KHÁI NIỆM OOP

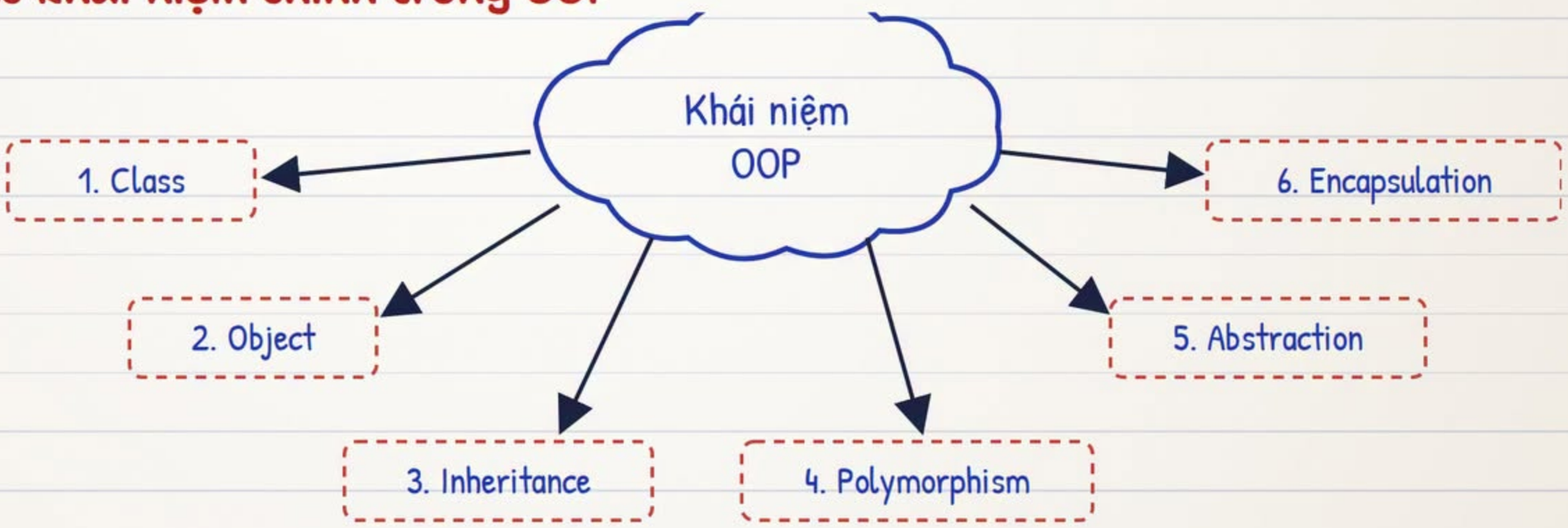
★ OOP là gì?

OOPS (Object-Oriented Programming System) là mô hình lập trình dựa trên khái niệm "Object" (đối tượng) – thứ có thể chứa dữ liệu dưới dạng các field (thuộc tính) và code dưới dạng các procedure (method).

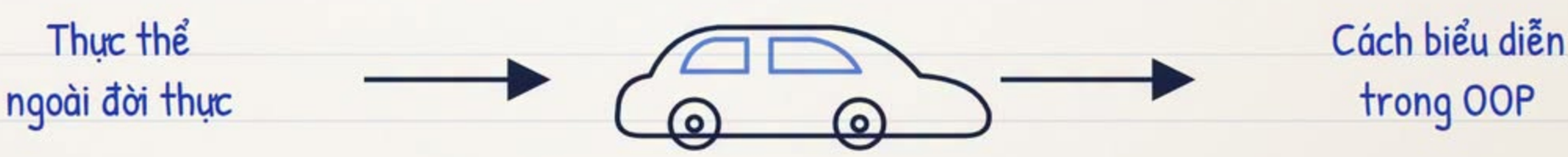
★ Vì sao dùng OOP?

