Lập trình JAVA cơ bản

Tuần 2

Giảng viên: Trần Đức Minh

Nội dung bài giảng

- Lớp
- Lớp Mutable và Immutable
- Quan hệ bao hàm
- Kế thừa
- Từ khóa this và super
- Nạp chồng
- Tính đóng gói
- Đa hình

Lớp

- Định nghĩa lớp

  [<phạm vi>] **class** <Tên lớp> {

  <phạm vi> <kiểu dữ liệu> <tên thuộc tính>;

  <phạm vi> <kiểu trả về> <tên phương thức>() { … }

  }

- Ví dụ:

```
public class XeHoi {
    private double dDungTichXiLanh;
    private String strNhanHieu;
    private String strMauSac;

    public void setDungTichXiLanh(double dDungTichXiLanh) {
        this.dDungTichXiLanh = dDungTichXiLanh;
    }

    public String getNhanHieu(String strNhanHieu) {
        return this.strNhanHieu;
    }
}
```

Lớp

- Sử dụng lớp

  <Tên lớp> <tên đối tượng> = **new** <Tên lớp>(...);

  <tên đối tượng>.<tên phương thức>(…);

- Ví dụ:

```
XeHoi xehoi = new XeHoi();

xehoi.setMauSac(“Màu đỏ”);
double dCham = xehoi.getDungTichXiLanh();
System.out.println(“Nhãn hiệu xe là: “ +
xehoi.getNhanHieu());
```

Cấu tử

- Cấu tử được sử dụng để khởi tạo các giá trị ban đầu cho các thuộc tính trong lớp.
- Cấu tử có tên trùng với tên lớp và không có kiểu dữ liệu trả về.
- Ta nên quá tải cấu tử để phủ (cover) các trường hợp có thể xảy ra.
- Ví dụ:

```
public class XeHoi {
    ...
    public XeHoi() {
        this(“Toyota”, 1.2, “Trắng”);
    }
    public XeHoi(String strNhanHieu) {
        this.strNhanHieu = strNhanHieu;
    }
    public XeHoi(String strNhanHieu, double dDungTichXiLanh, String strMauSac) {
        this.strNhanHieu = strNhanHieu;
        this.dDungTichXiLanh = dDungTichXiLanh;
        this.strMauSac = strMauSac;
    }
}
```

Cấu tử

- Cấu tử tự động được gọi đến khi khởi tạo đối tượng, tùy vào tham số tương ứng.
- Phân tích đoạn code sau:
  - XeHoi xehoi = new XeHoi();
  - XeHoi xehoi1 = new XeHoi(“Toyota”);
  - XeHoi xehoi2 = new XeHoi(“Nissan”, 1.8, “Vàng”);

Lớp Mutable và Immutable

- **Mutable Class**: Là một lớp mà khi khởi tạo đối tượng của lớp đó thì **trạng thái của đối tượng** có thể thay đổi sau khi khởi tạo đối tượng thành công.
  - ***Đặc điểm***: Khi xây dựng lớp có cả hai phương thức **get và set**.
- **Immutable Class**: Là một lớp mà khi khởi tạo đối tượng của lớp đó thì **trạng thái của đối tượng không thể thay đổi** được sau khi khởi tạo đối tượng thành công.
  - ***Đặc điểm***: Khi xây dựng lớp chỉ có phương thức **get**.

