Các công thức OOP - (Object-Oriented Programming - OOP)

Khái niêm: Lập trình hướng đối tượng: Tổ chức mã nguồn dựa trên đối tượng và class, mô phỏng các thực thể trong thế giới thực.

Ưu điểm: Dễ bảo trì, tái sử dụng,mở rộng, cấu trúc mã nguồn rõ ràng.

- Các ngôn ngữ: C++, c#,python, java, php

1. Lớp (Class)

- **Khái niệm:** Là khuôn mẫu để tạo ra các đối tượng. Lớp định nghĩa thuộc tính (dữ liệu) và phương thức (hành vi) của các đối tượng.

Ví dụ:

```
public class Car {

    // Thuộc tính

    String color;

    String model;



    // Phương thức

    void drive() {

        System.out.println("Car is driving");

    }
```

```
}
```

-

2. Đối tượng (Object)

- **Khái niệm:** Là một thể hiện cụ thể của lớp. Đối tượng có thuộc tính và hành vi được định nghĩa bởi lớp.

Ví dụ:

```
Car myCar = new Car();

myCar.color = "Red";

myCar.model = "Toyota";

myCar.drive();
```

3. Tính kế thừa (Inheritance)

- **Khái niệm:** Cho phép một lớp kế thừa các thuộc tính và phương thức từ một lớp khác. Lớp con (subclass) kế thừa từ lớp cha (superclass).

Ví dụ:

```
public class ElectricCar extends Car {

    // Lớp ElectricCar kế thừa từ lớp Car

    void charge() {

        System.out.println("Charging the car");

    }

}
```

-

4. Tính đóng gói (Encapsulation)

- **Khái niệm:** Đóng gói dữ liệu và phương thức trong lớp để bảo vệ dữ liệu khỏi bị truy cập trái phép. Sử dụng các modifier như `private`, `protected`, và `public` để kiểm soát quyền truy cập.

Ví dụ:

```
public class Person {

    private String name;



    // Getter

    public String getName() {

        return name;

    }



    // Setter

    public void setName(String name) {

        this.name = name;

    }

}
```

-

5. Tính đa hình (Polymorphism)

- **Khái niệm:** Cho phép các phương thức có cùng tên nhưng thực hiện các chức năng khác nhau. Có hai loại đa hình chính:
    - **Đa hình tại thời điểm biên dịch (Compile-time Polymorphism):** Thường được thực hiện bằng cách overload các phương thức.

- **Đa hình tại thời điểm chạy (Runtime Polymorphism):** Thực hiện qua overriding các phương thức trong lớp con.

Ví dụ (Đa hình tại thời điểm chạy):

```
class Animal {

    void makeSound() {

        System.out.println("Animal makes a sound");

    }

}



class Dog extends Animal {

    @Override

    void makeSound() {

        System.out.println("Dog barks");

    }

}



public class TestPolymorphism {

    public static void main(String[] args) {

        Animal myAnimal = new Dog(); // Lớp con Dog được gán
cho tham chiếu lớp cha Animal

        myAnimal.makeSound(); // Gọi phương thức của lớp con
Dog
```

```
    }

}
```

-

6. Abstraction (Trừu tượng)

- **Khái niệm:** Cho phép ẩn đi các chi tiết cài đặt và chỉ hiện ra những đặc điểm cần thiết. Trong Java, điều này có thể thực hiện thông qua lớp trừu tượng (abstract class) và giao diện (interface).

Ví dụ (Lớp trừu tượng):

```
abstract class Animal {

    abstract void makeSound(); // Phương thức trừu tượng

}



class Cat extends Animal {

    @Override

    void makeSound() {

        System.out.println("Cat meows");

    }

}
```

-

7. Giao diện (Interface)

- **Khái niệm:** Là một tập hợp các phương thức abstract (trừu tượng) mà các lớp có thể triển khai. Một lớp có thể triển khai nhiều giao diện.

Ví dụ:

```
interface Animal {

    void makeSound();

}



class Bird implements Animal {

    @Override

    public void makeSound() {

        System.out.println("Bird chirps");

    }

     }
```

1. Modifier `default` trong Interface

- **Khái niệm:** `default` được sử dụng để cung cấp cài đặt mặc định cho phương thức trong giao diện. Điều này giúp thêm phương thức mới vào giao diện mà không làm ảnh hưởng đến các lớp đã triển khai giao diện đó.