- Giúp chương trình dễ hiểu, dễ bảo trì và dễ chỉnh sửa hơn.
- Bảo vệ dữ liệu nhờ cơ chế che giấu dữ liệu (data hiding).
- Hỗ trợ tái sử dụng code thông qua inheritance (kế thừa).
- Mang lại sự linh hoạt nhờ polymorphism (đa hình).
- Giúp mô hình hóa các thực thể ngoài đời thực.

★ Các khái niệm chính trong OOP



★ OOP trong đời thực



★ Tóm tắt

OOP là mô hình lập trình mạnh mẽ, dùng object và class để thiết kế ứng dụng có tính module, dễ tái sử dụng và dễ mở rộng. Các khái niệm phía trên phối hợp với nhau để giúp code hiệu quả và dễ quản lý hơn.

Ghi nhớ:

Nghĩ theo Class,
Tạo ra Object,
Xây nên giải pháp! ☆

☆ CẨM NANG OOP TOÀN TẬP ☆

1. CLASS

★ Class là gì?

Class là bản thiết kế hay khuôn mẫu để tạo ra các object. Nó định nghĩa tập hợp các attribute (dữ liệu) và method (hàm) mà các object của class đó sẽ có.

★ Điểm cốt lõi

- Class không chiếm bộ nhớ.
- Class là kiểu dữ liệu do người dùng định nghĩa.
- Class đóng vai trò bản thiết kế để tạo ra object.
- Class giúp che giấu dữ liệu và tái sử dụng code.

★ Cú pháp (trong Java)

```
class ClassName {
    // Thuộc tính
    dataType attribute1;
    dataType attribute2;
    ...
    // Method
    returnType methodName(parameters) {
        // thân method
    }
}
```

→ Từ khóa class

→ Attribute / Biến (thành phần dữ liệu)

→ Method / Hàm (phương thức thành viên)

★ Ví dụ

```
class Student {
    // Thuộc tính
    int id;
    String name;
    double marks;
    // Method
    void display() {
        System.out.println("ID: " + id);
        System.out.println("Name: " + name);
        System.out.println("Marks: " + marks);
    }
}
```

Cấu trúc một Class

Student (Tên class)
Attribute (Dữ liệu) int id; String name; double marks;
Method (Hàm) void display();

→ Chứa dữ liệu (trạng thái)

→ Thực hiện hành động (hành vi)

Ghi nhớ:

Class chỉ là một khuôn mẫu. Không có bộ nhớ nào được cấp phát khi class được tạo. Bộ nhớ chỉ được cấp phát khi object được tạo ra từ class.

Ví dụ:

```
Student s1 = new Student();
// Tạo object
```

☆ ≒ CẨM NANG OOP TOÀN TẬP ≒ ☆

≒ 2. OBJECT ≒

★ Object là gì?

Object là một thực thể ngoài đời thực, chính là một instance (thể hiện) của class. Nó có state (dữ liệu) và behavior (method).

★ Điểm cốt lõi

- Object được tạo ra từ một class.
- Object chiếm bộ nhớ.
- Mỗi object có bản sao dữ liệu của riêng nó.
- Các object tương tác với nhau bằng cách gọi method.
- Có thể tạo nhiều object từ cùng một class.

★ Cú pháp (trong Java)

```
ClassName objectName = new ClassName();
```

⟶ Tên class
⟶ Tên object (tham chiếu)
⟶ Từ khóa new
⟶ Constructor

★ Ví dụ

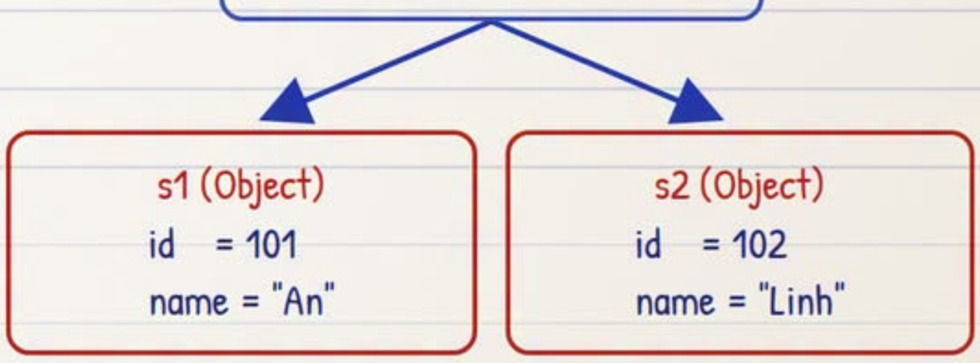
```
class Student {
    int id;
    String name;

    void display() {
        System.out.println(id + " - " + name);
    }
}
public class Main {
    public static void main(String[] args) {
        // Tạo các object
        Student s1 = new Student();
        Student s2 = new Student();

        // Gán giá trị
        s1.id = 101;
        s1.name = "An";
        s2.id = 102;
        s2.name = "Linh";

        // Gọi method
        s1.display();  // Kết quả: 101 - An
        s2.display();  // Kết quả: 102 - Linh
    }
}
```

