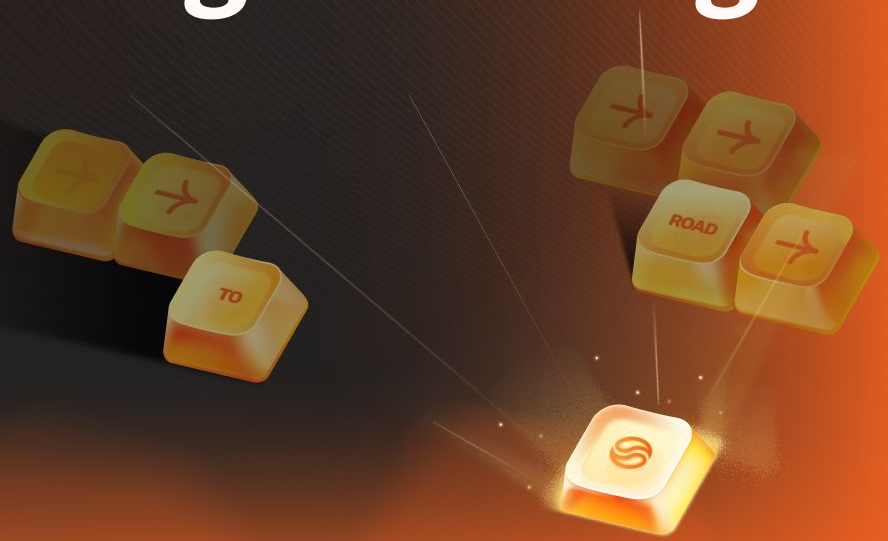
Ministry of SMEs and Startups × KOSME Korea SMEs and Startups Agency × LIKELION

Object-Oriented Programming

Nguyễn Anh Tuấn



K TECH COLLEGE

Nội dung bài giảng

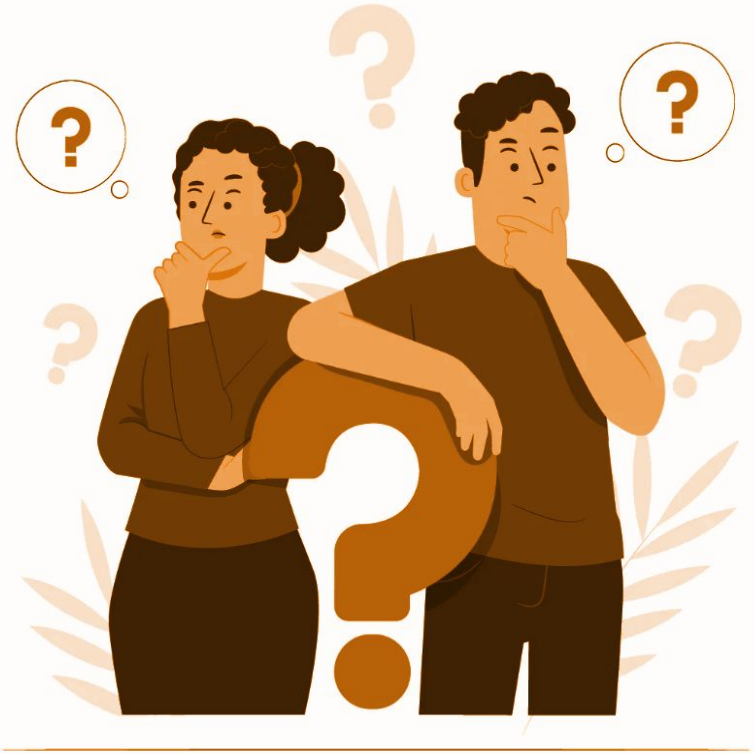
1 Object-Oriented Programming

2 Constructor trong class

3 Từ khoá final

4 Từ khoá static

What is Object Oriented Programming?