Lớp Mutable và Immutable

- Ví dụ:

```java
class MutableClass {
    private String name;

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

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

```java
final class ImmutableClass {

    private String name;

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

    public String getName() {
        return name;
    }
}
```

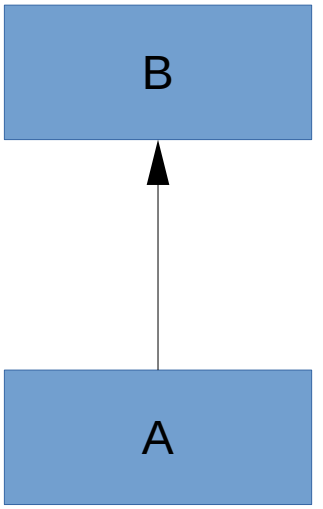
Quan hệ bao hàm

- **Quan hệ bao hàm** (composition) nghĩa là thành phần thuộc tính của lớp này là đối tượng của lớp kia.
- Ví dụ:
  - **Lớp Circle chứa đối tượng lớp Point** dùng để lưu trữ tâm của hình tròn.

```
class Point {
    private int x;
    private int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}

public class Circle {
    Point center;
    double radius;

    public Circle(Point center, double radius) {
        this.center = center;
        this.radius = radius;
    }

    public static void main(String[] args) {
        Point point = new Point( x: 4,  y: 5);
        Circle circle = new Circle(point,  radius: 3.6);
    }
}
```

Kế thừa

- **Tính kế thừa** được sử dụng để tạo ra một lớp mới dựa trên các lớp đã tồn tại.
- Một lớp A (**lớp dẫn xuất – subclass**) kế thừa từ một lớp đã tồn tại B (**lớp cơ sở – superclass**) thì:
  - Lớp A có thể sử dụng lại các phương thức và thuộc tính của lớp B (mà có phạm vi là **public** hoặc **protected**)
  - Lớp A có thể khai báo thêm các phương thức và thuộc tính của riêng nó.
  - Lớp A không được phép truy cập đến thành viên **private** của lớp B.
- Trong Java chỉ có đơn kế thừa (tức là mỗi lớp mới được tạo ra chỉ được phép kế thừa từ duy nhất một lớp).
- Mặc định tất cả các lớp được tạo bởi Java và do người dùng tạo ra đều kế thừa từ lớp **java.lang.Object**.



Kế thừa

```
class <SuperClass> {
}

class <SubClass> extends <SuperClass> {
}
```

- Java sử dụng từ khóa **extends** để tạo lớp kế thừa.

Ví dụ kế thừa

- **Animal.java**

```
public class Animal {
    private int weight;
    private String name;

    public Animal(String name, int weight) {
        this.name = name;
        this.weight = weight;
    }

    public void eat() {
        System.out.println(“Animal.eat() is called”);
    }
}
```

Kế thừa

- **Fish.java**

```
public class Fish extends Animal {
    private String livingArea;

    public Fish(String name, int weight, String livingArea) {
        super(name, weight);
        this.livingArea = livingArea;
    }
}
```

- **Main.java**

```
public class Main {
    public static void main(String[] args) {
        Fish fish = new Fish("Cá voi sát thủ", 3500, "Đông Bắc Thái Bình Dương");
        fish.eat();
        System.out.println(fish.getName());
        System.out.println(fish.getLivingArea());
    }
}
```

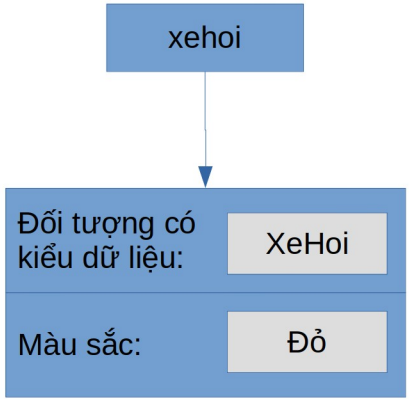
Lớp, đối tượng, thể hiện và tham chiếu

- Xây dựng Lớp XeHoi dạng Mutable với một thuộc tính màu sắc

