
Tính đóng gói (Encapsulation)

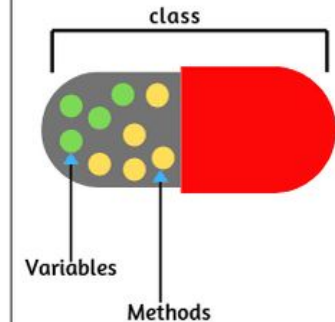
Tính đóng gói

Tính đóng gói là một trong 4 nguyên lý cơ bản của OOP

Đóng gói trong Java là một cơ chế gói biến và phương thức lại với nhau thành một đơn vị duy nhất

Với tính đóng gói, các biến của một lớp sẽ bị ẩn khỏi các lớp khác và chỉ có thể truy cập thông qua các phương thức của lớp hiện tại của chúng

```
class
{
    data members
    +
    methods (behavior)
}
```

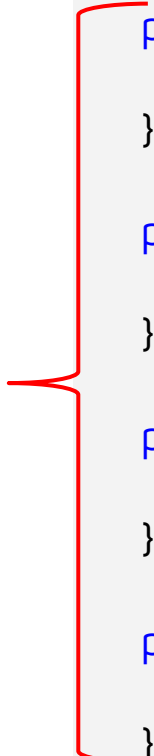


Để đạt được tính đóng gói trong Java, ta cần phải:

Khai báo các biến của một lớp là private

Cung cấp các phương thức setter và getter để sửa đổi và xem các giá trị của biến

Các phương
thức setter
và getter



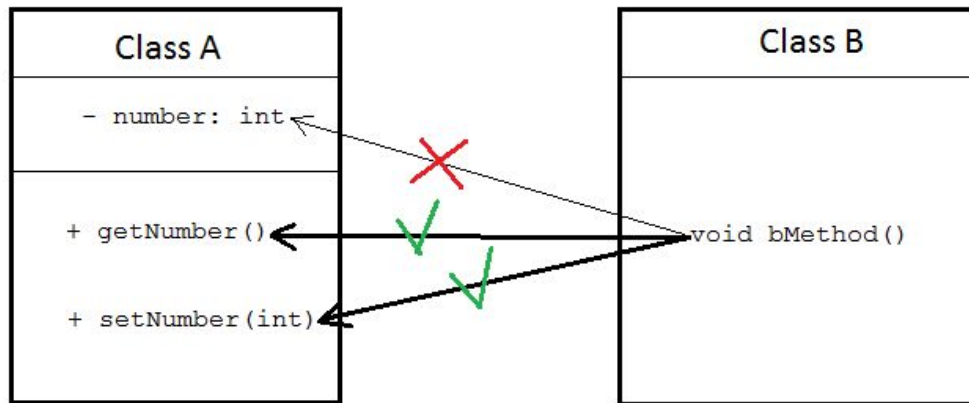
```
public class Person {  
    private String name;  
    private int age;  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public int getAge() {  
        return age;  
    }  
  
    public void setAge(int age) {  
        this.age = age;  
    }  
}
```

Tính chất đóng gói có những đặc điểm sau:

- Tạo ra cơ chế ngăn ngừa việc gọi phương thức của lớp này hay truy xuất dữ liệu của đối tượng thuộc về lớp khác
- Dữ liệu riêng của mỗi đối tượng được bảo vệ khỏi sự truy xuất không hợp lệ từ bên ngoài
- Người lập trình có thể dựa vào cơ chế này để ngăn ngừa sự gán giá trị không hợp lệ vào thành phần dữ liệu của mỗi đối tượng
- Cho phép thay đổi cấu trúc bên trong của mỗi lớp mà không làm ảnh hưởng đến những lớp bên ngoài có sử dụng lớp đó

Getter & Setter

Getter và Setter là hai phương thức sử dụng để lấy ra hoặc cập nhật giá trị thuộc tính, đặc biệt dành cho các thuộc tính ở phạm vi private



Getter



Phương thức *Getter* là phương thức truy cập vào thuộc tính của đối tượng và trả về các thuộc tính của đối tượng

Cú pháp:

```
public <Kiểu dữ liệu trả về> get<Tên thuộc tính>() {  
    return <Tên thuộc tính>;  
}
```

Ví dụ:

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

```
public int getAge() {  
    return age;  
}
```

```
System.out.println(person.getName()+" , "+ person.getAge());
```

Setter



Phương thức *Setter* là phương thức truy cập vào thuộc tính của đối tượng và gán giá trị cho các thuộc tính của đối tượng đó

Cú pháp:

```
public void set<Tên thuộc tính>(<Tham số giá trị mới>) {  
    this.<Tên thuộc tính> = <Tham số giá trị mới>;  
}
```


Ví dụ:

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

```
public void setAge(int age) {  
    this.age = age;  
}
```

```
person.setName("Ngoc");  
person.setAge(25);
```

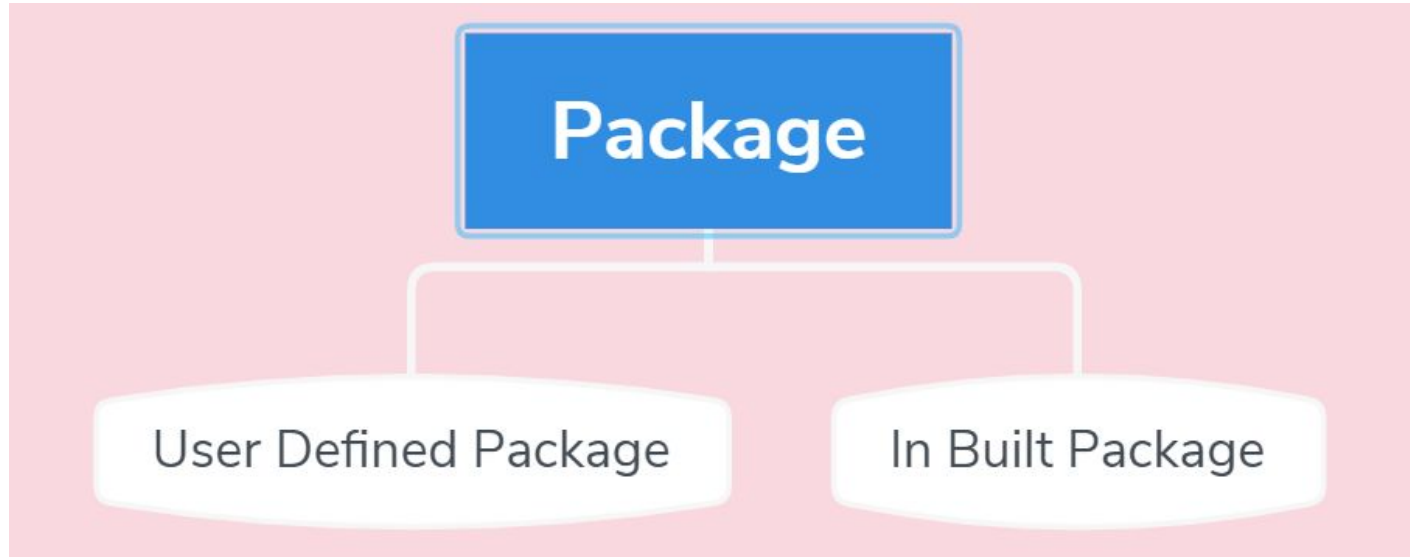
Package

Một package trong java được sử dụng để nhóm các lớp liên quan. Ta có thể coi nó như một thư mục.

Việc sử dụng package nhằm tránh xung đột về tên và code có thể dễ dàng bảo trì hơn



Package được chia làm 2 loại:



Built – in Package

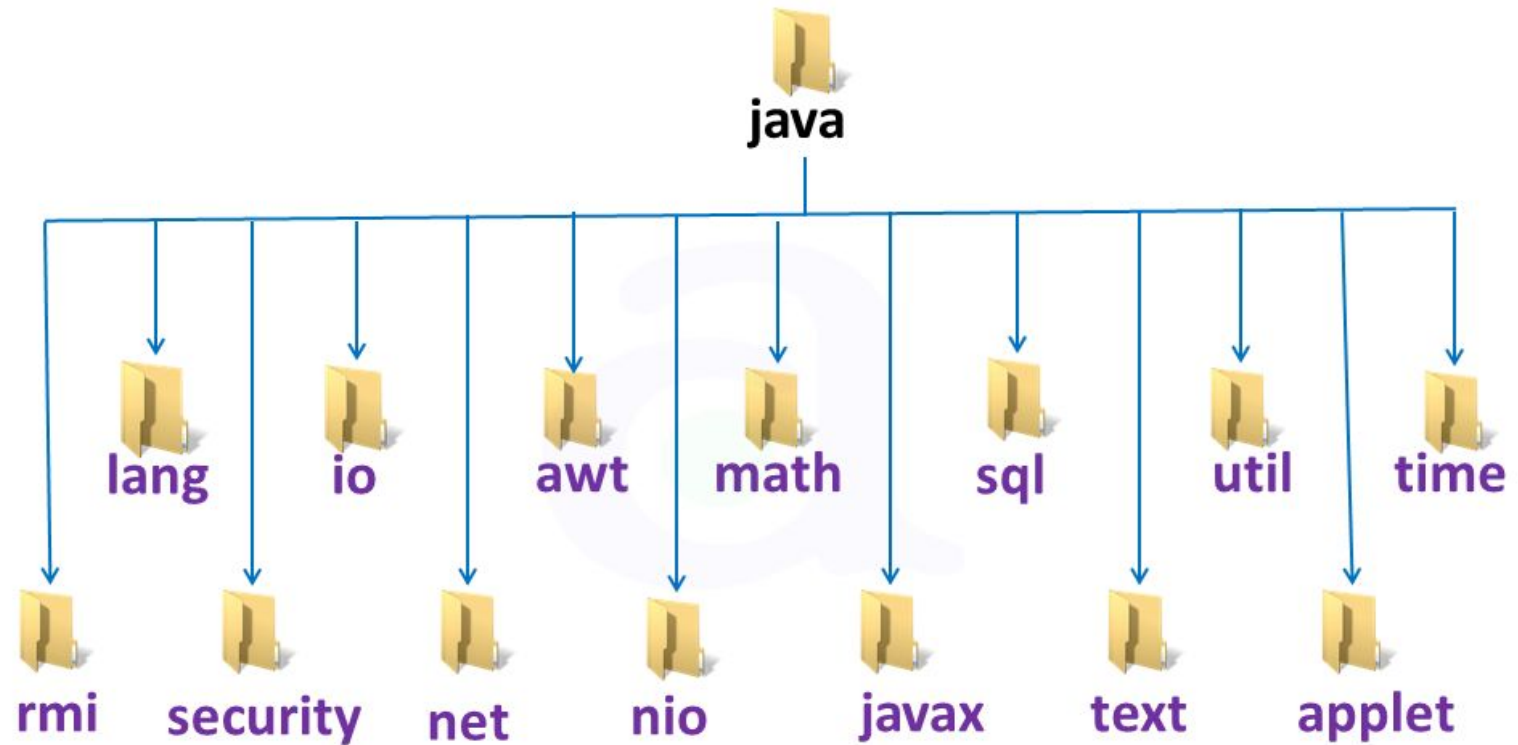


Java API là một thư viện các lớp được viết sẵn, được sử dụng miễn phí.

Thư viện này chứa các thành phần để quản lý đầu vào, cơ sở dữ liệu, ... Có thể tham khảo tại <https://docs.oracle.com/javase/8/docs/api/>

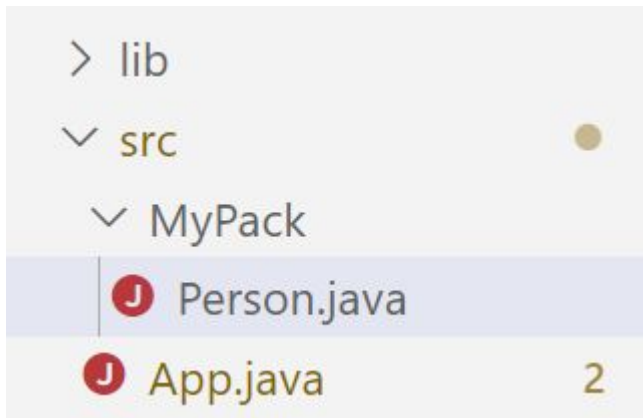
Thư viện này được chia thành các package và các lớp. Có nghĩa là ta có thể truy cập một lớp duy nhất hoặc toàn bộ package chứa tất cả các lớp thuộc package đó. Ví dụ:

```
import java.util.Scanner; //Truy cập lớp Scanner
import java.util.*; //Truy cập vào package
```