Object-Oriented Programming (OOP)

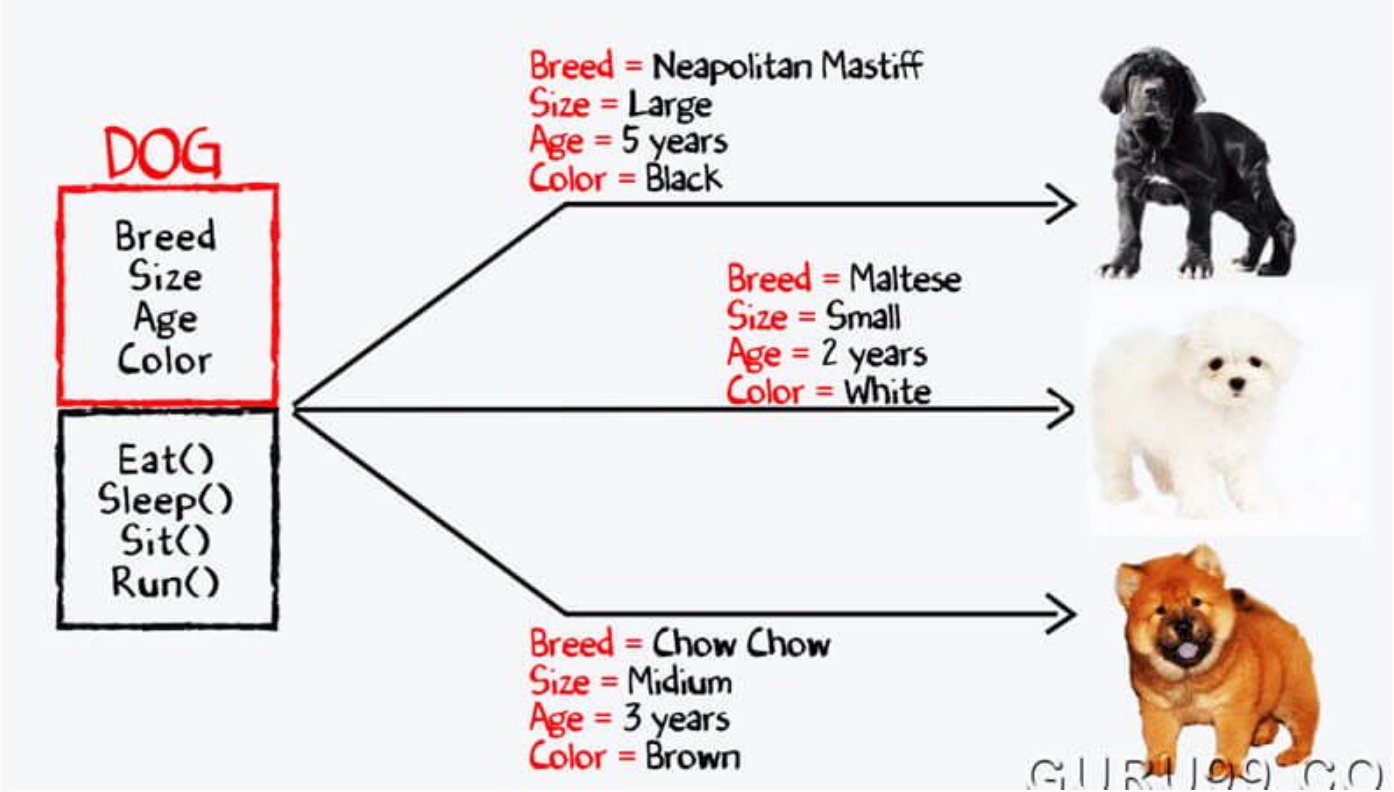
- **Định nghĩa**:
  - Lập trình hướng đối tượng là mô phỏng thế giới thực vào chương trình máy tính.
  - Lập trình hướng đối tượng là phương pháp lập trình lấy đối tượng làm nền tảng để xây dựng chương trình.

