

OOP CƠ BẢN








Bộ phận Đào tạo

2019/09

【Bí Mật】 Object & Class – Khái niệm








- Object (Đối tượng) là tất cả mọi thứ hiện hữu trong thực tế.
- Class (Lớp) có thể xem là mẫu thiết kế của một hoặc nhiều đối tượng, được sử dụng để mô tả các đặc điểm của đối tượng bao gồm các Thuộc tính (Property), Phương thức (Method), Trạng thái (State)...



Vehicle	
	name: string
	type: string
	price: double
	...
	run(): void
	stop(): void
	...

[Bí Mật] Object & Class – Đặc tính

- Class chứa các đặc điểm của một thực thể, Object chứa các thông tin chi tiết về thực thể đó.
- Một hoặc nhiều Object có thể cùng một Class.

Vehicle	
	name: string
	type: string
	manufacturer: string
	...
	run(): void
	stop(): void
	...



【Bí Mật】 Object & Class – Đặc tính

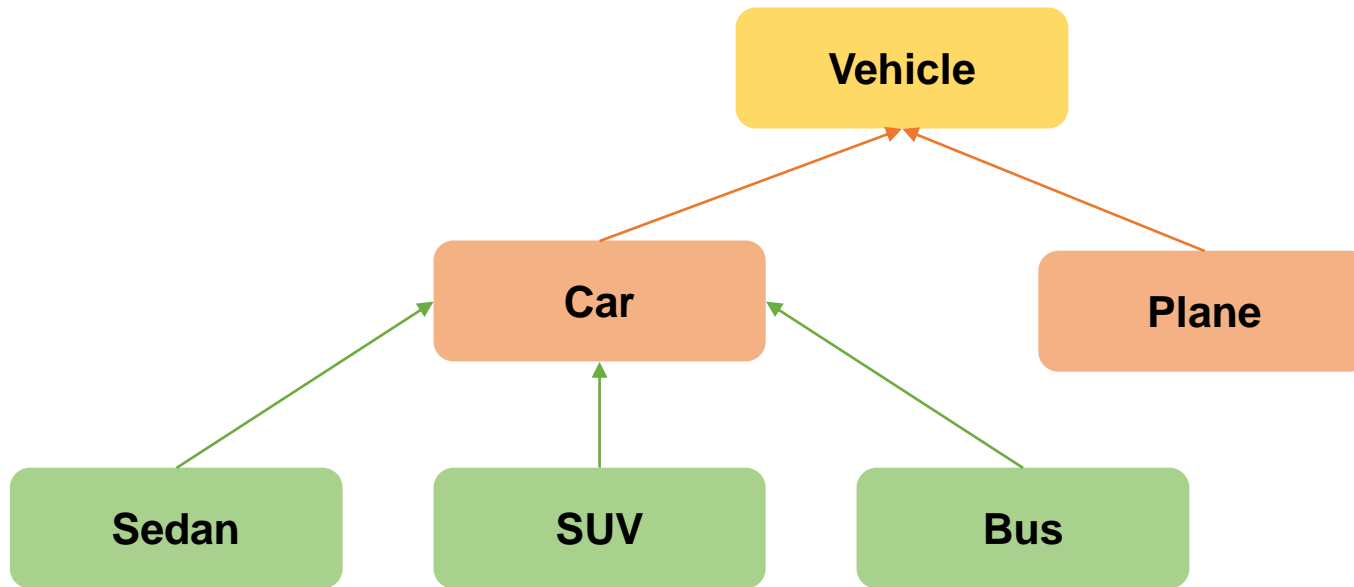
- Các Object cùng một Class có chung các thuộc tính và phương thức, nhưng giá trị của các thuộc tính và phương thức hoàn toàn độc lập, có thể giống, cũng có thể khác nhau.