Sơ đồ Object



Ở đây, s1 và s2 là các object (instance) của class Student. Mỗi object có dữ liệu riêng của mình.

Ghi nhớ:

- Class là bản thiết kế.
- Object là thực thể thật được tạo ra từ class.
- Có thể tạo nhiều object từ một class.

Ví dụ ngoài đời thực:

Class : Điện thoại

Object : iPhone, Samsung, OnePlus (mỗi máy có thông số và hành vi riêng)

☆ ≒ CẨM NANG OOP TOÀN TẬP ≒ ☆

≒ 3. METHOD ≒

★ Method là gì?

Method là một khối code thực hiện một nhiệm vụ cụ thể. Nó được định nghĩa bên trong class và dùng để mô tả hành vi của object.

★ Điểm cốt lõi

- Method là một hàm thuộc về một class.
- Method dùng để thực hiện một thao tác nào đó.
- Method có thể truy cập và thay đổi dữ liệu (attribute) của class.
- Method giúp code có tính module và dễ tái sử dụng.
- Method có thể trả về giá trị hoặc không trả về gì (void).

★ Cú pháp (trong Java)

```
returnType methodName(parameters) {
    // thân method
    // các câu lệnh
    return value;   // không bắt buộc

}
```

→ Kiểu trả về của method (int, void, String, ...)

→ Tên method

→ Tham số truyền vào method (không bắt buộc)

→ Thân method

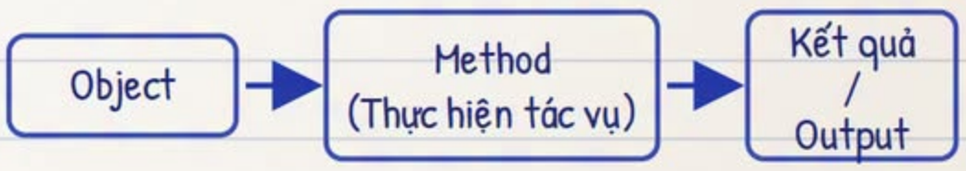
★ Ví dụ

```
class Calculator {
    int add(int a, int b) {
        int sum = a + b;
        return sum;
    }
    void showMessage() {
        System.out.println("Hello, Java!");
    }
}
public class Main {
    public static void main(String[] args) {
        Calculator c = new Calculator();

        int result = c.add(10, 20);
        // gọi method có trả về
        System.out.println("Sum = " + result);

        c.showMessage();
        // gọi method kiểu void
    }
}
```

Method hoạt động thế nào?



Object gọi method.

Method thực hiện tác vụ.

Method trả về kết quả (nếu có).

Ghi nhớ:

- Method chính là hành vi của object.
- Method luôn được định nghĩa bên trong class.
- Method có thể trả về giá trị hoặc không.
- Tên method tốt phải mô tả được việc nó làm.

Ví dụ ngoài đời thực:

Hãy nghĩ đến chiếc điều khiển TV.
Nhấn một nút (method) sẽ tạo ra một hành động như đổi kênh, tăng âm lượng...
Mỗi nút có một chức năng riêng.

☆ CẨM NANG OOP TOÀN TẬP ☆

4. CONSTRUCTOR

★ Constructor là gì?

Constructor là một method đặc biệt trong class, dùng để khởi tạo object. Nó được gọi tự động khi một object được tạo ra.