- **Các lợi thế của OOP**:
  - Có cấu trúc rõ ràng
  - Dễ dàng maintain (bảo trì), sửa đổi (modify), nâng cấp (upgrade), kiểm lỗi (debug), sửa lỗi (fix bug)
  - Hữu ích khi giải các bài toán lớn, phức tạp

Object-Oriented Programming (OOP)

- **1 ví dụ mô phỏng các đối tượng, thành phần trong bài Assignment, đề 1:**
  - Danh sách người thuê sách
    - 1 người thuê sách cụ thể
      - Tên
      - Tuổi
      - Danh sách sách người này thuê
        - 1 cuốn sách cụ thể mà người này thuê
          - Tên sách
          - Số lượng thuê
          - Ngày thuê
          - Ngày trả

Object-Oriented Programming (OOP)



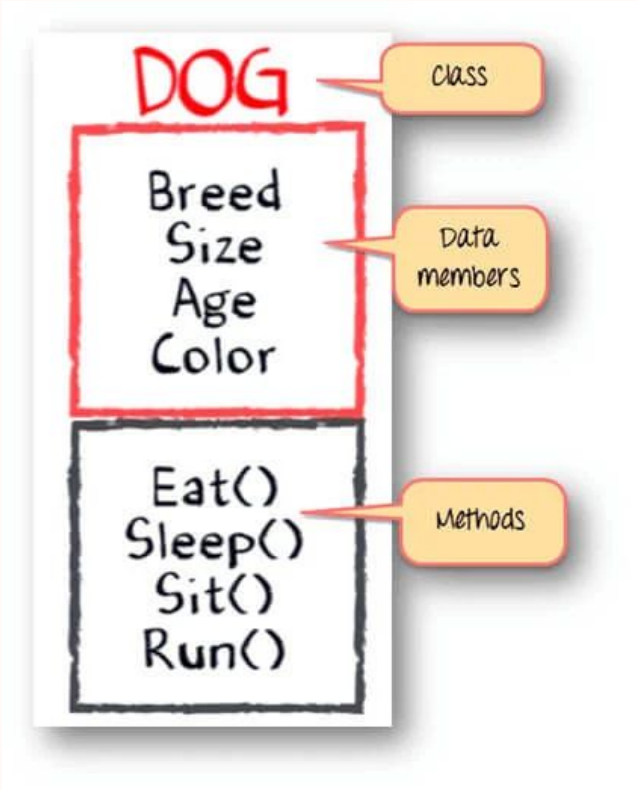
Object-Oriented Programming

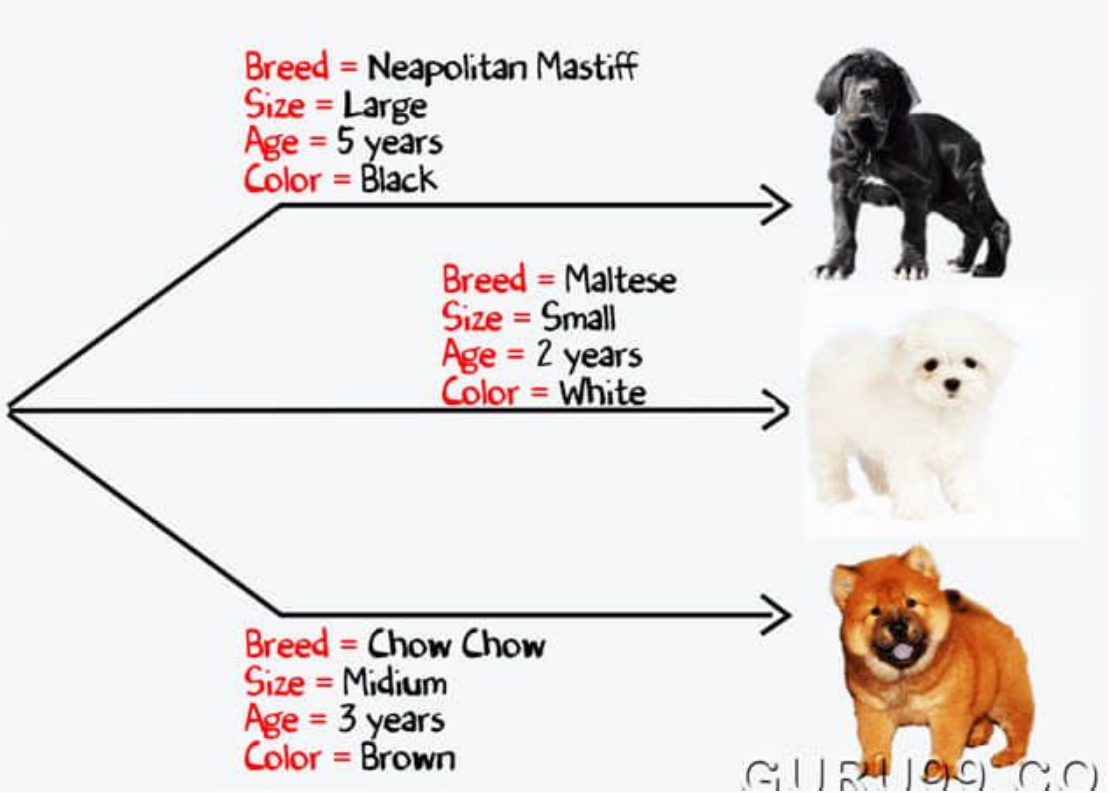
- **Như vậy:** Object-Oriented Programming (OOP) là mô phỏng 1 thế giới ảo bằng những gì có trong thực tế.