Vehicle	Porsche 911	Airbus A320
name	Porsche 911	Airbus A320
type	Sport Car	Plane
manufacturer	Porsche	Airbus

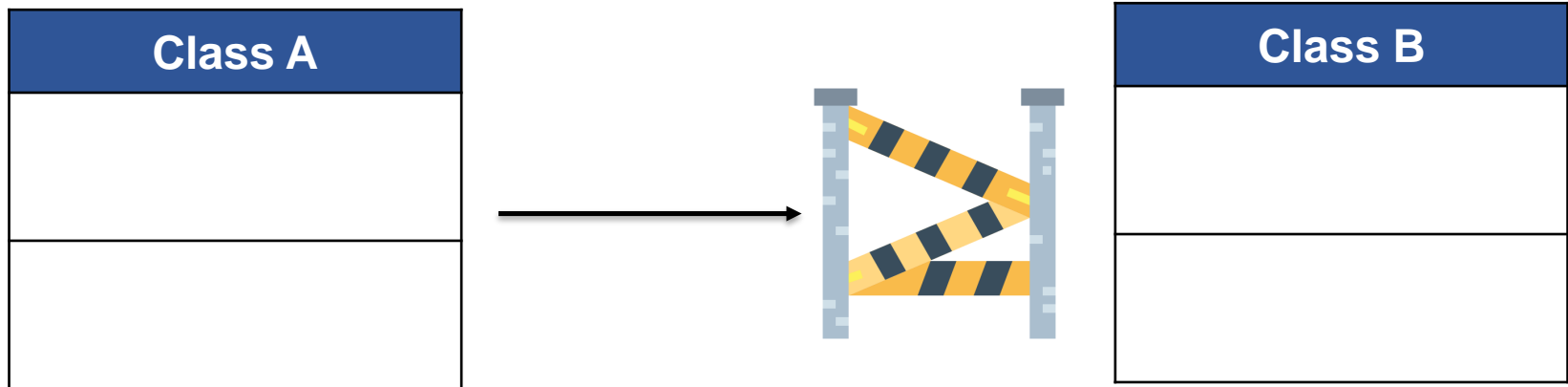
【Bí Mật】 Inheritance

- Inheritance (Kế thừa) là một đặc điểm của Object-Oriented Programming (OOP, Lập trình Hướng đối tượng). Mục đích của việc kế thừa chính là nhằm tái sử dụng những đoạn code đã tồn tại.
- Một Class có thể kế thừa thuộc tính và phương thức từ một Class khác, cũng có thể được một Class khác kế thừa thuộc tính và phương thức.



