LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Hướng đối tượng - Object Oriented Programming hay OOP là một mô hình lập trình quan trọng, được ứng dụng rộng rãi trong phát triển phần mềm. OOP tập trung xoay quanh dữ liệu và đối tượng thay vì tập trung vào thủ tục và hàm như các mô hình lập trình truyền thống.

1. Class và Object:

Hai khái niệm quan trọng trong lập trình hướng đối tượng là: **Lớp (Class) và Đối tượng (Object).**

Mục tiêu của OOP đó là cố gắng đưa các thực thể, đối tượng trong thực tế vào phần mềm.

Phần mềm cần quản lý những đối tượng như Người, Xe, Sách thì khi đó, chúng được xây dựng trong phần mềm dưới dạng các lớp.

Lớp chính là bản mô phỏng của một đối tượng trong thực tế.

1. Class và Object:

Lớp có nhiệm vụ phác họa, mô phỏng những thông tin chung còn khái niệm về đối tượng của lớp lại là một thực thể cụ thể của lớp đó.

Ví dụ: Bạn cần xây dựng một lớp là Person để mô phỏng con người thì khi đó một đối tượng của lớp Person là “Nguyễn Văn Tèo” chính là một thực thể, một đối tượng của lớp Person.

2. Thuộc tính và phương thức:

Để mô tả thông tin của một lớp, bạn cần bổ sung **các thuộc tính (Attribute)** và **các phương thức (Method).**

VD: Lớp Person cần

Thông tin

- Tên
- Ngày sinh
- Số điện thoại
- Địa chỉ

Đây là các **thuộc tính** cần bổ sung cho lớp.

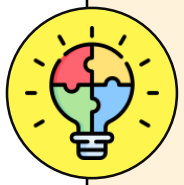
Hành động

- Đi lại
- Ăn uống
- Giao tiếp

Đây là các **phương thức** cần bổ sung cho lớp.

Person
Tên Ngày sinh Địa chỉ Số điện thoại
Đi lại() Ăn uống () Giao tiếp()

3. Xây dựng lớp:



Trong class sẽ chứa các member có thể là thuộc tính hoặc phương thức. Khi khai bao các member này thì bạn phải chỉ ra access_specifier cho các thành phần này.

CÚ PHÁP

```
class class_name{
    access_specifier1:
        member1, member2...
    access_specifier2:
        member1, member2...
};
```

Các access_specifier (tạm dịch là quyền truy cập): public, protected, private.

3. Xây dựng lớp:

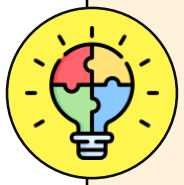
Đối với các member là thuộc tính bạn để quyền truy cập là private để đảm bảo tính chất đóng gói của OOP (Encapsulation). Khi quyền truy cập là private thì các thuộc tính này chỉ có thể truy cập bên trong phạm vi của lớp.

Đối với các member là phương thức bạn để quyền truy cập là public. Quyền truy cập protected sẽ được giải thích rõ hơn ở phần kế thừa.


```java
public class SinhVien {
    private String maSinhVien;
    private String hoTen;
    private double gpa;

    public void greet(){
        System.out.println("Hello !");
    }
}
```

4. Khai báo đối tượng:



Để khai báo một đối tượng của lớp các bạn sử dụng tên lớp như kiểu dữ liệu. Ngoài ra bạn cũng có thể khai báo mảng đối tượng, ArrayList đối tượng.

EXAMPLE

```
public static void main(String[] args)
{
        SinhVien s = new SinhVien();
        s.greet();
}
```

OUTPUT

Hello !

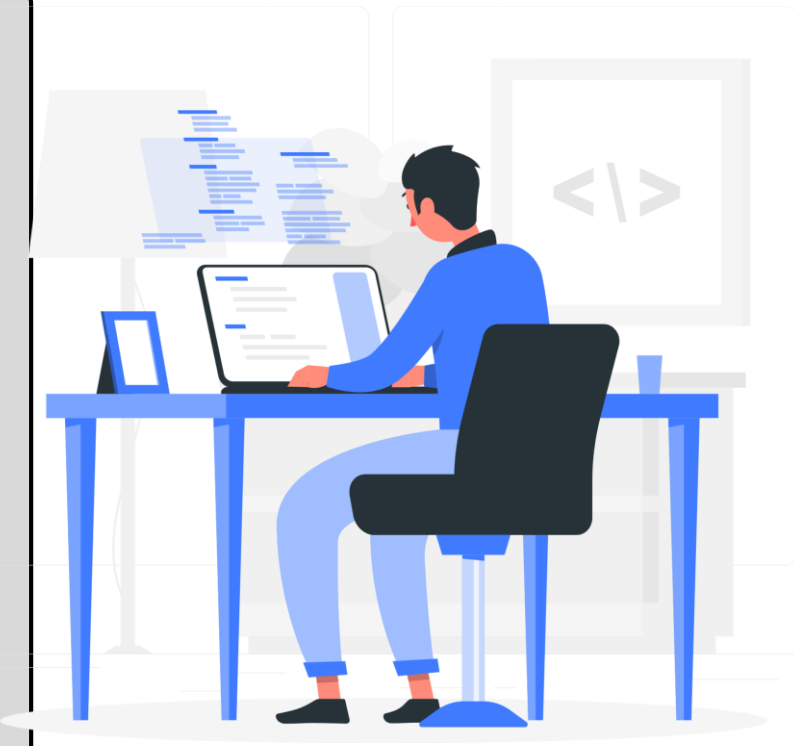
Để truy cập các phương thức và thuộc tính của đối tượng ta sử dụng **toán tử '.'**

Bạn không thể truy cập vào các member private của lớp bên ngoài phạm vi của lớp.