HOW

- **Class**: Đối tượng khái quát - Miêu tả khái quát về đối tượng trong thực tế
- **Object**: Đối tượng cụ thể - Thực thể (instance) của class đó, miêu tả chi tiết về đối tượng của class đó
- Mỗi đối tượng đều được mô tả bằng 2 thứ:
  - Đặc điểm (thuộc tính - attributes)
  - Hành vi (phương thức - method)

Object-Oriented Programming (OOP)





Object-Oriented Programming

❖ **Sử dụng đối tượng khái quát để mô hình hoá sinh viên, sau đó miêu tả về 2 đối tượng cụ thể là Tấn và Kiên**

Class

- **Đặc điểm:**
  - Tên
  - Tuổi
  - Chiều cao
- **Hành vi:**
  - Học
  - Ngủ
  - Giải trí

Object

- **Đặc điểm:**
  - Tấn
  - 16 tuổi
  - 170 cm
- **Hành vi:**
  - Học trong tủ
  - Ngủ trong lớp
  - Xem youtube đợi qua môn

Object

- **Đặc điểm:**
  - Kiên
  - 15 tuổi
  - 180 cm
- **Hành vi:**
  - Học trong tủ
  - Ngủ trong lớp
  - Xem youtube đợi qua môn

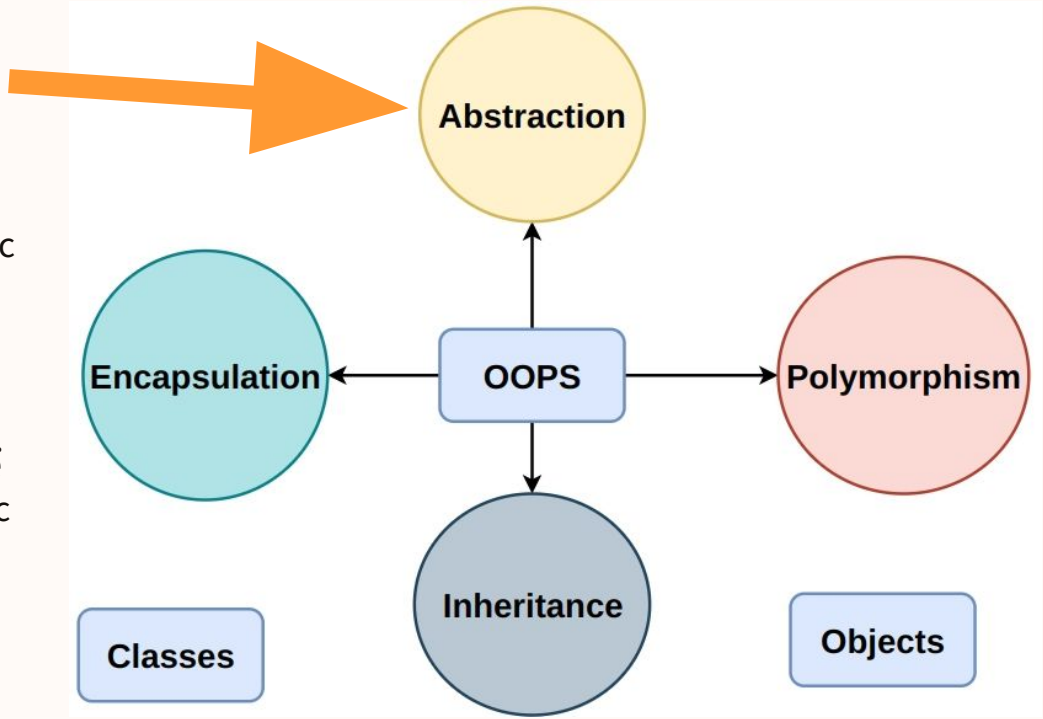
Object-Oriented Programming (OOP)

```
        // Tạo 1 đối tượng khái quát tên là Student
        // Mô tả khái quát đặc điểm là có tên, tuổi, chiều cao
        public class Student {
                String name;
                int age;
                int height;

                public String learn(String str) { return str; }
        }
```

```
public class Application {
        public static void main(String[] args) {
                // Khai báo đối tượng tan từ lớp Student
                Student tan = new Student();
                p.name = “Tấn"; // Gán tên cho tan
                p.age = 16; // Gán tuổi cho tan
                p.height = 170; // Gán chiều cao cho tan
                p.learn(“Học Java”);

                // Khai báo đối tượng kien từ lớp Student
                Student kien = new Student();
                k.name = “Kiên"; // Gán tên cho kien
                k.age = 15; // Gán tuổi cho kien
                k.height = 180; // Gán chiều cao cho kien
                p.learn(“Học JAV”);
        }
}
```