Cú pháp:

```
interface MyInterface {

    // Phương thức abstract (không có thân)

    void abstractMethod();
```

```
    // Phương thức default (có cài đặt mặc định)
    default void defaultMethod() {
        System.out.println("This is a default
method");
    }
}
```

-

Ví dụ:

```
interface Animal {
    void makeSound(); // Phương thức abstract

    default void sleep() {
        System.out.println("The animal is
sleeping");
    }
}

class Dog implements Animal {
    @Override
```

```
    public void makeSound() {
        System.out.println("Dog barks");
    }
}

public class TestDefaultMethod {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.makeSound(); // Output: Dog barks
        dog.sleep();     // Output: The animal is sleeping
    }
}
```

-

2. Modifier `default` trong Lớp (Class)

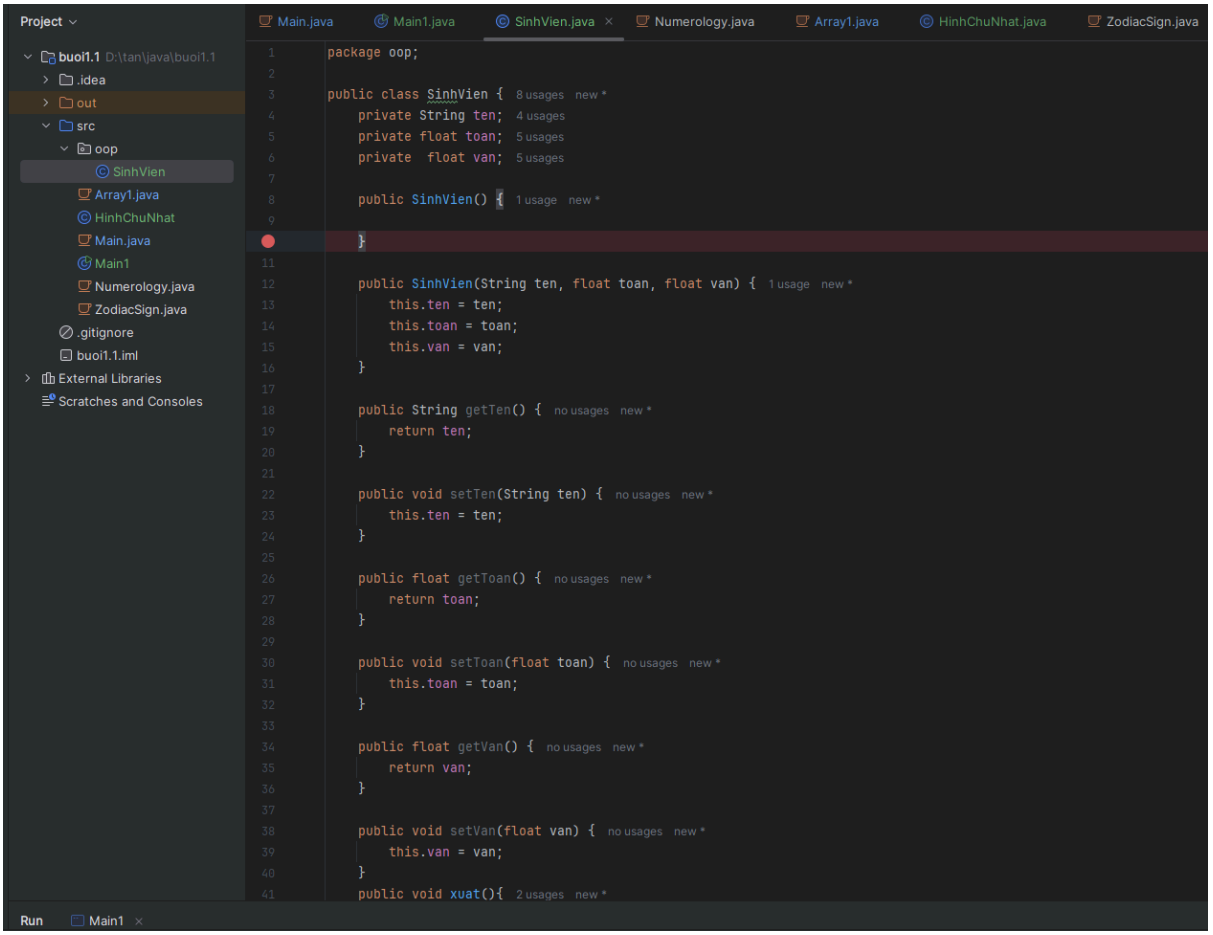
- **Khái niệm:** Trong các lớp, từ khóa `default` không phải là một modifier cụ thể. Thay vào đó, nó là một từ khóa để chỉ định phạm vi truy cập của lớp hoặc thành viên lớp. Trong ngữ cảnh này, `default` không được dùng như một modifier mà là

mặc định nếu không chỉ định phạm vi cụ thể như `public`, `protected`, hay `private`.