5. Hàm tạo:

Hàm tạo (Constructor)

- Là hàm được gọi mặc định khi bạn khai báo một đối tượng của 1 lớp nào đó, kể cả khi bạn không xây dựng hàm này thì hàm này vẫn tồn tại.
- Tuy nhiên bạn có thể tự xây dựng hàm tạo để nhanh chóng khởi tạo thông tin cho các thuộc tính của đối tượng bằng cách nạp chồng hàm tạo.
- Hàm tạo không có kiểu trả về, có tên trùng với tên lớp, tùy theo tham số bạn truyền vào cho đối tượng khi khai báo thì hàm tạo phù hợp sẽ được gọi.



5. Hàm tạo:

Lớp SinhVien

```
public class SinhVien {
    private String maSinhVien;
    private String hoTen;
    private double gpa;

    SinhVien(){
        System.out.println("Ham tao khong tham so");
    }
    SinhVien(String ma, String ten, double diem){
        maSinhVien = ma;
        hoTen = ten;
        gpa = diem;
    }
    public void display(){
        System.out.println(maSinhVien + " " + hoTen + " " + gpa);
    }
}
```

5. Hàm tạo:

Hàm main

```
public class Main {
     public static void main(String[] args) {
        SinhVien s = new SinhVien();
        SinhVien t = new SinhVien("123", "Teo", 3.5);
        s.display();
        t.display();
    }
}
```

OUTPUT

```
Ham tao khong tham so
null null 0.0
123 Teo 3.5
```

6. Con trỏ this, getter() và setter():

Bạn có thể sử dụng **con trỏ this** trước tên các thuộc tính hoặc phương thức để đảm bảo sự rõ ràng khi có những tham số trùng tên với các thuộc tính.

Getter(): Khi làm việc với các đối tượng, đôi khi bạn muốn lấy ra các thuộc tính của lớp nhưng không thể truy cập trực tiếp vào các thuộc tính này. Giải pháp: xây dựng phương thức getter() để lấy về thuộc tính mong muốn.

Setter(): Tương tự như getter(), khi bạn muốn thay đổi thuộc tính của đối tượng bạn phải sử dụng hàm setter để thay đổi vì không thể trực tiếp truy cập vào các thuộc tính này.

6. Con trỏ this, getter() và setter():

Con trỏ this:

```
public class SinhVien {
    private String maSinhVien;
    private String hoTen;
    private double gpa;

    SinhVien(){
        System.out.println("Ham tao khong tham so");
    }
    SinhVien(String maSinhVien, String hoTen, double gpa){
        this.maSinhVien = maSinhVien;
        this.hoTen = hoTen;
        this.gpa = gpa;
    }
    public void display(){
        System.out.println(this.maSinhVien + " " + this.hoTen + " " + this.gpa);
    }
}
```

6. Con trỏ this, getter() và setter():

Getter():

```
public class SinhVien {
    private String maSinhVien;
    private String hoTen;
    private double gpa;
    SinhVien(String maSinhVien, String hoTen, double gpa){
        this.maSinhVien = maSinhVien;
        this.hoTen = hoTen;
        this.gpa = gpa;
    }
    public String getMaSinhVien() {
        return maSinhVien;
    }
    public String getHoTen() {
        return hoTen;
    }
    public double getGpa() {
        return gpa;
    }
}
```

6. Con trỏ this, getter() và setter():

Setter():

```
public class SinhVien {
    private String maSinhVien;
    private String hoTen;
    private double gpa;
    SinhVien(String maSinhVien, String hoTen, double gpa){
        this.maSinhVien = maSinhVien;
        this.hoTen = hoTen;
        this.gpa = gpa;
    }
    public void setMaSinhVien(String maSinhVien) {
        this.maSinhVien = maSinhVien;
    }
    public void setHoTen(String hoTen) {
        this.hoTen = hoTen;
    }
    public void setGpa(double gpa) {
        this.gpa = gpa;
    }
}
```

7. Static keyword:

Biến static: Biến static thuộc về class chứ không thuộc về bất cứ một đối tượng nào của lớp, tức là các đối tượng của lớp sẽ chung biến static này.

```
public class SinhVien {
    private String maSinhVien;
    private String hoTen;
    private double gpa;
    private static int count = 0;
    SinhVien(String maSinhVien, String hoTen, double gpa){
        ++count;
        this.maSinhVien = maSinhVien;
        this.hoTen = hoTen;
        this.gpa = gpa;
    }
    public int getCount(){
        return count;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        SinhVien s = new SinhVien("123", "Teo", 3.2);
        SinhVien t = new SinhVien("124", "Ti", 3.5);
        System.out.println(s.getCount());
        System.out.println(t.getCount());
    }
}
```

OUTPUT

```
2 2
```

7. Static keyword:

Ngoài biến static ta còn hàm static, hàm static sẽ thuộc về class, bạn có thể gọi hàm static mà không cần thông qua đối tượng cụ thể của lớp. Hàm static có thể gọi thông qua tên lớp. Tuy nhiên bạn không thể sử dụng các thuộc tính, phương thức không phải static bên trong hàm static.

```
public class Main {
     public static void main(String[] args) {
        SinhVien s = new SinhVien("123", "Teo", 3.2);
        SinhVien t = new SinhVien("124", "Ti", 3.5);
        System.out.println(s.getCount());
        System.out.println(t.getCount());
        System.out.println(SinhVien.getCount());
    }
}
```

OUTPUT

```
2 2 2
```