Object-Oriented Programming

- **Abstraction (Trừu tượng)**: Sử dụng đối tượng khái quát để mô hình hoá đối tượng mà không miêu tả chi tiết.
- **Polymorphism (Đa hình)**: Một phương thức có thể có nhiều hình thức khác nhau.
- **Encapsulation (Đóng gói)**: Giấu dữ liệu và chỉ cho phép truy cập qua các phương thức công khai.
- **Inheritance (Kế thừa)**: Một lớp có thể kế thừa các thuộc tính và phương thức từ một lớp khác.



Constructor chaining in Java

Constructor

❖ **Định nghĩa:**

- ➢ Constructor trong java là method đặc biệt để khởi tạo các đối tượng
- ➢ Constructor được gọi tại thời điểm tạo đối tượng. Constructor khởi tạo các giá trị để cung cấp dữ liệu đặc điểm cho các đối tượng
- ➢ Khai báo của Constructor phải cùng tên với class (lớp) và không có giá trị trả về
- ➢ Trình biên dịch sẽ tự động tạo một constructor mặc định nếu trong lớp đó không khai báo constructor
- ➢ Có 2 kiểu của constructor:
  - ■ Constructor mặc định (không có tham số truyền vào – default constructors)
  - ■ Constructor tham số (parameter constructors)

❖ **Syntax của constructor:**

```
<Phạm vi truy cập> <Tên lớp>() {
        // Khởi tạo dữ liệu
}
```

Constructor

❖ **So sánh giữa Constructor và Phương thức (method)**

Constructor	Method
Constructor được sử dụng để khởi tạo trạng thái của một đối tượng.	Phương thức được sử dụng để thể hiện hành động của một đối tượng.
Constructor không có kiểu trả về.	Phương thức có kiểu trả về.
Trình biên dịch Java tạo ra constructor mặc định nếu bạn không có constructor nào.	Phương thức không được tạo ra bởi trình biên dịch Java.
Tên của constructor phải giống tên lớp.	Tên phương thức có thể giống hoặc khác tên lớp.