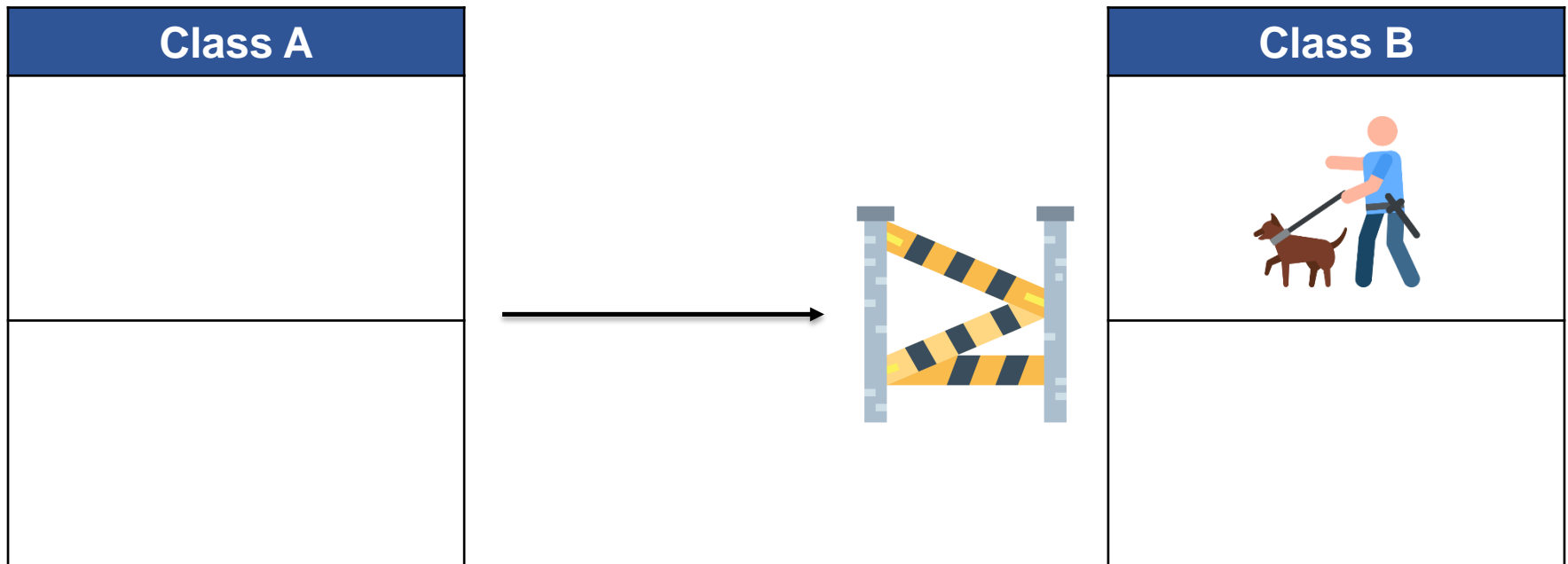
【Bí Mật】 Scope – Khái niệm

- Scope (Tầm vực) là một giá trị được tạo ra để giới hạn khả năng truy cập một thuộc tính hay một phương thức từ bên trong hoặc bên ngoài một Class.



