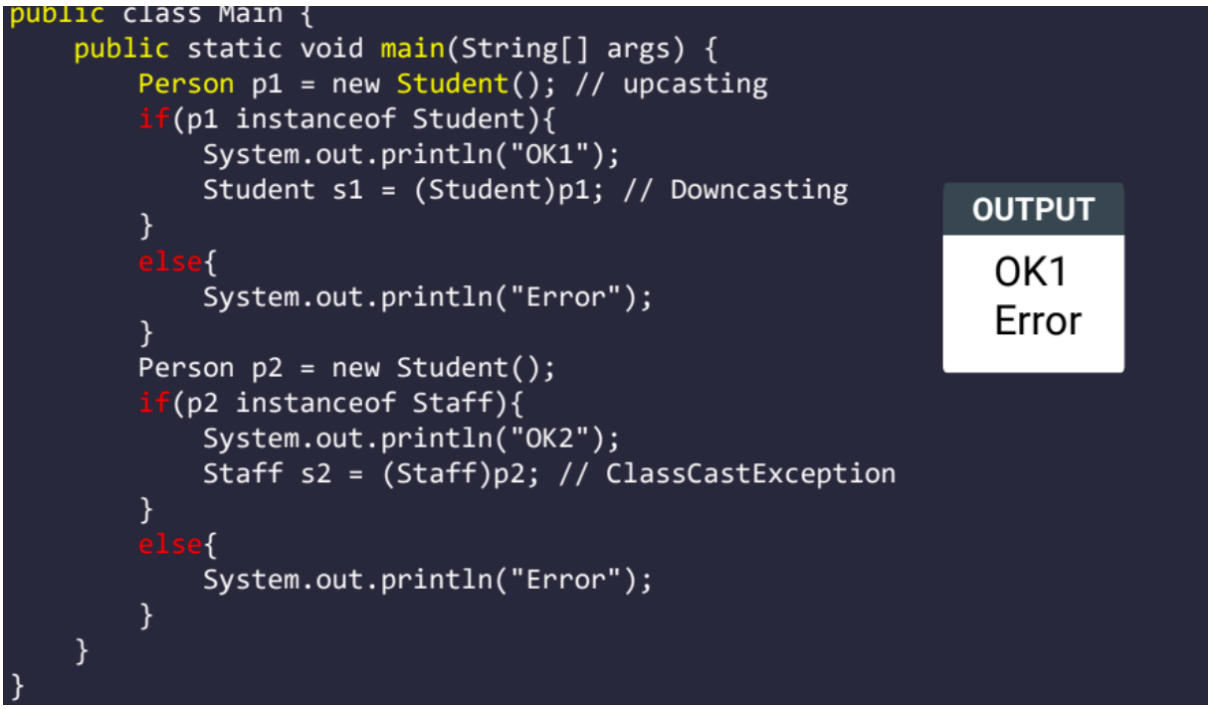
  + Khi ép một thực thể của lớp cha sang biến đối tượng của lớp con, phải đảm bảo rằng thực thể của lớp cha là một thực thể của lớp con, nếu không sẽ phát sinh lỗi

```java
public class Main {
    public static void main(String[] args) {
        Person p1 = new Student(); // upcasting
        Student s1 = (Student)p1; // Downcasting
        Person p2 = new Student();
        Staff s2 = (Staff)p2; // ClassCastException
    }
}
```

OUTPUT

```
Exception in thread "main"
java.lang.ClassCastException:
Polymophism.Student cannot be
cast to Polymophism.Staff
```