Constructor

❖ **Ví dụ về khai báo Constructor mặc định**

```
public class Student {
        String name;
        int age;

        // Khai báo constructor mặc định
        public Student() { }
}

public class Application {
        public static void main(String[] args) {
                // Khởi tạo đối tượng với constructor mặc định
                Student stu = new Student();
        }
}
```

Constructor

❖ **Ví dụ về khai báo Constructor tham số**

```
public class Application {
    public static void main(String[] args) {
        // Khởi tạo đối tượng với constructor có 1 tham số name
        Student stu1 = new Student("tên gì đó");
        // Khởi tạo đối tượng với constructor có 1 tham số age
        Student stu2 = new Student(18);
        // Khởi tạo đối tượng với constructor có 2 tham số name và age
        Student stu3 = new Student("đặt tên", 20);
    }
}

public class Student {
    String name;
    int age;
    public Student(String name) { this.name = name; }
    public Student(int age) { this.age = age; }
    public Student(String name, int age) { this.name = name; this.age = age; }
}
```

Nạp chồng (Overloading)

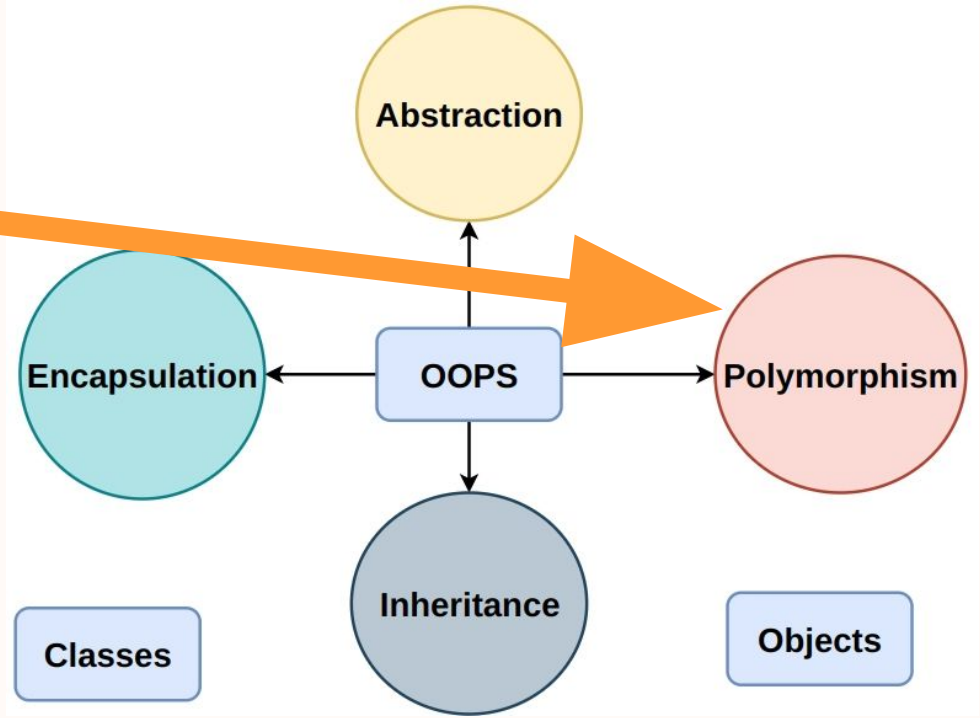
- **Overloading constructor**:
  - Có thể tạo nhiều constructor trong cùng 1 lớp với danh sách tham số và kiểu khác nhau
  - Trình biên dịch phân biệt các constructor này thông qua số lượng và kiểu của các tham số truyền vào

- **Overloading method**:
  - Tương tự như overloading constructor, có thể tạo nhiều method cùng tên với danh sách tham số và kiểu tham số khác nhau

- **Ý nghĩa của Overloading:**
  - Làm tăng tính hữu dụng của lớp sử dụng cơ chế nạp chồng