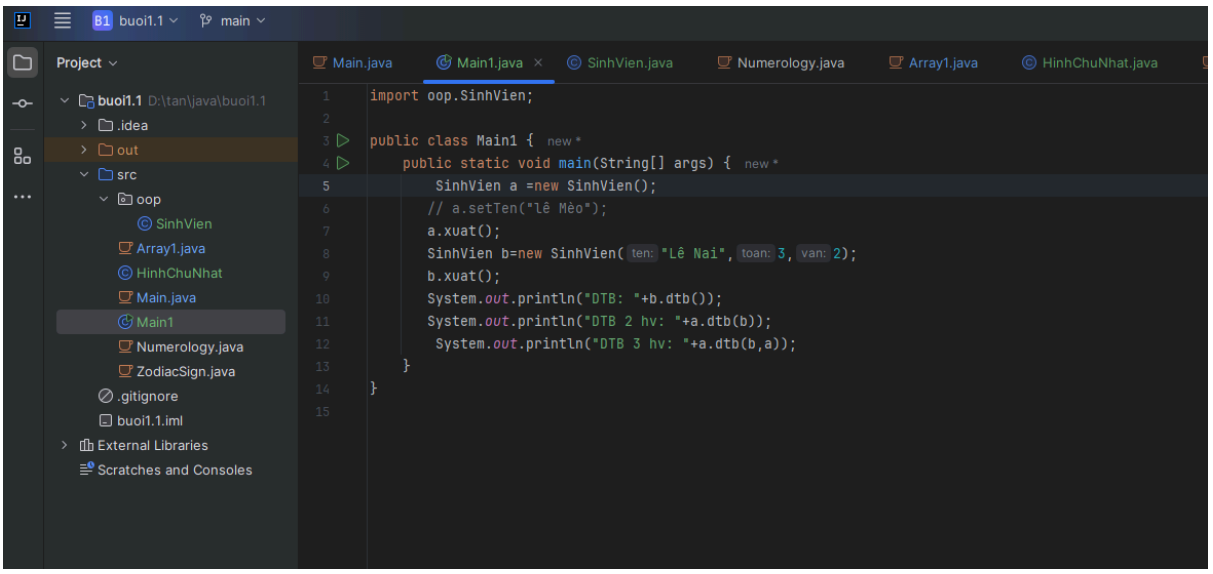
- **Cú pháp:**
  - Trong lớp, nếu không có từ khóa cụ thể, các thành viên của lớp (như phương thức và biến) mặc định là `package-private` (có thể truy cập từ cùng một package).

Ví dụ:

```
class MyClass {

    void defaultMethod() {

        System.out.println("This method has
default access");

    }

}



public class TestDefaultAccess {

    public static void main(String[] args) {

        MyClass obj = new MyClass();

        obj.defaultMethod(); // Output: This
method has default access

    }

    }
```