```
public class XeHoi {
    private String strMauSac;

    public XeHoi(String strMauSac) {
        this.strMauSac = strMauSac;
    }

    public void setMauSac(String strMauSac) {
        this.strMauSac = strMauSac;
    }

    public String getMauSac() {
        return strMauSac;
    }
}
```

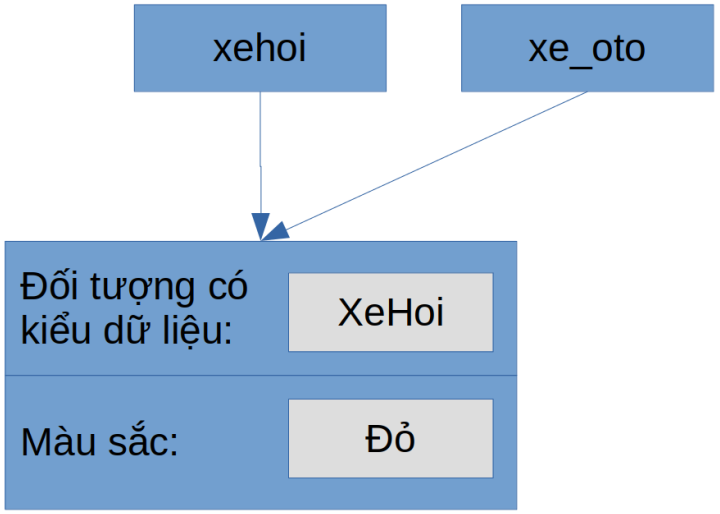
Lớp, đối tượng, thể hiện và tham chiếu

- **XeHoi xehoi = new XeHoi(“Đỏ”);**
- Dòng lệnh trên tạo ra một **thể hiện** (instance) của lớp **XeHoi** và gán nó cho biến **xehoi**.
- Ngoài ra, dòng lệnh trên còn có ý nghĩa là tạo ra một **tham chiếu** (reference) đến một **đối tượng** (object) bên trong bộ nhớ.



Lớp, đối tượng, thể hiện và tham chiếu

- **XeHoi xe_oto = xehoi;**
- Dòng lệnh trên tạo ra một **tham chiếu** khác đến cùng một đối tượng bên trong bộ nhớ. Tức là, ta chỉ có một đối tượng nhưng lại có 2 tham chiếu đến đối tượng đó.



Lớp, đối tượng, thể hiện và tham chiếu

- Phân tích đoạn code sau:

```
System.out.println(xehoi.getMauSac());
System.out.println(xe_oto.getMauSac());

xe_oto.setMauSac(“Xanh”);