Object-Oriented Programming

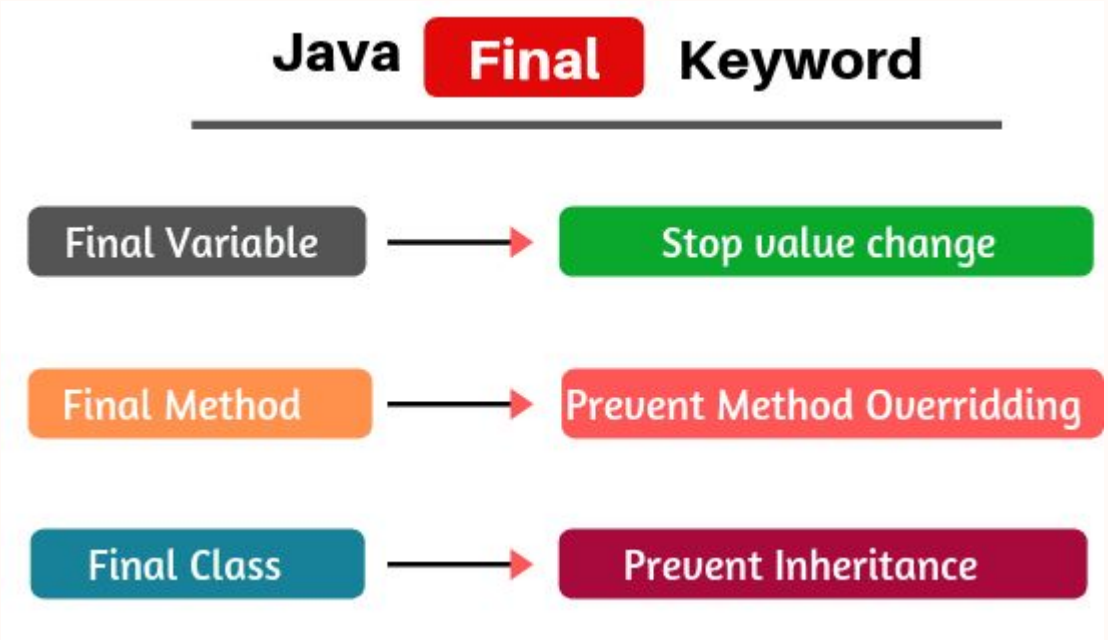
- **Abstraction (Trừu tượng)**: Sử dụng đối tượng khái quát để mô hình hoá đối tượng mà không miêu tả chi tiết.
- **Polymorphism (Đa hình)**: Một phương thức có thể có nhiều hình thức khác nhau.
- **Encapsulation (Đóng gói)**: Giấu dữ liệu và chỉ cho phép truy cập qua các phương thức công khai.
- **Inheritance (Kế thừa)**: Một lớp có thể kế thừa các thuộc tính và phương thức từ một lớp khác.



final trong Java

final trong Java

❖ Từ khóa **final** trong Java được sử dụng để hạn chế chỉnh sửa dữ liệu.



final trong Java

❖ **Ví dụ với biến final:**

```
public class Student {
        final MA_LOP_HOC = 112233; // Khởi tại 1 hằng số (1 biến final)

        public static void main(String[] args) {
                Student st = new Student();
                // Gán hằng số MA_LOP_HOC vào biến maLop
                int maLop = st.MA_LOP_HOC;

                // Gán giá trị khác vào hằng số MA_LOP_HOC
                st.MA_LOP_HOC = 123123;
                // Ko thể gán giá trị khác cho hằng số
                // Nếu cố run chương trình sẽ gặp lỗi complie error
        }
}
```

final trong Java

❖ **Ví dụ với biến final:**

```
public class Student {
        final MA_LOP_HOC; // Khởi tại 1 hằng số trống (1 biến final trống)

        public Student () {
                // Gán giá trị vào hằng số MA_LOP_HOC trong constructor
                MA_LOP_HOC = 112233;
                // Chỉ có thể gán giá trị vào hằng số MA_LOP_HOC trong constructor
        }
}
```

final trong Java

❖ **Ví dụ với phương thức final:**

```
public class Room {
        // Khởi tạo phương thức final test()
        final void test() { }
}

public class Student extends Room {

        // Ghi đè phương thức test của lớp Room
        @Override
        void test() { }
        // Phương thức final có thể kế thừa, nhưng ko thể ghi đè
        // Nếu cố run chương trình sẽ gặp lỗi exception
}
```

final trong Java

❖ **Ví dụ với lớp final:**

```
// Khởi tạo lớp final Room
final class Room {
        public void test() { }
}

// Kế thừa lớp Room
public class Student extends Room {

        // Ko thể kế thừa lớp final
        // Nếu cố run chương trình sẽ gặp lỗi compile error
}
```