User – defined Package

Để tạo package của riêng bạn, bạn cần hiểu rằng Java sử dụng một thư mục hệ thống tệp để lưu trữ chúng. Cũng giống như các thư mục trên máy tính của bạn



Để tạo package, ta sử dụng từ khóa *package*:

```
package MyPack;

public class Person {
    private String name;
    private int age;
}
```

```
import MyPack.*; //hoặc import.MyPack.Person;

public class App {
    public static void main(String[] args) {
        //TODO
    }
}
```

Access Modifiers



	public	private	protected	default
class	Allowed	Not allowed	Not allowed	Allowed
constructor	Allowed	Allowed	Allowed	Allowed
variable	Allowed	Allowed	Allowed	Allowed
method	Allowed	Allowed	Allowed	Allowed


```
public class Person  
class Person
```



```
private class Person  
protected class Person
```



	class	subclass	package	outside
private	Allowed	Not allowed	Not allowed	Not allowed
protected	Allowed	Allowed	Allowed	Not allowed
public	Allowed	Allowed	Allowed	Allowed
default	Allowed	Not allowed	Allowed	Not allowed

Tính kế thừa (Inheritance)

Inheritance (IS-A)

Tính kế thừa là một trong 4 nguyên lý cơ bản của OOP

Trong Java, kế thừa đề cập đến việc tất cả các thuộc tính và phương thức của lớp này có thể kế thừa bởi một lớp khác.

Với kế thừa ta có thêm 2 thuật ngữ:

- ☐ Subclass: Lớp con
- ☐ Superclass: Lớp cha



Để kế thừa từ một lớp, ta sử dụng từ khóa *extends*

```
public class Animal {  
    public void eat(){  
        System.out.println("Eating...");  
    }  
}
```

Lớp cha

Sử dụng để thể
hiện sự kế thừa

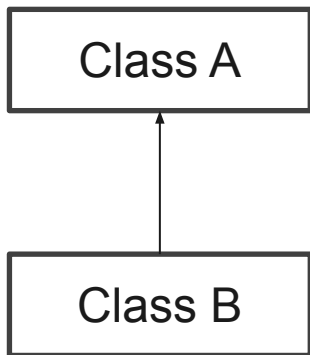
```
public class Cat extends Animal {  
    public void sound(){  
        System.out.println("Meow...");  
    }  
}
```

Lớp con

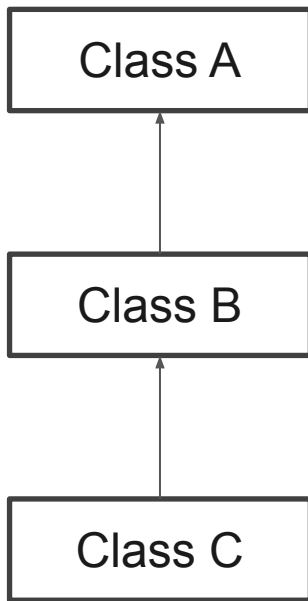
```
public class Main {  
    public static void main(String[] args) {  
        Cat cat = new Cat();  
        cat.eat();  
        cat.sound();  
    }  
}
```

Truy cập tới phương thức của lớp cha

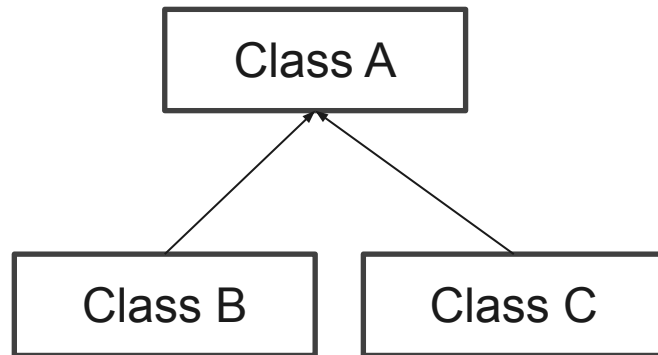
Các loại kế thừa trong java



Đơn kế thừa



Kế thừa nhiều cấp



Kế thừa thứ bậc

Đơn kế thừa

```
public class ClassA {  
  
}
```

Class A là superclass
Class B kế thừa Class A

```
public class ClassB extends ClassA {  
  
}
```