System.out.println(xehoi.getMauSac());
System.out.println(xe_oto.getMauSac());
```

  - **Cho biết kết quả in ra màn hình và giải thích.**

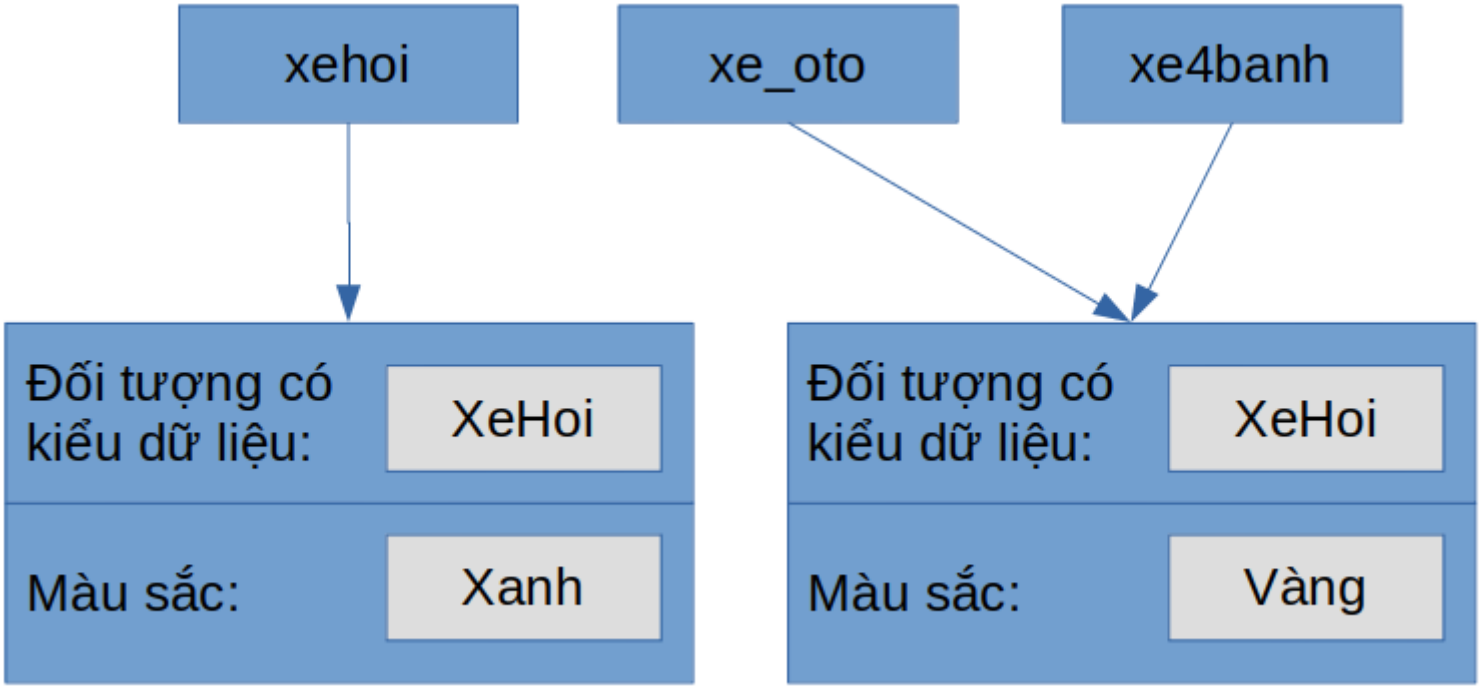
Lớp, đối tượng, thể hiện và tham chiếu

- Phân tích đoạn code sau:

```
XeHoi xe4banh = new XeHoi("Vang");
xe_oto = xe4banh;

System.out.println(xehoi.getMauSac());
System.out.println(xe4banh.getMauSac());
System.out.println(xe_oto.getMauSac());
```

  - **Cho biết kết quả in ra màn hình và giải thích.**

Lớp, đối tượng, thể hiện và tham chiếu



Từ khóa this và super

- Từ khóa **super** được sử dụng để truy cập hay gọi đến các thuộc tính và phương thức của lớp cơ sở (lớp cha).
- Từ khóa **this** được sử dụng để truy cập hay gọi đến các thuộc tính và phương thức của lớp hiện tại.
- Hai từ khóa này không được phép sử dụng trong vùng có phạm vi **static** (tĩnh).

Từ khóa this và super

- Phân tích đoạn code sau:

```
class SuperClass {
    public void printMethod() {
        System.out.println(“Printed in SuperClass.”);
    }
}

class SubClass extends SuperClass {
    public void printMethod() {
        super.printMethod();
        System.out.println(“Printed in SubClass.”);
    }
}

class MainClass {
    public static void main(String[] args) {
        SubClass sub = new SubClass();
        sub.printMethod();
    }
}
```

Hàm this()

- Hàm **this()** được dùng để gọi đến một cấu tử từ một **cấu tử quá tải** khác trong cùng một lớp.
- Hàm this() chỉ được phép gọi bên trong cấu tử và phải là câu lệnh đầu tiên bên trong cấu tử.
- Mục đích hàm this() được sử dụng là để giảm thiểu số mã lệnh bị lặp lại.

Hàm this()

- Chúng ta không nên viết code như hình bên:

```java
public class Rectangle {
    private int x;
    private int y;
    private int width;
    private int height;

    public Rectangle() {
        this.x = 0;
        this.y = 0;
        this.width = 0;
        this.height = 0;
    }

    public Rectangle(int width, int height) {
        this.x = 0;
        this.y = 0;
        this.width = width;
        this.height = height;
    }

    public Rectangle(int x, int y, int width, int height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }
}
```

Hàm this()

- Đoạn code nên được viết như sau:

```java
public class Rectangle {
    private int x;
    private int y;
    private int width;
    private int height;

    public Rectangle() {
        this( width: 0,  height: 0);
    }

    public Rectangle(int width, int height) {
        this( x: 0,  y: 0, width, height);
    }

    public Rectangle(int x, int y, int width, int height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }
}
```

Hàm super()

- Hàm **super()** được sử dụng để gọi đến cấu tử lớp cha (lớp cơ sở).
- Trình biên dịch Java luôn luôn đặt mặc định việc gọi đến hàm super() với không tham số.
- Hàm super() cũng phải là câu lệnh đầu tiên bên trong cấu tử.
- **Chú ý: Một cấu tử có thể thực hiện hàm this() hoặc hàm super() nhưng không bao giờ được phép thực hiện cả hai hàm này.**

Hàm super()

- Phân tích đoạn code sau:

```
class Shape {
    private int x;
    private int y;

    public Shape(int x, int y) {
        this.x = x;
        this.y = y;
    }
}

public class Rectangle extends Shape {
    private int width;
    private int height;

    public Rectangle(int x, int y) {
        this( x: 0,  y: 0,  width: 0,  height: 0);
    }

    public Rectangle(int x, int y, int width, int height) {
        super(x, y);
        this.width = width;
        this.height = height;
    }
}
```

Nạp chồng phương thức

- **Nạp chồng** (overriding) phương thức tức là định nghĩa một phương thức ở lớp con (lớp dẫn xuất) mà phương thức này đã được định nghĩa ở lớp cha (lớp cơ sở) (**trùng cả tên lẫn tham số**).
- Khi sử dụng nạp chồng, chúng ta nên thêm từ khóa @**Override** ở phía trên phương thức để khi trình biên dịch đọc, nó sẽ báo lỗi cho chúng ta biết nếu code của ta không tuân theo quy tắc nạp chồng phương thức.
- Không được phép nạp chồng phương thức tĩnh (static).
- Không được phép dùng **Access Modifier** thấp hơn đối với phương thức được nạp chồng
  - Ví dụ: Nếu phương thức được nạp chồng ở lớp cha có phạm vi là **protected** thì phương thức nạp chồng ở lớp con không được phép để phạm vi là **private**.

Nạp chồng phương thức

- Một số điểm cần ghi nhớ về nạp chồng
  - Các phương thức nạp chồng chỉ có thể được đặt ở trong các lớp con.
  - Cấu tử và các phương thức có phạm vi là **private** không được phép nạp chồng.
  - Các phương thức **final** không được phép nạp chồng.
  - Để gọi đến phương thức bị nạp chồng ở lớp cha, chúng ta sử dụng **super.<tên phương thức>()**

Nạp chồng phương thức

- Ví dụ:

```
class Dog {
    public void bark() {
        System.out.println(“gau gau gau”);
    }
}

class Wolf extends Dog {
    @Override
    public void bark() {
        System.out.println(“woof woof woof”);
    }
}
```

Nạp chồng phương thức

- Ví dụ: Gọi đến phương thức bị nạp chồng ở lớp cha

```
class Wolf extends Dog {
    @Override
    public void bark() {
        System.out.println(“woof woof woof”);
    }

    public void dogBark() {
        super.bark();
    }
}

public class MainClass {
    public static void main( String[] args ) {
        Wolf wolf = new Wolf();
        wolf.bark();
        wolf.dogBark();
    }
}
```

Quá tải phương thức trong lớp

- Chú ý việc quá tải phương thức trong khi kế thừa **có thể bị nhầm lẫn** với nạp chồng.
- Phân tích ví dụ sau:

```
class TwoNumber {
    public int getTotal(int x, int y) {
        return x + y;
    }
}

public class ThreeNumber extends TwoNumber {
    public int getTotal(int x, int y, int z) {
        return x + y + z;
    }

    public static void main(String[] args) {
        ThreeNumber threeNumber = new ThreeNumber();
        System.out.println(threeNumber.getTotal( x: 4,  y: 5));
        System.out.println(threeNumber.getTotal( x: 4,  y: 5,  z: 9));
    }
}
```

Phương thức tĩnh

- **Phương thức tĩnh** (static) không được phép truy cập (gọi) đến các phương thức instance (instance methods) và biến instance (instance variables) ở bên trong lớp.
- Phương thức tĩnh luôn được áp dụng cho các thao tác **KHÔNG** yêu cầu dữ liệu từ đối tượng của lớp (từ **this**).
- Phương thức tĩnh không được sử dụng từ khóa **this** ở bên trong.
- Ví dụ: **main()** là một phương thức tĩnh và được gọi bởi JVM khi bắt đầu ứng dụng.

Phương thức tĩnh

- Ví dụ: Phân tích vì sao chương trình lỗi.

```
public class MainClass {

    private int number1;
    private int number2;

    public int getTotal(int x, int y) {
        return x + y;
    }

    public static void main( String[] args ) {
        number1 = 5;
        this.number2 = 6;
        getTotal( x: 6,  y: 7);
    }
}
```

Phương thức tĩnh

- Truy cập phương thức tĩnh thông qua tên lớp

```
class Calculator {
    public static void printSum(int a, int b) {
        System.out.println(“sum = “ + (a + b));
    }
}

public class Main {
    public static void main(String[] args) {
        Calculator.printSum(6, 8);     // Gọi phương thức tĩnh của một lớp khác: Cần chỉ rõ tên lớp chứa phương thức
        printHello();                  // Gọi phương thức tĩnh trong cùng một lớp: Không cần chỉ rõ tên lớp chứa phương thức
    }

    public static void printHello() {
        System.out.println(“Hello”);
    }
}
```

Phương thức instance

- **Phương thức instance** (instance method) được gọi thông qua đối tượng của lớp chứa phương thức.
- Để sử dụng phương thức instance, trước tiên chúng ta phải khởi tạo đối tượng lớp bằng cách sử dụng từ khóa **new**.