final trong Java

❖ **Ví dụ với biến static final trống:**

```
public class Room {
        // Khởi tạo biến static final trống
        static final int MA_LOP_HOC;

        static {
                // Gán giá trị cho hằng số trong khối static
                MA_LOP_HOC = 112233;
                // Ko thể gán giá trị cho hằng số trong phương thức static
        }
}
```

final trong Java

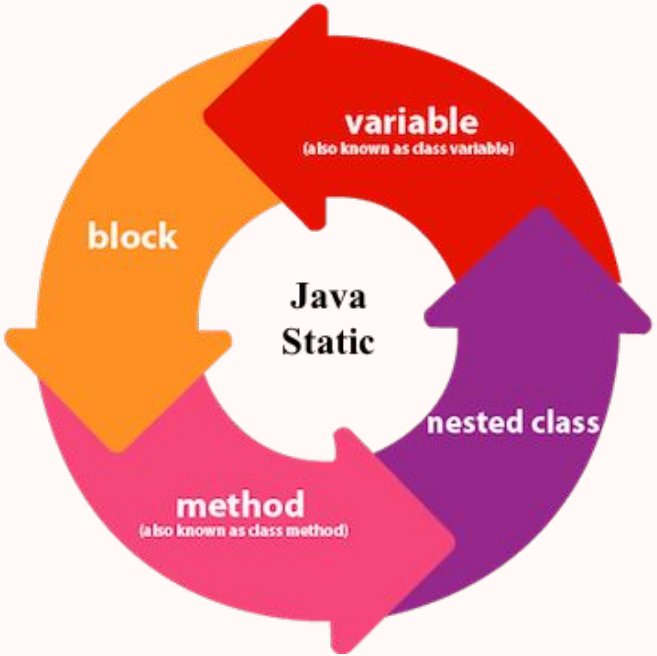
❖ **Ví dụ với tham số final:**

```
public class Student {
        // Khởi tạo phương thức có tham số final
        public int test (final int age) {
                // Thay đổi giá trị của age
                age = 112233;
                // Ko thể thay đổi giá trị của age. Nếu cố run chương trình sẽ gặp lỗi complie error
        }
}

public class Application {
        public static void main(String[] args) {
                Student stu = new Student();
                stu.test(123);
        }
}
```

static trong Java

final trong Java

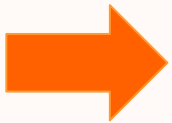
- Từ khoá **static** trong Java được sử dụng để quản lý bộ nhớ tốt hơn và nó có thể được truy cập trực tiếp thông qua lớp mà không cần khởi tạo



static trong Java

❖ **Ví dụ với biến static:**

```
class Student {

        // Khởi tạo biến thông thường
        String name;
        int age;
        String class = “Lớp Java";

        public Student(String name,
int age) {
                this.name = name;
                this.age = age;
        }
}
```



```
class Student {

        // Khởi tạo biến thông thường
        String name;
        int age;
        // Khởi tạo biến static
        static String class = “Lớp Java";
        // Biến class sẽ chỉ sử dụng bộ nhớ 1 lần

        public Student(String name, int age) {
                this.name = name;
                this.age = age;
        }
}
```

static trong Java

❖ **Ví dụ với phương thức static:**

```
public class Student {
        // Khởi tạo biến static
        static int age = 123;
        // Khởi tạo phương thức static để thay đổi giá trị của biến static
        static void test () { age = 456; }
}

public class Application {
        public static void main(String[] args) {
                // Một phương thức static có thể gọi method static
                // mà không cần tạo đối tượng của lớp đó
                Student.test();
        }
}
```

static trong Java

❖ **Ví dụ với khối static:**

```
public class Application {

        // Khởi tạo 1 khối static
        // Khối static được dùng để khởi tạo giá trị thuộc tính static
        static void test() {
                // Khối static được thực thi trước phương thức main
                System.out.println("Câu lệnh này chạy trước");
        }

        public static void main(String[] args) {
                System.out.println("Câu lệnh này chạy sau");
        }
}
```

Tại sao phương thức main trong Java là static?

ROAD TO KOREA

Nếu có bất kỳ thắc mắc nào, hãy đặt câu hỏi qua

mail@mail.com hoặc Zalo 0xxx xxx xxx