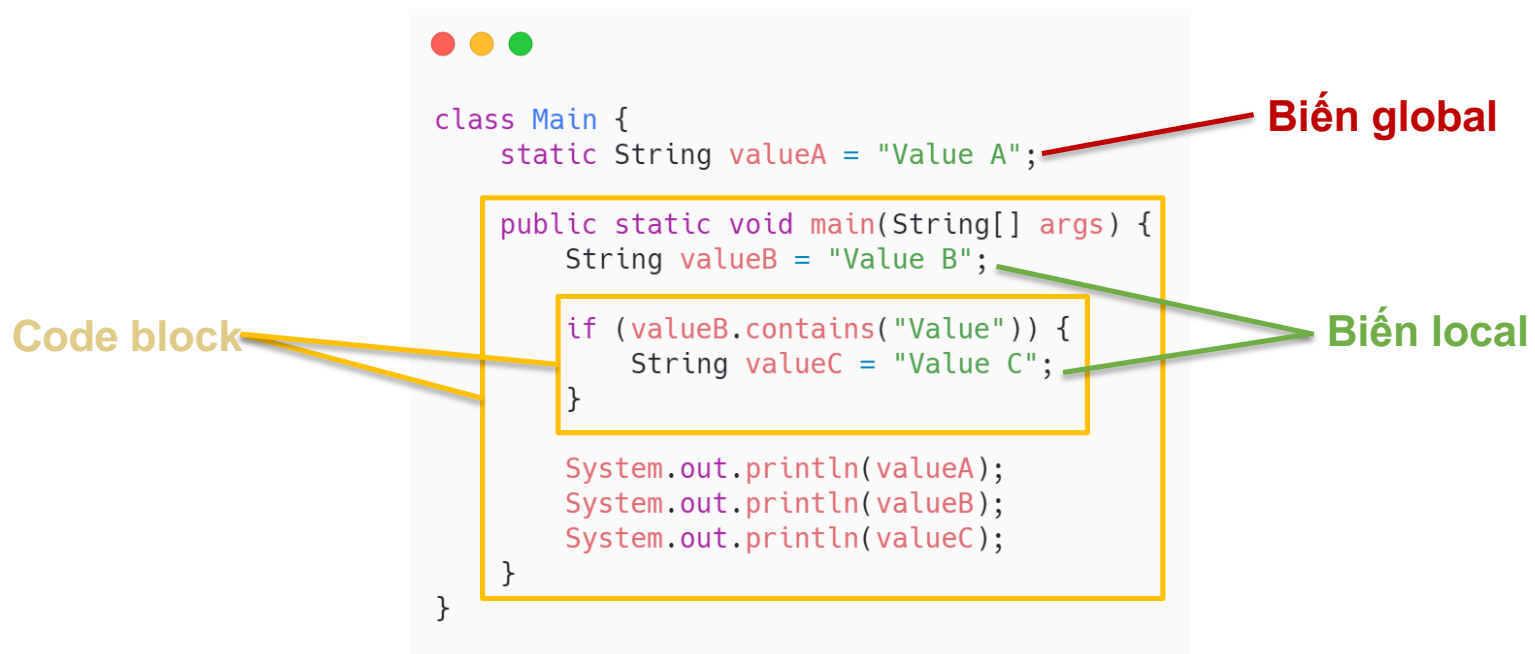
【Bí Mật】 Scope – Phân loại

- Có hai loại Scope mà Developer (Lập trình viên) cần chú ý:
 - Scope được sử dụng để kiểm soát các Code block. (Block Scope)
 - Scope được sử dụng để kiểm soát khả năng tiếp cận một thuộc tính hoặc một phương thức của một Class từ một Class khác. (Access Modifier)



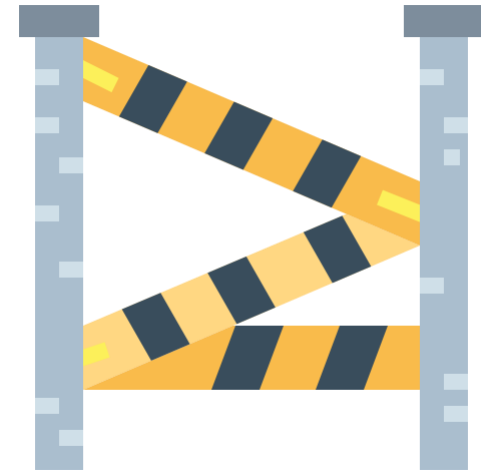
Scope – Block Scope

- Block Scope là loại Scope được sử dụng khi khai báo biến hoặc hằng trong một Class.
- Các biến và hằng sẽ được giới hạn khả năng truy xuất phụ thuộc vào vị trí mà chúng được khai báo.



Scope – Access Modifier

- Access Modifier là loại Scope được sử dụng khi khai báo các thuộc tính và phương thức trong một Class.
- Các thuộc tính và phương thức sẽ được giới hạn khả năng truy xuất bởi một từ khóa tương ứng.
- Có ba từ khóa thông dụng đó là:
 - Private
 - Protected
 - Public



[Bí Mật] Scope – Outer-class Scope

Biến **private**

Biến **protected**

Biến **public**

Kế thừa
constructor của
lớp cha bằng từ
khóa **super**

Thuộc tính
public có thể gọi
trực tiếp ở bất cứ
class nào

```
class Vehicle {
    private String name;
    protected String type;
    public String manufacturer;

    public Vehicle(String name) { this.name = name; }

    public String getName() { return this.name; }
}

class Car extends Vehicle {
    public Car(String name, String type) {
        super(name);
        this.type = type;
    }

    public void printInfo() {
        System.out.println("---INFO---");
        System.out.println("Name: " + this.getName());
        System.out.println("Type: " + this.type);
        System.out.println("Manufacturer: " + this.manufacturer);
        System.out.println("-----");
    }
}

class Main {
    public static void main(String[] args) {
        Car myCar = new Car("Porsche 911", "Sport Car");
        myCar.manufacturer = "Porsche";
        myCar.printInfo();
    }
}
```