```
class Dog {
    public void bark() {
        System.out.println("Gau gau gau");
    }
}
```

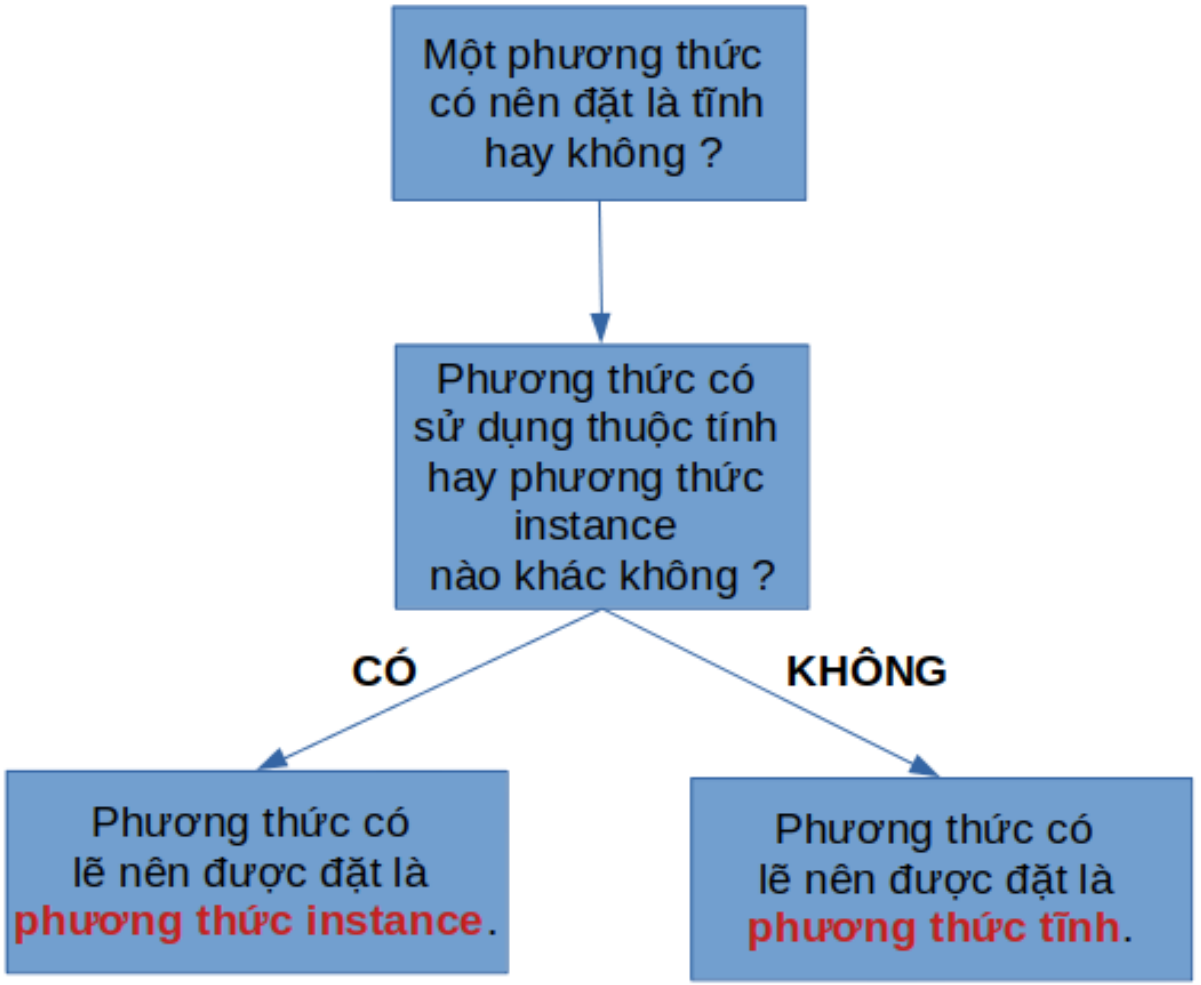
```
public class MainClass {
    public static void main(String[] args) {
        Dog lulu = new Dog();
        lulu.bark();
    }
}
```

Phương thức instance

- Phương thức instance có thể **truy cập** đến các **phương thức và biến instance** khác cũng như các **phương thức và biến tĩnh** trong cùng một lớp.

```java
public class MainClass {

    private int number1;
    private int number2;

    public MainClass(int number1, int number2) {
        this.number1 = number1;
        this.number2 = number2;
    }

    public static int getTotal(int x, int y) {
        return x + y;
    }

    public int myTotal() {
        return getTotal(this.number1, this.number2);
    }

    public static void main( String[] args ) {
        MainClass mc = new MainClass(8, 9);
        System.out.println(mc.myTotal());
    }
}
```

Chọn phương thức instance hay tĩnh



Biến tĩnh

- **Biến tĩnh** (static variable) có thể được dùng chung đối với tất cả các đối tượng của lớp. Nghĩa là nếu ta thay đổi giá trị của biến tĩnh thì các đối tượng sẽ thấy được sự thay đổi này.

Biến tĩnh

- Ví dụ: Hãy cho biết kết quả của đoạn chương trình sau