Kế thừa nhiều cấp

Class A là superclass
Class B kế thừa ClassA
ClassC kế thừa ClassB

```
public class ClassA {  
  
}
```

```
public class ClassB extends ClassA {  
  
}
```

```
public class ClassC extends ClassB {  
  
}
```

Kế thừa thứ bậc

Class A là superclass
Cả ClassB và ClassC
đều kế thừa từ ClassA

```
public class ClassA {  
  
}
```

```
public class ClassB extends ClassA {  
  
}
```

```
public class ClassC extends ClassA {  
  
}
```

Ngoài ra ta còn đa kế thừa, tuy nhiên đa kế thừa không được hỗ trợ thông qua lớp mà chỉ được hỗ trợ thông qua interface

```
public class ClassA {  
    void msg(){System.out.println("Hello!!!");}  
}
```

```
public class ClassB {  
    void msg(){System.out.println("Hi!!!");}  
}
```

```
public class ClassC extends ClassA, ClassB {  
    public static void main(String[] args) {  
        C c = new C();  
        c.msg();  
    }  
}
```

Không biết phương thức của lớp nào được gọi

Viết chương trình quản lý thư viện. Thực hiện các yêu cầu sau:

Tạo class Library chứa:

Các thuộc tính: Mã sách, tên sách, nhà xuất bản, năm xuất bản, số lượng

Các phương thức để nhập và xuất thông tin.

Tạo class SchoolBook kế thừa class Library chứa:

Các thuộc tính: Mã sách, tên sách, nhà xuất bản, năm xuất bản, số lượng, số trang, tình trạng, số lượng mượn.

Các phương thức để nhập xuất thông tin, phương thức đưa ra vị trí và tồn kho



Aggregation (HAS-A)

Nếu một lớp có một tham chiếu thực thể, thì nó được biết đến như là một lớp có quan hệ HAS-A

Sử dụng quan hệ HAS-A giúp làm tăng tính tái sử dụng của code. Và khi không có mối quan hệ IS-A, thì quan hệ HAS-A là lựa chọn tốt nhất.

```
public class Address {  
    String district, city, country;  
  
    public Address(String district, String city, String country) {  
        this.district = district;  
        this.city = city;  
        this.country = country;  
    }  
}
```

```
public class Person {
```

```
    String name;
```

```
    int age;
```

```
    Address address;
```

Address là một lớp



```
    public Person(String name, int age, Address address) {
```

```
        this.name = name;
```

```
        this.age = age;
```

```
        this.address = address;
```

```
    }
```

```
    public void display(){
```

```
        System.out.print(name + " - " + age + " - ");
```

```
        System.out.print(address.district + ", " + address.city + ", " + address.country);
```

```
    }
```