★ Điểm cốt lõi

- Tên constructor trùng với tên class.
- Constructor không có kiểu trả về, kể cả void.
- Được gọi tự động khi object được tạo.
- Dùng để khởi tạo dữ liệu cho object.
- Một class có thể có nhiều constructor (constructor overloading).
- Nếu không viết constructor nào, Java sẽ tự cấp một default constructor.

★ Cú pháp (trong Java)

```
class ClassName {
    ClassName(parameters) {
        // code khởi tạo
    }
}
```

→ Tên constructor
→ Tham số (không bắt buộc)
→ Thân constructor

★ Ví dụ

```
class Student {
    int id;
    String name;
    // Constructor
    Student(int id, String name) {
        this.id = id;
        this.name = name;
    }
    void display() {
        System.out.println(id + " - " + name);
    }
}
public class Main {
    public static void main(String[] args) {
        Student s1 = new Student(101, "An");
        Student s2 = new Student(102, "Linh");
        s1.display();  // Kết quả: 101 - An
        s2.display();  // Kết quả: 102 - Linh
    }
}
```

Constructor hoạt động thế nào?

new Student(101, "An")

↓

Constructor được gọi

↓

Object được tạo ra

↓

Dữ liệu được khởi tạo

Các loại Constructor

- Default Constructor (không tham số)
- Parameterized Constructor (có tham số)
- Copy Constructor (qua tham số)

Ghi nhớ:

- Constructor chỉ chạy một lần duy nhất khi object được tạo.
- Constructor giúp khởi tạo dữ liệu của object ngay lúc tạo.

Ví dụ ngoài đời thực:

Khi bạn đăng ký tài khoản trên một website, thông tin của bạn (tên, email, mật khẩu) được điền vào form ngay lúc tài khoản được tạo.

☆ CẨM NANG OOP TOÀN TẬP ☆

5. INHERITANCE

★ Inheritance là gì?

Inheritance (kế thừa) là cơ chế cho phép một class mới (child / subclass) thừa hưởng các thuộc tính và hành vi (field và method) của một class đã có (parent / superclass). Nó thúc đẩy việc tái sử dụng code và thiết lập quan hệ IS-A.

★ Điểm cốt lõi

- Class có các thành phần được kế thừa gọi là superclass (class cha).
- Class đi kế thừa gọi là subclass (class con).
- Subclass có thể có thành phần riêng bên cạnh những gì được kế thừa.
- Inheritance hỗ trợ tái sử dụng code và khả năng mở rộng.
- Java không hỗ trợ đa kế thừa với class (nhưng hỗ trợ thông qua interface).

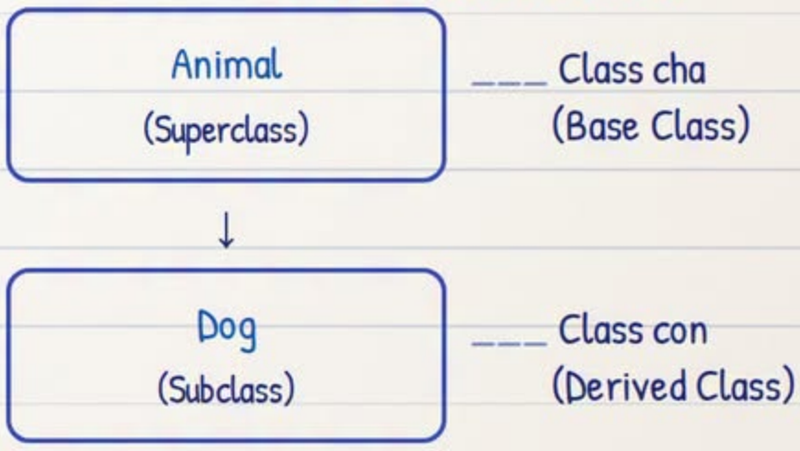
★ Cú pháp (trong Java)

```
class SuperClass {                          ⟶ Class cha / Superclass
    // field và method
}
class SubClass extends SuperClass {         ⟶ Class con / Subclass
    // field và method riêng
}
```