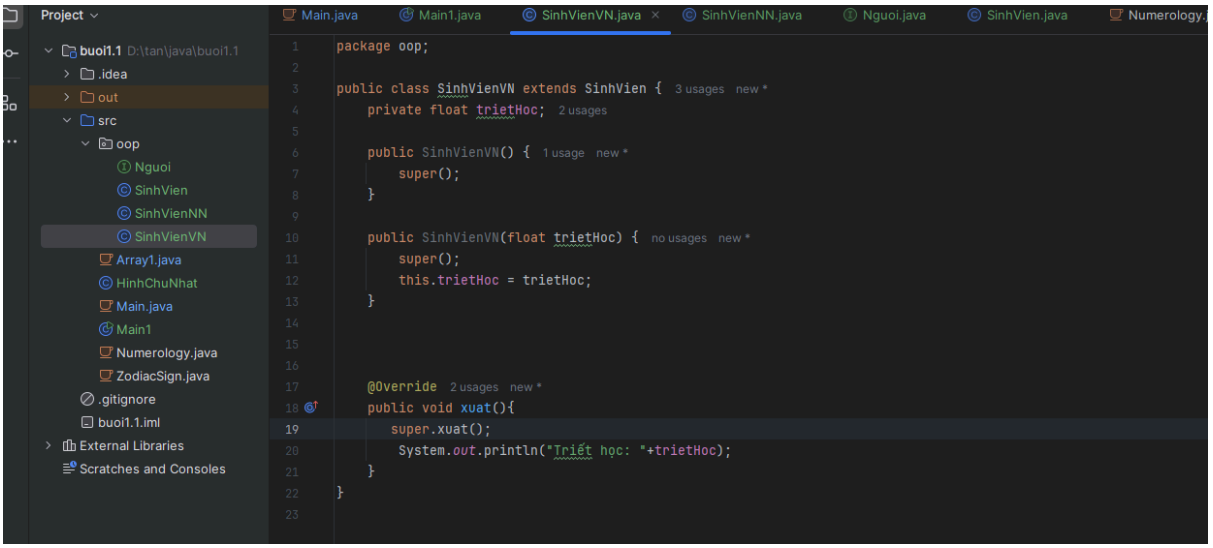


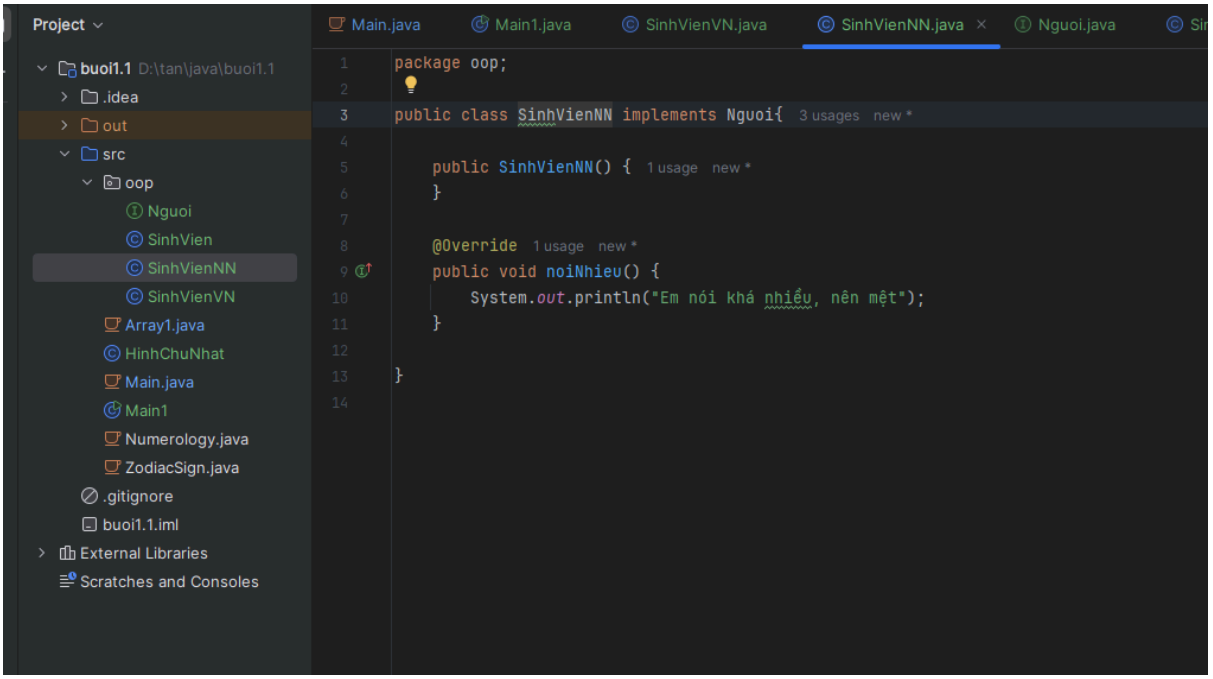

```java
package oop;

public class SinhVien {
    private String ten;
    private float toan;
    private  float van;

    public SinhVien() {

    }

    public SinhVien(String ten, float toan, float van) {
        this.ten = ten;
        this.toan = toan;
        this.van = van;
    }

    public String getTen() {
        return ten;
    }

    public void setTen(String ten) {
        this.ten = ten;
    }

    public float getToan() {
        return toan;
    }

    public void setToan(float toan) {
        this.toan = toan;
    }

    public float getVan() {
        return van;
    }

    public void setVan(float van) {
        this.van = van;
    }
    public void xuat(){
        System.out.println("Họ tên: "+this.ten+" -- Toan: "+this.toan+ " -- Van: "+this.van);
    }
    //Overloading
    public  float dtb(){
        return (this.toan+this.van)/2;
    }

    // Viết phương thức tính điểm trung bình 2 học viên
    public float dtb(SinhVien a) {
        return (this.dtb()+a.dtb()) / 2;
    }

    // Viết phương thức tính điểm trung bình 3 học viên
    public float dtb(SinhVien a,SinhVien b) {
        return (this.dtb()+a.dtb() +b.dtb()) / 3;
    }
}
```



Java không hỗ trợ đa kế thừa qua các lớp, để tránh các vấn đề phức tạp như "Diamond Problem".

Java hỗ trợ đa kế thừa thông qua giao diện, cho phép một lớp triển khai nhiều giao diện và cung cấp cài đặt cho các phương thức của các giao diện đó.