- Sử dụng toán tử instanceof



IV. Tính trừu tượng

- Trong lập trình hướng đối tượng, tính trừu tượng là một khái niệm quan trọng giúp mô hình hóa thế giới thực bằng cách ẩn đi các chi tiết triển khai bên trong của một đối tượng, chỉ tập trung vào hành vi và giao diện bên ngoài của đối tượng đó. Tính trừu tượng cho phép chúng ta xác định các lớp trừu tượng, các phương thức trừu tượng và các biến trừu tượng trong mã của chúng ta.
- Các phương thức trừu tượng là các phương thức không có phần thân, chỉ có phần khai báo, các phương thức trừu tượng phải được ghi đè trong các lớp con. Biến trừu tượng là các biến chưa được gán giá trị ban đầu và giá trị của chúng phải được gán trong các lớp con.
- Tạo ra các lớp trừu tượng: Như đã đề cập ở trên, một lớp trừu tượng là một lớp không thể khởi tạo và chỉ có thể được sử dụng để tạo các lớp con. Điều này có nghĩa là chúng ta không thể tạo một đối tượng từ một lớp trừu tượng bằng cách sử dụng từ khóa `new`. Thay vào đó, chúng ta phải tạo một lớp con và triển khai các phương thức trừu tượng của lớp cha.
- Phương thức trừu tượng: Phương thức trừu tượng là các phương thức không có phần thân, chỉ có phần khai báo. Điều này có nghĩa là chúng ta chỉ định tên, kiểu dữ liệu và tham số cho phương thức mà không cần triển khai nội dung của nó. Các phương thức trừu tượng phải được ghi đè trong các lớp con và triển khai nội dung của chúng.

- Lớp con: Một lớp con là một lớp kế thừa từ một lớp trừu tượng. Điều này có nghĩa là lớp con sẽ kế thừa các thuộc tính và phương thức của lớp cha, bao gồm cả các phương thức trừu tượng. Tuy nhiên, lớp con phải cung cấp triển khai cho tất cả các phương thức trừu tượng của lớp cha.

1.Interface là gì

a) Khái niệm

- Interface là một hợp đồng định nghĩa các phương thức (method) mà một lớp (class) phải thực hiện, nhưng không cung cấp phần triển khai (implementation).
- Mục đích: Interface giúp tạo ra một cơ chế để các lớp khác nhau tuân thủ cùng một cấu trúc, dễ mở rộng và duy trì mã nguồn.
- Ví dụ: Trong Java interface thường chứa các phương thức abstract. Một lớp thực hiện (implement) interface phải định nghĩa tất cả các phương thức trong đó.

```
public interface Animal {
void makeSound(); // Chỉ định nghĩa, không có triển khai
}

public class Dog implements Animal {
public void makeSound() {
System.out.println("Woof!");
}
}
```

b) Các đặc điểm của interface trong lập trình

- Không có thân phương thức: Một trong những đặc điểm nổi bật của Interface là nó không chứa phần thân của phương thức. Interface chỉ định nghĩa chữ ký của phương thức mà không thực hiện hành động cụ thể. Điều này có nghĩa là các lớp con phải cung cấp cài đặt cho các phương thức này. Điều này giúp đảm bảo tính linh hoạt trong việc triển khai và tạo ra sự trừu tượng trong thiết kế phần mềm.
- Không thể tạo đối tượng từ Interface: Interface không thể được khởi tạo như một đối tượng thông thường. Khi một lớp triển khai một interface, nó phải cung cấp phần thân của các phương thức trong interface đó. Lớp con sẽ là nơi thực thi cụ thể các phương thức của interface. Do đó, một interface chỉ đóng vai trò là bản thiết kế, không phải là đối tượng có thể khởi tạo.

```
// Không thể khởi tạo trực tiếp từ interface
// Animal a = new Animal(); // Lỗi
```

- Một lớp có thể triển khai nhiều interface: Khác với kế thừa (inheritance) trong lập trình hướng đối tượng, nơi một lớp chỉ có thể kế thừa từ một lớp cha duy nhất, các ngôn ngữ như Java cho phép một lớp triển khai nhiều interface. Điều này giúp tạo ra sự linh hoạt cao trong việc thiết kế hệ thống, vì một lớp có thể kết hợp nhiều tính năng từ các interface khác nhau mà không bị ràng buộc bởi chỉ một lớp cha duy nhất.

```
public interface Animal {
void makeSound();
}

public interface Domestic {
void playWithOwner();
}

public class Dog implements Animal, Domestic {
@Override
public void makeSound() {
System.out.println("Bark");
}
@Override
public void playWithOwner() {
System.out.println("Playing with owner");
}
}
```

- Tính kế thừa trong interface: Mặc dù interface không có phần thân phương thức, nhưng nó hỗ trợ tính kế thừa giữa các interface. Một interface có thể kế thừa từ một hoặc nhiều interface khác, thừa hưởng các phương thức mà các interface cha đã định nghĩa. Điều này giúp tối ưu hóa mã nguồn và tái sử dụng các định nghĩa interface.