```
class Dog {
    private static String name;

    public Dog(String name) {
        Dog.name = name;
    }
    public void printName() {
        System.out.println("name = " + name);
    }
}

public class Main {
    public static void main(String[] args) {
        Dog lulu = new Dog("lulu");
        Dog fluffy = new Dog("fluffy");
        lulu.printName();
        fluffy.printName();
    }
}
```

Biến instance

- Tất cả các đối tượng của lớp đều chứa một bản sao của **biến instance** (instance variable). Do đó, mỗi biến instance có thể có giá trị khác nhau.

Biến instance

- Ví dụ:

```
class Dog {
    private String name;

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

    public void printName() {
        System.out.println(“name = “ + name);
    }
}

public class Main {
    public static void main(String[] args) {
        Dog lulu = new Dog(“lulu”);
        Dog fluffy = new Dog(“fluffy”);
        lulu.printName();
        fluffy.printName();
    }
}
```

Tính đóng gói

- **Tính đóng gói** (encapsulation) tạo ra cơ chế ngăn ngừa việc gọi phương thức của lớp này nhưng lại tác động hay truy xuất dữ liệu của đối tượng lớp khác.
- Dựa vào tính đóng gói để ngăn ngừa việc gán giá trị không hợp lệ vào thành phần dữ liệu của mỗi đối tượng (khai báo **private**).
- Cho phép thay đổi cấu trúc bên trong của một lớp mà không làm ảnh hưởng đến các lớp đã sử dụng lớp đó.
- Ví dụ: Các phương thức **get** và **set** dùng để đặt và lấy giá trị thuộc tính có phạm vi **private** chính là một phần của tính đóng gói.

Đa hình

- **Tính đa hình** (polymorphism) được hiểu là tùy trong từng ngữ cảnh, từng trường hợp, hoàn cảnh khác nhau thì đối tượng có hình thái khác nhau.
- Để thực hiện tính đa hình trong Java ta cần đảm bảo những quy tắc sau:
  - Các lớp con đều phải có quan hệ kế thừa với một lớp cha nào đó.
  - Phương thức đa hình phải được nạp chồng ở lớp con.

Đa hình

- Ví dụ:

```
class TaiKhoanNganHang {
    public double laiSuat() {
        return 0;
    }
}
```

- class TaiKhoanVCB extends TaiKhoanNganHang {

```
    public double laiSuat() {
        return 8.5;
    }
}
```

- class TaiKhoanAGB extends TaiKhoanNganHang {
  - public double laiSuat() {

```
        return 7.8;
    }
}
```

- class TaiKhoanTCB extends TaiKhoanNganHang {

```
    public double laiSuat() {
        return 9.2;
    }
}
```

Đa hình

```
public class MainClass {
    public static void main(String[] args) {
        TaiKhoanNganHang tk1 = new TaiKhoanVCB();
        TaiKhoanNganHang tk2 = new TaiKhoanAGB();
        TaiKhoanNganHang tk3 = new TaiKhoanTCB();

        System.out.println("Lãi suất VCB là: " + tk1.laiSuat());
        System.out.println("Lãi suất ARG là: " + tk2.laiSuat());
        System.out.println("Lãi suất CTG là: " + tk3.laiSuat());
    }
}
```

Đa hình

- Khi **thiết kế bài toán muốn áp dụng tính đa hình** ta cần thực hiện các công việc sau:
  - Tìm các **đặc điểm chung** của toàn bộ thực thể trong bài toán, sau đó đưa những đặc điểm chung **(thuộc tính, phương thức)** này vào cùng một lớp **(lớp cơ sở)**
  - Tiếp theo, tương ứng với mỗi khía cạnh của từng thực thể trong bài toán, ta xây dựng lớp kế thừa từ lớp cơ sở trên.
  - Nạp chồng các phương thức để chi tiết hóa các đặc thù riêng biệt liên quan đến từng khía cạnh của thực thể.

Hết Tuần 2

Cảm ơn các bạn đã chú ý lắng nghe !!!