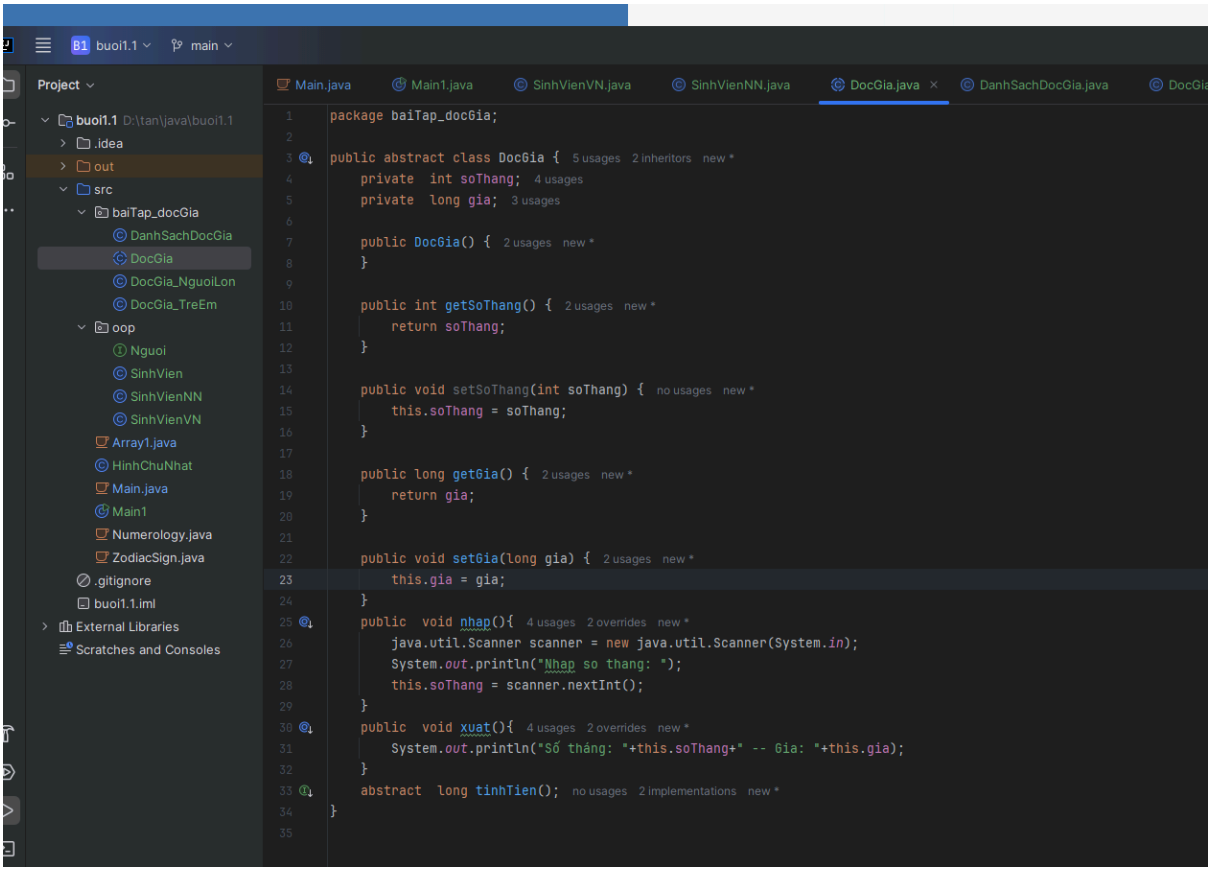


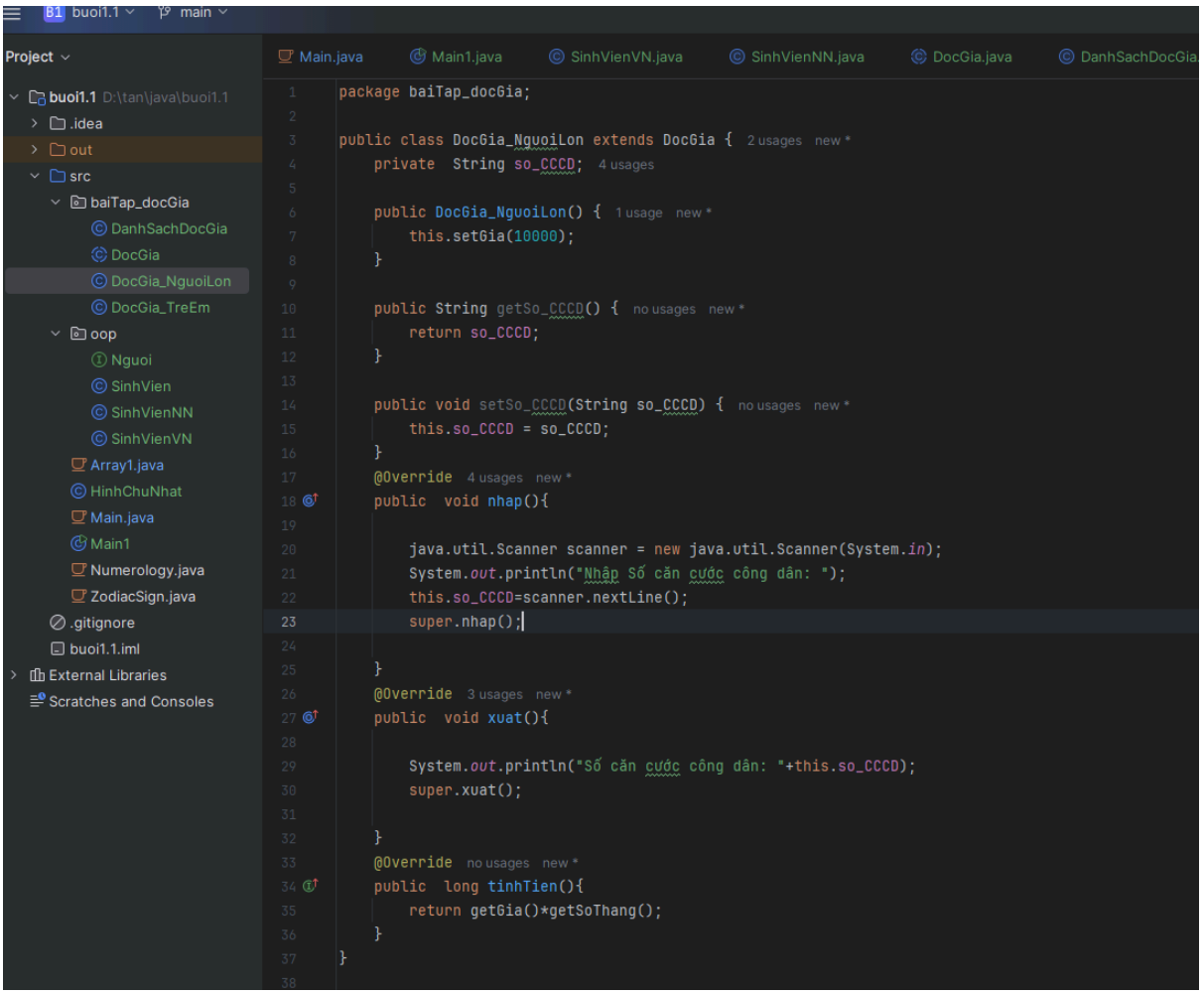
// Thông tin độc giả trẻ em gồm: Họ tên người bảo hộ, số tháng, giá (5k/1 tháng)

