

LỚP VÀ ĐỐI TƯỢNG

Nội dung chính

- 1. Lớp và đối tượng**
- 2. Ký hiệu UML**
- 3. Gói**
- 4. Quan hệ lớp và đối tượng**
- 5. Quan hệ hai lớp**
- 6. Lớp trong (Inner Class)**
- 7. Lớp không đổi (Immutable Class)**
- 8. Lớp ArrayList**

Lớp

- Lớp (class) là **khuôn mẫu** (template) định nghĩa các **thuộc tính** (properties) và **hành vi** (behaviors) của các thực thể.
- Cú pháp khai báo lớp trong Java

```
[public] class TênLớp {  
    [Các thuộc tính]  
    [Các phương thức]  
}
```

Lớp

- Tên lớp thường là **danh từ**, ngắn gọn, có ý nghĩa và **không trùng** với các từ khoá.
- Tập tin mã nguồn *.java có thể chứa nhiều lớp, nhưng chỉ chứa **một** lớp public cùng tên với tên tập tin.

Lớp

- Tên lớp đặt theo quy tắc **UpperCamelCase**.
 - Ví dụ: **SinhVien**, **NhanVien**, **HoaDon**

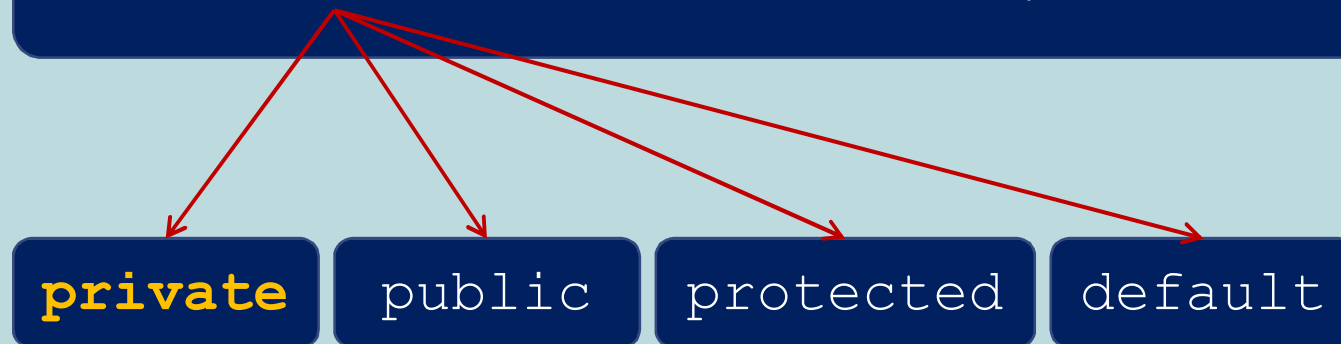


Nguồn: wikipedia

Thuộc tính

- Các thuộc tính của lớp dùng **lưu trữ** thông tin của thực thể.
- Cú pháp khai báo thuộc tính trong Java

```
[AccessModifier] KiểuDữLiệu tênThuộcTính;
```



Thuộc tính

- Ví dụ

```
public class SinhVien {  
    private String maSo;  
    private String hoTen;  
    private boolean daTotNghiep;  
}
```

```
public class PhanSo {  
    private int tuSo;  
    private int mauSo;  
}
```

Phương thức

- Các phương thức thể hiện **hành vi** thực thể.
- Các loại phương thức



Phương thức

- Cú pháp khai báo phương thức

```
[AccessModifier] KiểuDữLiệu tênPhươngThức ([ThamSố]) ;
```

private

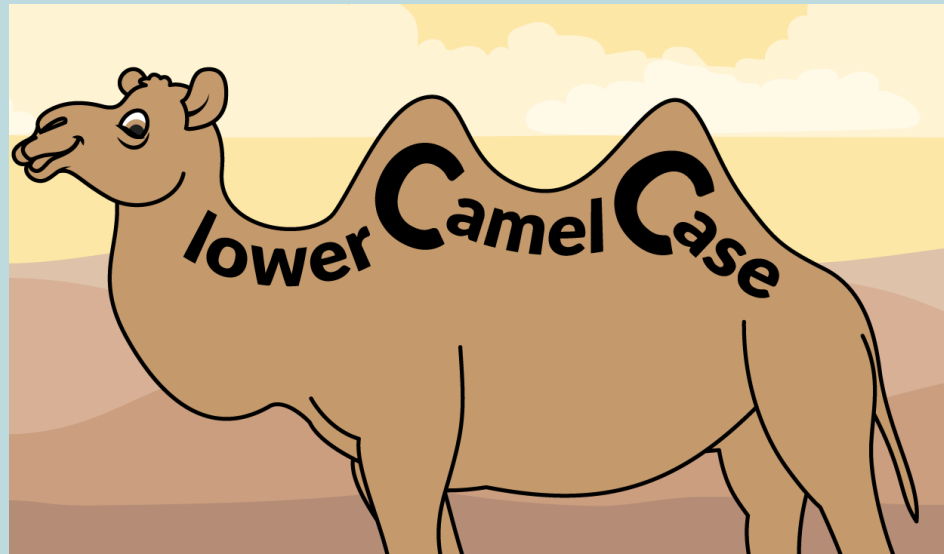
public

protected

default

Phương thức

- Tên thuộc tính và phương thức nên đặt theo quy tắc **lowerCamelCase**.
 - Ví dụ: **rutGon()**, **hienThi()**, **soLuong**, **heSo**.