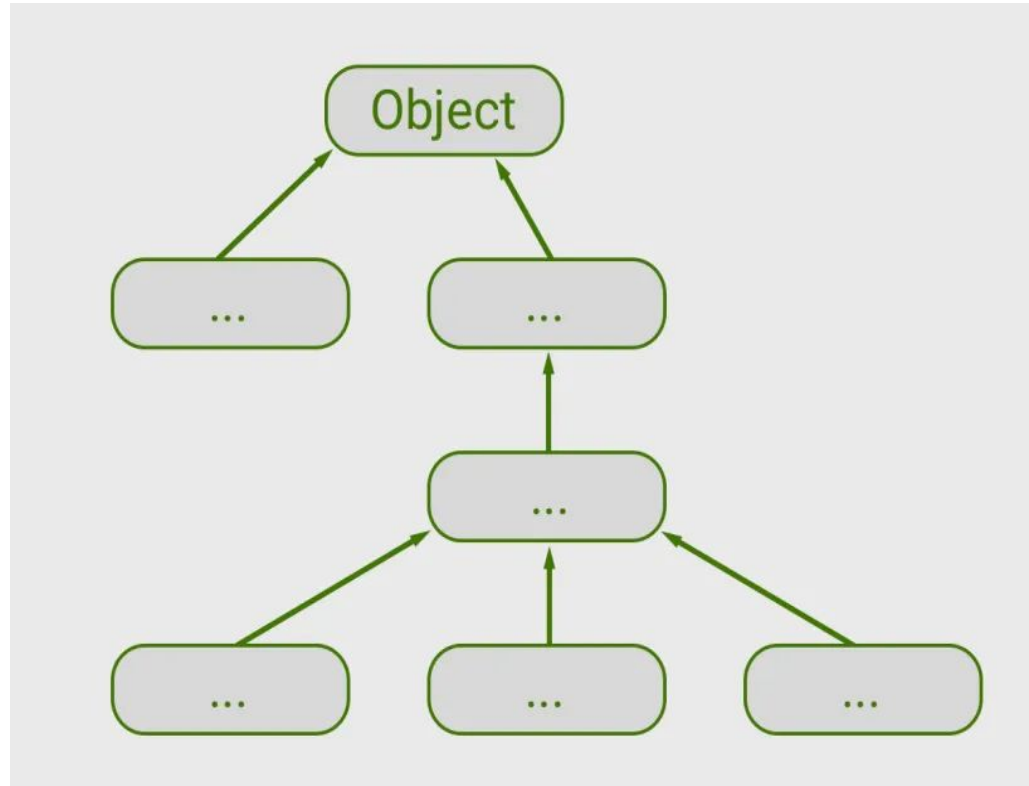
```
}
```

```
public static void main(String[] args) throws Exception {  
    Address address = new Address("Nam Tu Liem", "Ha Noi", "Viet Nam  
");  
    Person person = new Person("Ngoc", 25, address);  
  
    person.display();  
}
```

Ngoc - 25 - Nam Tu Liem, Ha Noi, Viet Nam

Lớp Object

Lớp Object là lớp cha của tất cả các lớp trong Java



Phương thức	Miêu tả
<code>public final Class getClass()</code>	Trả về đối tượng lớp Class của đối tượng này. Lớp Class có thể được sử dụng để lấy metadata của lớp này
<code>public int hashCode()</code>	Trả về hashcode cho đối tượng này
<code>public boolean equals(Object obj)</code>	So sánh đối tượng đã cho với đối tượng này
<code>protected Object clone() throws CloneNotSupportedException</code>	Tạo và trả về bản sao (bản mô phỏng) của đối tượng này
<code>public String toString()</code>	Trả về biểu diễn chuỗi của đối tượng này
<code>public final void notify()</code>	Thông báo Thread đơn, đợi trên monitor của đối tượng này
<code>public final void notifyAll()</code>	Thông báo tất cả Thread, đợi trên monitor của đối tượng này

<code>public final void wait(long timeout)throws InterruptedException</code>	Làm cho Thread hiện tại đợi trong khoảng thời gian là số mili giây cụ thể, tới khi Thread khác thông báo (triệu hồi phương thức <code>notify()</code> hoặc <code>notifyAll()</code>)
<code>public final void wait(long timeout,int nanos)throws InterruptedException</code>	Làm cho Thread hiện tại đợi trong khoảng thời gian là số mili giây và nano giây cụ thể, tới khi Thread khác thông báo (triệu hồi phương thức <code>notify()</code> hoặc <code>notifyAll()</code>)
<code>public final void wait()throws InterruptedException</code>	Làm Thread hiện tại đợi, tới khi Thread khác thông báo (<code>invoke notify()</code> or <code>notifyAll()</code> method).
<code>protected void finalize()throws Throwable</code>	Được triệu hồi bởi Garbage Collector trước khi đối tượng bị dọn rác

Kiểm tra một đối tượng có phải là thể
hiện của một kiểu dữ liệu cụ thể không

```
@Override
public boolean equals(Object obj) {
    if(obj instanceof Person){
        if((((Person)obj).name.equals(this.name)){
            return true;
        }
    }
    return false;
}
```

So sánh dựa vào tên

```
public class Person implements Cloneable {
```



Implements interface
Cloneable

```
    @Override  
    protected Object clone() throws CloneNotSupportedException {  
        // TODO Auto-generated method stub  
        return super.clone();  
    }  
}
```

```
public static void main(String[] args) throws Exception {  
    Person person = new Person("Ngoc", 25, "Ha Noi");  
    person.display();  
  
    Person person3 = (Person) person.clone();  
    person3.display();  
}
```

Viết chương trình quản lý trường học.