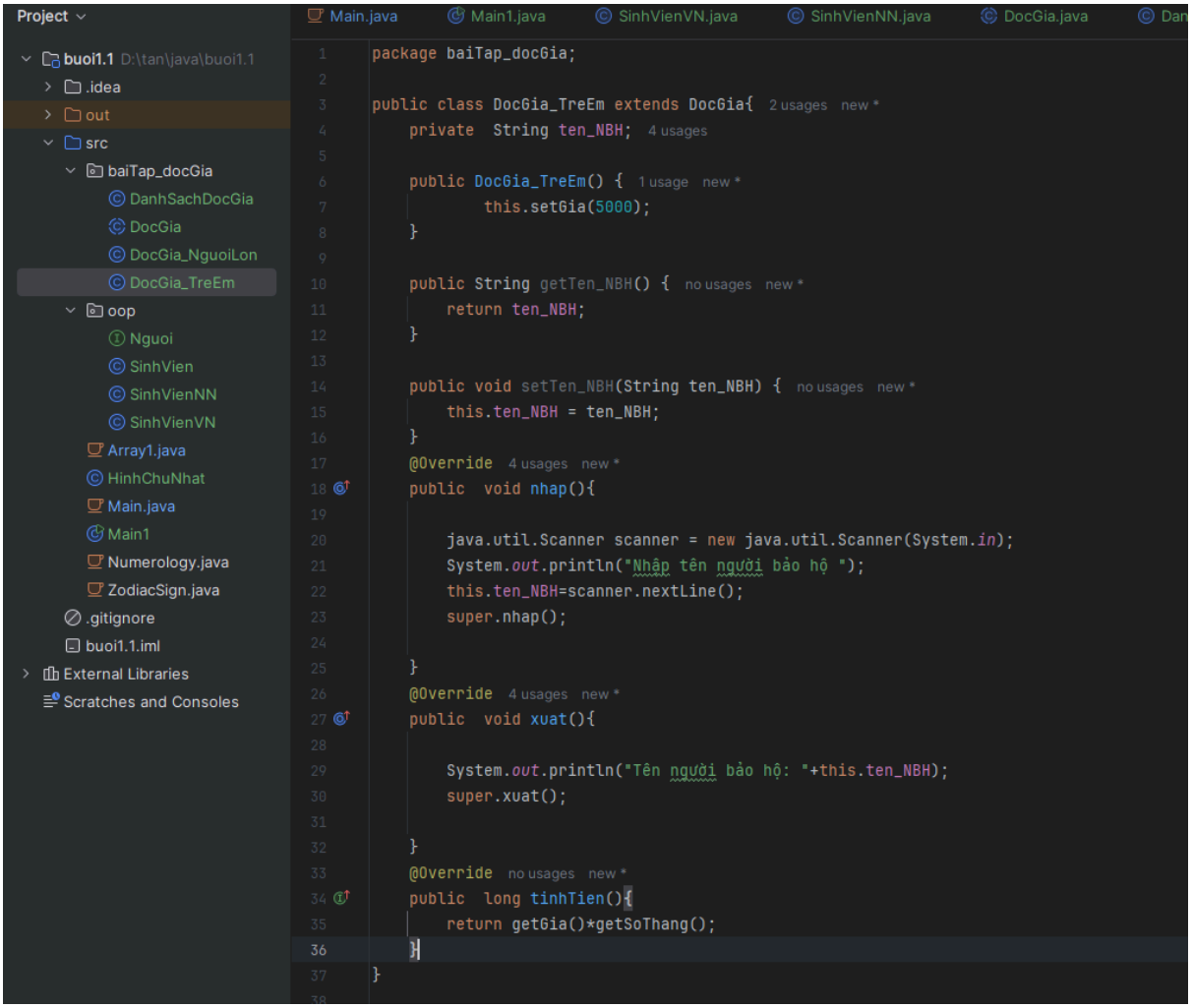
// Thông tin độc giả người lớn gồm: số căn cước công dân, số tháng, giá (10k / tháng)