2. Abstract class

- Abstract Class có nghĩa là một lớp trừu tượng. Abstract Class là một loại lớp trong OOP – Lập trình hướng đối tượng. Abstract Class khai báo một hay nhiều method trừu tượng. Chúng có thể có cả method trừu tượng và method cụ thể. Một lớp bình thường không thể có method trừu tượng. Điều này có nghĩa là một abstract class phải chứa ít nhất một method trừu tượng.

- Một method trừu tượng là một method được khai báo không có triển khai (không có dấu ngoặc nhọn và kết thúc là một dấu chấm phẩy).
- Một Abstract Class không thể tạo đối tượng nhưng có thể bao gồm lớp con. Khi một lớp trừu tượng bao gồm các lớp con, lớp con thường cung cấp triển khai cho tất cả các method trừu tượng của lớp cha. Ngoài ra, lớp trừu tượng có thể có trường động và trường tĩnh. Tuy nhiên, nếu lớp con không triển khai cho lớp cha, nó cũng sẽ là một lớp trừu tượng.
- chú ý:

Không thể tạo đối tượng trực tiếp từ abstract class.

→ chỉ có thể dùng nó làm lớp cha.

Có thể chứa 2 loại phương thức:

- abstract method (chỉ khai báo, không có code) → lớp con **bắt buộc** override.
- normal method (có code sẵn) → lớp con **không bắt buộc** override, nhưng có thể ghi đè nếu muốn.

- **Có thể có biến (field), constructor** và cả phương thức static.-
- **Khi một lớp kế thừa abstract class**:
  - + Nếu override hết các phương thức abstract → lớp đó trở thành **lớp thường**.
  - + Nếu **không override hết** → lớp đó cũng phải khai báo là abstract.
- **Dùng khi**: muốn tạo ra một khuôn mẫu (template) có sẵn một phần code chung, nhưng bắt buộc lớp con phải tự cài đặt một số hành vi cụ thể.

```
public interface Animal {
void makeSound();
}
public interface Mammal extends Animal {
void feedMilk();
}
```

- Trong ví dụ trên, `Mammal` kế thừa từ `Animal`, do đó bất kỳ lớp nào triển khai `Mammal` cũng sẽ phải triển khai phương thức `makeSound()` từ interface `Animal`, ngoài việc triển khai phương thức `feedMilk()` từ `Mammal`.
- Interface hỗ trợ đa hình: Interface giúp hỗ trợ đa hình trong lập trình. Khi một lớp triển khai một interface, bạn có thể sử dụng một đối tượng của lớp đó thông qua kiểu interface mà nó triển khai, thay vì sử dụng trực tiếp lớp đó. Điều này giúp mã nguồn dễ bảo trì và linh hoạt hơn.

```
public class Dog implements Animal {
@Override
public void makeSound() {
System.out.println("Bark");
}
}
public class Main {
public static void main(String[] args) {
Animal myAnimal = new Dog();
myAnimal.makeSound(); // Sẽ in ra "Bark"
}
}
```

- Các phương thức mặc định và static trong interface: kể từ phiên bản Java 8, interface có thể chứa các phương thức mặc định (`default methods`) và phương thức tĩnh (`static methods`). Điều này giúp mở rộng khả năng của interface mà không làm gián đoạn các lớp hiện có đã triển khai interface trước đó
    + Phương thức mặc định: Cho phép cung cấp một cài đặt mặc định cho phương thức trong interface mà không yêu cầu các lớp con phải cài đặt lại.

        ```
        public interface Animal {
        default void sleep() {
        System.out.println("Animal is sleeping");
        }
        }
        ```

    + Phương thức tĩnh: Là phương thức được khai báo với từ khóa `static` trong interface, có thể được gọi thông qua tên của interface.