★ Ví dụ (trong Java)

```
// Class cha (Superclass)
class Animal {
    String color = "Brown";
    void eat() {
        System.out.println("Animal is eating");
    }
}
// Class con (Subclass)
class Dog extends Animal {
    void bark() {
        System.out.println("Dog is barking");
    }
}
// Class Main
public class Main {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.eat();  // method kế thừa
        d.bark();  // method riêng
        System.out.println(d.color); // field kế thừa
    }
}
```

Sơ đồ kế thừa



Cơ chế hoạt động

① Dog là một Animal (quan hệ IS-A).
② Dog kế thừa color và eat() từ Animal.
③ Dog dùng được thành phần kế thừa và định nghĩa thêm thành phần riêng.

Lợi ích

- Tái sử dụng code
- Dễ bảo trì
- Dễ mở rộng
- Tổ chức code tốt hơn

Ghi nhớ

✓ Dùng từ khóa extends để kế thừa.
✓ Thành phần private không được kế thừa.
✓ Constructor không được kế thừa.
✓ Inheritance thể hiện quan hệ IS-A.

6. POLYMORPHISM

★ Polymorphism là gì?

Polymorphism nghĩa là "nhiều hình thái". Nó cho phép dùng một interface cho nhiều kiểu khác nhau.

Trong Java, polymorphism đạt được nhờ Method Overloading và Method Overriding.

★ Các loại Polymorphism

1. Compile-time Polymorphism (Method Overloading)
2. Run-time Polymorphism (Method Overriding)

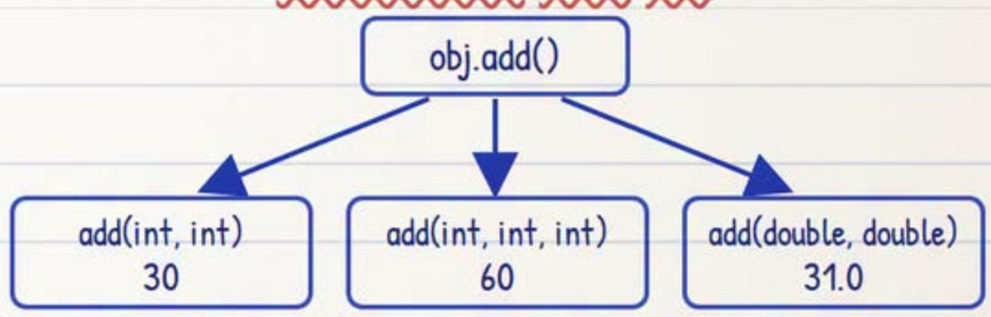
1. Compile-time Polymorphism (Method Overloading)

- Xảy ra trong cùng một class.
- Cùng tên method nhưng khác danh sách tham số (số lượng, kiểu hoặc thứ tự).
- Được xác định ngay lúc biên dịch.

Ví dụ (trong Java)

```
class Calculator {
    int add(int a, int b) {
        return a + b;
    }
    int add(int a, int b, int c) {
        return a + b + c;
    }
    double add(double a, double b) {
        return a + b;
    }
}
public class Main {
    public static void main(String[] args) {
        Calculator obj = new Calculator();
        System.out.println(obj.add(10, 20));     // 30
        System.out.println(obj.add(10, 20, 30)); // 60
        System.out.println(obj.add(10.5, 20.5)); // 31.0
    }
}
```

Cơ chế hoạt động



Trình biên dịch quyết định gọi method nào dựa trên các đối số được truyền vào.

Ghi nhớ

- Overloading giúp code dễ đọc hơn.
- Overriding tạo ra đa hình lúc runtime.
- Dùng @Override là thói quen tốt.