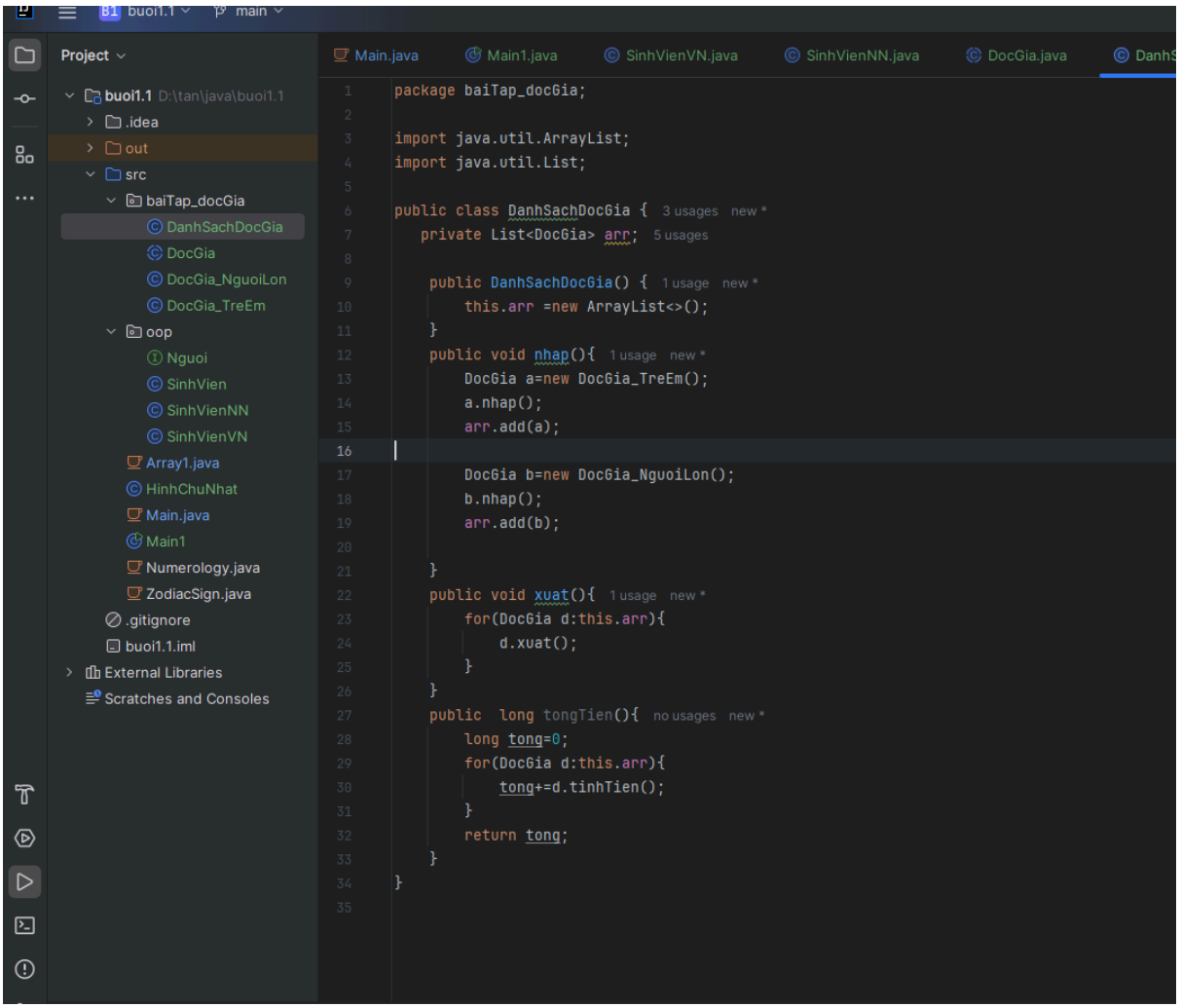
// a. Tạo class độc giả trẻ em, độc giả ngươi lớn, danh sách các độc giả

// b.Viết các phương thức nhập, xuất, tính tiền =số tháng * giá


buoi1.1
main
Project
Main.java
Main1.java
SinhVienVN.java
SinhVienNN.java
DocGia.java
DanhSachDocGia.java
buoi1.1 D:\tan\java\buoi1.1
.idea
out
src
baiTap_docGia
DanhSachDocGia
DocGia
DocGia_NguoiLon
DocGia_TreEm
oop
Nguoi
SinhVien
SinhVienNN
SinhVienVN
Array1.java
HinhChuNhat
Main.java
Main1
Numerology.java
ZodiacSign.java
.gitignore
buoi1.1.iml
External Libraries
Scratches and Consoles
package baiTap_docGia;
public abstract class DocGia { 5 usages 2 inheritors new *
private int soThang; 4 usages
private long gia; 3 usages
public DocGia() { 2 usages new *
}
public int getSoThang() { 2 usages new *
return soThang;
}
public void setSoThang(int soThang) { no usages new *
this.soThang = soThang;
}
public long getGia() { 2 usages new *
return gia;
}
public void setGia(long gia) { 2 usages new *
this.gia = gia;
}
public void nhap(){ 4 usages 2 overrides new *
java.util.Scanner scanner = new java.util.Scanner(System.in);
System.out.println("Nhap so thang: ");
this.soThang = scanner.nextInt();
}
public void xuat(){ 4 usages 2 overrides new *
System.out.println("Số tháng: "+this.soThang+" -- Gia: "+this.gia);
}
abstract long tinhTien(); no usages 2 implementations new *
}