Thuộc tính
private không thể
gọi trực tiếp

Thuộc tính **protected**
có thể gọi **trực tiếp** ở
lớp con

【Bí Mật】 Static & Non-static


- Static là từ khóa giúp hệ thống biết cách mà các thuộc tính và phương thức sẽ ghi vào vùng nhớ hệ thống.
- Nếu một thuộc tính hoặc phương thức là Static thì từ lúc ứng dụng bắt đầu chạy, các giá trị này sẽ được khởi tạo và ghi vào vùng nhớ của hệ thống.



```
class Feature {  
    static String valueA = "Value A";  
}  
  
class Main {  
    public static void main(String[] args) {  
        System.out.println(Feature.valueA);  
    }  
}
```

【Bí Mật】 Static & Non-static

- Nếu một thuộc tính hoặc phương thức là Non-static thì chỉ khi một Object của Class chứa thuộc tính hoặc phương thức này được khởi tạo thì thuộc tính hoặc phương thức này mới được khởi tạo và chỉ có thể truy xuất qua Object đó.



```
class Feature {  
    String valueA = "Value A";  
}  
  
class Main {  
    public static void main(String[] args) {  
        Feature fnc = new Feature();  
        System.out.println(fnc.valueA);  
    }  
}
```

Abstract & Interface – Interface

- Interface không phải là Class, có thể xem Interface là một khuôn mẫu chứa những thuộc tính và phương thức rỗng.
- Một Class có thể implements nhiều Interface. Khi Class implements một Interface thì phải triển khai tất cả phương thức mà Interface đã định nghĩa.

```
interface IFeature1 {  
    public void printInfo();  
}  
  
interface IFeature2 {  
    public void printPrice();  
}  
  
class Main implements IFeature1, IFeature2 {  
    public static void main(String[] args) {  
        System.out.println("Hello world!");  
    }  
  
    @Override  
    public void printInfo() {}  
  
    @Override  
    public void printPrice() {}  
}
```

Abstract & Interface – Abstract

- Abstract class là một Class nhưng lại khá giống Interface.
- Một Abstract class bao gồm:
 - Các thuộc tính chưa gán giá trị và các hằng.
 - Abstract methods – những phương thức trống.
 - Các phương thức thông thường.
- Một Class có thể extends duy nhất một Abstract class. Khi Class extends một Abstract class thì phải triển khai tất cả Abstract method mà Abstract class đã định nghĩa.

Abstract & Interface – Abstract

Phương thức
trong Interface

Phương thức
Abstract

```
abstract class Employee {
    void diemDanh() {}
    abstract int tinhLuong();
    abstract int tinhBonus();
}
```

```
interface ITraining {
    public void trainOOP();
    public void trainSQL();
}
```

```
class Fresher extends Employee implements ITraining {
    int tinhLuong() { return 400; };
    int tinhBonus() { return 100; };
    public void trainOOP() {};
    public void trainSQL() {};
}
```

```
class Senior extends Employee {
    int tinhLuong() { return 2000; };
    int tinhBonus() { return 500; };
}
```

【Bí Mật】 Generics

- Generics là một phương pháp giúp Developer tạo ra các Class hoặc các phương thức mà không cần quan tâm đến khai báo một kiểu dữ liệu cứng cho chúng.
- Generics là một trong những phương pháp lập trình giúp sử dụng hiệu quả code và giúp giảm tải tài nguyên bên cạnh extends và implements đối với Class, cũng như Overload và Override đối với phương thức.



```
class Sedan {}  
  
class SUV {}  
  
class Car<T> {}  
  
class Main {  
    public static void main(String[] args) {  
        Car<Sedan> mySedan = new Car();  
        Car<SUV> mySUV = new Car();  
    }  
}
```