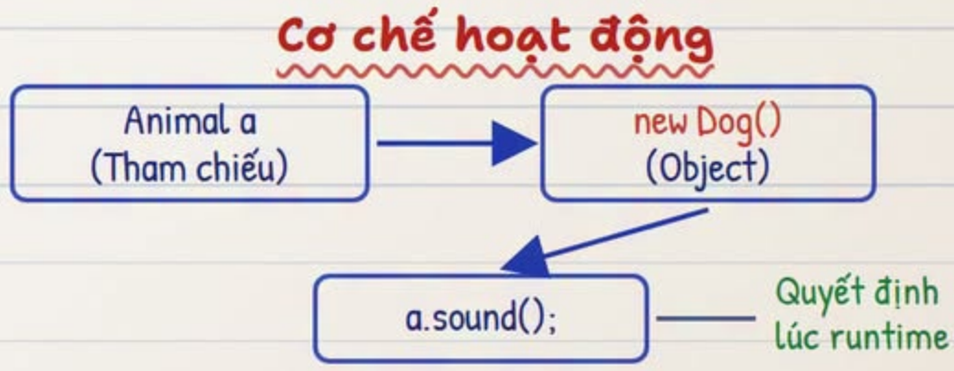
2. Run-time Polymorphism (Method Overriding)

- Xảy ra ở các class khác nhau thông qua inheritance.
- Cùng tên method, cùng tham số.
- Được xác định lúc chạy chương trình (runtime).

Ví dụ (trong Java)

```
class Animal {
    void sound() {
        System.out.println("Animal makes sound");
    }
}
class Dog extends Animal {
    @Override
    void sound() {
        System.out.println("Dog barks");
    }
}
public class Main {
    public static void main(String[] args) {
        Animal a = new Dog(); // cha tham chiếu, con là object
        a.sound();            // gọi method đã override
    }
}
```

Cơ chế hoạt động



JVM quyết định gọi method nào lúc runtime dựa trên object thực tế.

Khác biệt chính

Tiêu chí	Overloading	Overriding
Thời điểm	Compile-time	Run-time
Nơi xảy ra	Cùng 1 class	Khác class
Tham số	Phải khác nhau	Phải giống nhau

☆ CẨM NANG OOP TOÀN TẬP ☆

8. ENCAPSULATION

★ Encapsulation là gì?

Encapsulation (đóng gói) là cơ chế gói dữ liệu (biến) và code (method) lại thành một khối thống nhất, đồng thời hạn chế truy cập trực tiếp vào một số thành phần của object. Nó giúp che giấu dữ liệu và bảo vệ dữ liệu khỏi những truy cập trái phép.

★ Điểm cốt lõi

- Ta đạt được encapsulation bằng cách dùng access modifier.
- Các biến của class nên để private.
- Việc truy cập biến được thực hiện qua các method getter và setter public.
- Nó bảo vệ dữ liệu khỏi bị thay đổi ngoài ý muốn.
- Nó giúp code dễ bảo trì và linh hoạt hơn.

★ Cú pháp (trong Java)

```
class ClassName {
    private dataType variable;
    // biến private

    // method getter
    public dataType getVariable() {
        return variable;
    }

    // method setter
    public void setVariable(dataType value) {
        variable = value;
    }
}
```

★ Ví dụ (trong Java)

```
class Student {
    private int id;
    private String name;

    // Getter cho id
    public int getId() {
        return id;
    }
    // Setter cho id
    public void setId(int id) {
        this.id = id;
    }
    // Getter cho name
    public String getName() {
        return name;
    }
    // Setter cho name
    public void setName(String name) {
        this.name = name;
    }
}
```

★ Cơ chế hoạt động

Client Code

```
obj.setId(101);
int id = obj.getId();
```

→

Student Class

Private Data
- id
- name

Public Methods
+ setId() + getId()
+ setName() + getName()

Dữ liệu bên trong được khóa lại, thế giới bên ngoài chỉ chạm tới được thông qua các method public.

Lợi ích

- ☑ Bảo vệ dữ liệu khỏi truy cập trái phép.
- ☑ Tăng tính bảo mật.
- ☑ Cho phép kiểm soát cách truy cập dữ liệu.
- ☑ Giúp code dễ bảo trì và chỉnh sửa.

Ghi nhớ

- ☑ Để các biến ở chế độ private.
- ☑ Cung cấp method getter và setter public.
- ☑ Chỉ truy cập và thay đổi dữ liệu thông qua method.