buoi1.1
main
Project
buoi1.1 D:\tan\java\buoi1.1
.idea
out
src
baiTap_docGia
DanhSachDocGia
DocGia
DocGia_NguoiLon
DocGia_TreEm
oop
Nguoi
SinhVien
SinhVienNN
SinhVienVN
Array1.java
HinhChuNhat
Main.java
Main1
Numerology.java
ZodiacSign.java
.gitignore
buoi1.1.iml
External Libraries
Scratches and Consoles
Main.java
Main1.java
SinhVienVN.java
SinhVienNN.java
DocGia.java
DanhSachDocGia
package baiTap_docGia;
public class DocGia_NguoiLon extends DocGia { 2 usages new *
private String so_CCCD; 4 usages
public DocGia_NguoiLon() { 1 usage new *
this.setGia(10000);
}
public String getSo_CCCD() { no usages new *
return so_CCCD;
}
public void setSo_CCCD(String so_CCCD) { no usages new *
this.so_CCCD = so_CCCD;
}
@Override 4 usages new *
public void nhap(){
java.util.Scanner scanner = new java.util.Scanner(System.in);
System.out.println("Nhập Số căn cước công dân: ");
this.so_CCCD=scanner.nextLine();
super.nhap();
}
@Override 3 usages new *
public void xuat(){
System.out.println("Số căn cước công dân: "+this.so_CCCD);
super.xuat();
}
@Override no usages new *
public long tinhTien(){
return getGia()*getSoThang();
}
}


Project
buoi1.1 D:\tan\java\buoi1.1
.idea
out
src
baiTap_docGia
DanhSachDocGia
DocGia
DocGia_NguoiLon
DocGia_TreEm
oop
Nguoi
SinhVien
SinhVienNN
SinhVienVN
Array1.java
HinhChuNhat
Main.java
Main1
Numerology.java
ZodiacSign.java
.gitignore
buoi1.1.iml
External Libraries
Scratches and Consoles
Main.java
Main1.java
SinhVienVN.java
SinhVienNN.java
DocGia.java
package baiTap_docGia;
public class DocGia_TreEm extends DocGia{ 2 usages new *
private String ten_NBH; 4 usages
public DocGia_TreEm() { 1 usage new *
this.setGia(5000);
}
public String getTen_NBH() { no usages new *
return ten_NBH;
}
public void setTen_NBH(String ten_NBH) { no usages new *
this.ten_NBH = ten_NBH;
}
@Override 4 usages new *
public void nhap(){
java.util.Scanner scanner = new java.util.Scanner(System.in);
System.out.println("Nhập tên người bảo hộ ");
this.ten_NBH=scanner.nextLine();
super.nhap();
}
@Override 4 usages new *
public void xuat(){
System.out.println("Tên người bảo hộ: "+this.ten_NBH);
super.xuat();
}
@Override no usages new *
public long tinhTien(){
return getGia()*getSoThang();
}
}




```
DanhSachDocGia a=new DanhSachDocGia();
        a.nhap();
        a.xuat();
        System.out.println("Tổng tiền = "+a.tongTien());
```

Link topCIT:

https://drive.google.com/drive/folders/16edYHm2tonKQOSrET3VQLb30lYour5R3?usp=sharing

```
}
public void nhap(){  1 usage  new *
    int flag=1;
    System.out.println("1:Nhập DG trẻ em");
    System.out.println("2:Nhập DG người lớn");
    System.out.println("3:Exit");
    System.out.println("#########################");

    while(true){
        System.out.println("Mời bạn nhập lựa chon");
        java.util.Scanner scanner = new java.util.Scanner(System.in);
        flag = scanner.nextInt();
        DocGia a;
        if(flag==1){
            a=new DocGia_TreEm();
        }
        else  if(flag==2){
            a=new DocGia_NguoiLon();
        }
        else{
            break;
        }
        a.nhap();
        arr.add(a);
    }
}
```

Bài tập:

Cho biết có bao nhiêu độc giả trẻ em trong danh sách trên.

```
1   package oop;
2
3   public interface Nguoi {  1 usage  1 implementation  new *
4       void noiNhieu();  no usages  1 implementation  new *
5       default  void dungDienThoaiTrongLop(){  no usages  new *
6           System.out.println("1 vài học viên có dấu hiệu lạm dụng điện thoại quá mức rtong lớp học");
7       }
8   }
9
```

- **Abstract class** phù hợp khi bạn cần một cấu trúc lớp với một số phương thức cơ bản và có thể kế thừa một cách cụ thể.
- **Interface** phù hợp khi bạn cần định nghĩa một tập hợp các hành vi mà các lớp không liên quan đến nhau có thể thực thi, hỗ trợ khả năng mở rộng và linh hoạt hơn trong việc triển khai nhiều hành vi khác nhau.