Nguồn: reviewwalls

Phương thức

```
public class PhanSo {  
    ...  
    public PhanSo rutGon() {  
  
    }  
    public PhanSo cong(PhanSo p) {  
  
    }  
    public void hienThi() {  
  
    }  
}
```

Phương thức getter và setter

- Phương thức cho phép **đọc** dữ liệu của một thuộc tính gọi là phương thức **getter**. Quý ước đặt tên
 - Thuộc tính kiểu bool: **isTênThuộcTính()**
 - Thuộc tính kiểu khác: **getTênThuộcTính()**
- Phương thức cho phép **ghi** dữ liệu vào một thuộc tính gọi là phương thức **setter**. Quý ước đặt tên:
 - **setTênThuộcTính([ThamSố])**
 - **[ThamSố]** thường là một tham số cùng kiểu dữ liệu với thuộc tính.

Phương thức getter và setter

```
class SinhVien {  
    private String hoTen;  
    private boolean daTotNghiep;  
    public String getHoTen() {  
        return hoTen;  
    }  
    public void setHoTen(String hoTen) {  
        this.hoTen = hoTen;  
    }  
    public boolean isDaTotNghiep() {  
        return daTotNghiep;  
    }  
    public void setDaTotNghiep(boolean daTotNghiep) {  
        this.daTotNghiep = daTotNghiep;  
    }  
}
```

Phương thức khởi tạo

- Phương thức khởi tạo (constructor) được gọi đầu tiên khi tạo đối tượng.
- Đặc điểm phương thức khởi tạo trong Java
 - Tên phương thức **trùng tên lớp**.
 - **Không có** kiểu dữ liệu trả về.
- Cú pháp

```
[AccessModifier] TênLớp ([ThamSố]) ;
```

Phương thức khởi tạo

```
public class PhanSo {  
    private int tuSo;  
    private int mauSo;  
    public PhanSo() {  
        tuSo = 0;  
        mauSo = 1;  
    }  
    public PhanSo(int tu, int mau) {  
        tuSo = tu;  
        mauSo = mau;  
    }  
}
```

Phương thức khởi tạo

- Một lớp có thể khai báo **không hoặc nhiều** phương thức khởi tạo.
- Nếu lớp **không khai báo tường minh** bất kỳ phương thức khởi tạo nào thì Java sẽ ngầm định tạo ra phương thức khởi tạo không tham số (default constructor).

Phát biểu nào sau đây sai?

- A. Phương thức khởi tạo không có kiểu dữ liệu trả về.**
- B. Phạm vi truy cập của phương thức khởi tạo phải là public.**
- C. Tên phương thức khởi tạo phải trùng với tên lớp.**
- D. Lớp bắt buộc phải khai báo ít nhất một phương thức khởi tạo.**
- E. Một lớp có thể có nhiều phương thức khởi tạo.**

Đối tượng

- Đối tượng là thực thể thực sự hoạt động trong hệ thống.
- Một đối tượng xác định duy nhất bởi
 - Định danh (**identity**): tên đối tượng.
 - Trạng thái (**state**): giá trị hiện tại các thuộc tính của đối tượng.
 - Hành vi (**behavior**): là các phương thức.

Đối tượng

- Cú pháp tạo đối tượng

```
TênLớp tênĐốiTượng = new PhươngThứcTạo ( [ThamSố] ) ;
```

- Truy cập các thuộc tính và phương thức

```
tênĐốiTượng.tênThuộcTính;  
tênĐốiTượng.tênPhươngThức ( [ĐốiSố] ) ;
```

Đối tượng

- Ví dụ

```
PhanSo p1 = new PhanSo(12, 8);  
PhanSo kq1 = p1.rutGon();  
kq1.hienThi();
```

```
PhanSo p2 = new phanSo(2, 3);  
PhanSo kq2 = p1.cong(p2);  
kq2.hienThi();
```

Đối tượng

▪ Tìm lỗi trong chương trình sau?

```
class Point {  
    private int x, y;  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}  
  
public class Example {  
    public static void main(String[] args) {  
        Point p1 = new Point();  
        Point p2 = new Point(2, 4);  
    }  
}
```

Đối tượng

▪ Cho lớp Person như sau

```
class Person {  
    private String firstName;  
    private String lastName;  
    public Person(String fn) {  
        this.firstName = fn;  
    }  
    public Person(String fn, String ln) {  
        this.firstName = fn;  
        this.lastName = ln;  
    }  
    // Các phương thức getter và setter  
}
```

Đối tượng

- Cho biết lệnh tạo đối tượng nào sau đây không hợp lệ?

```
Person p1 = new Person();  
Person p2 = new Person("");  
Person p3 = new Person(null);  
Person p4 = new Person("A");  
Person p5 = new Person("A", "B");  
Person p6 = new Person("A", "B", "C");
```

Đối tượng

- **Viết lớp PhanSo để đoạn chương trình sau xuất kết quả là "5/3" trong console?**

```
PhanSo p1 = new PhanSo(1);  
PhanSo p2 = new PhanSo(2, 3);  
PhanSo p = p1.cong(p2);  
p.hienThi();
```


Đối tượng

- Tìm lỗi đoạn chương trình sau và giải thích?

```
class Person {  
    private String name;  
    public Person(String n) {  
        n = name;  
    }  
    public String getName() {  
        return name;  
    }  
}
```

```
Person p1 = new Person("Thanh");  
Person p2 = new Person("THANH");  
System.out.println(p1.getName().equals(p2.getName()));
```

Truyền mảng vào phương thức

- Java luôn dùng **truyền trị** (pass-by-value) để truyền các đối số vào phương thức.
 - Nếu đối số có **kiểu dữ liệu cơ bản** thì giá trị của nó được truyền vào.
 - Nếu đối số có **kiểu tham chiếu** thì giá trị tham chiếu được truyền vào.
- Mảng thực chất là **biến tham chiếu** nên khi truyền nó vào phương thức thì giá trị tham chiếu của nó sẽ được truyền vào.

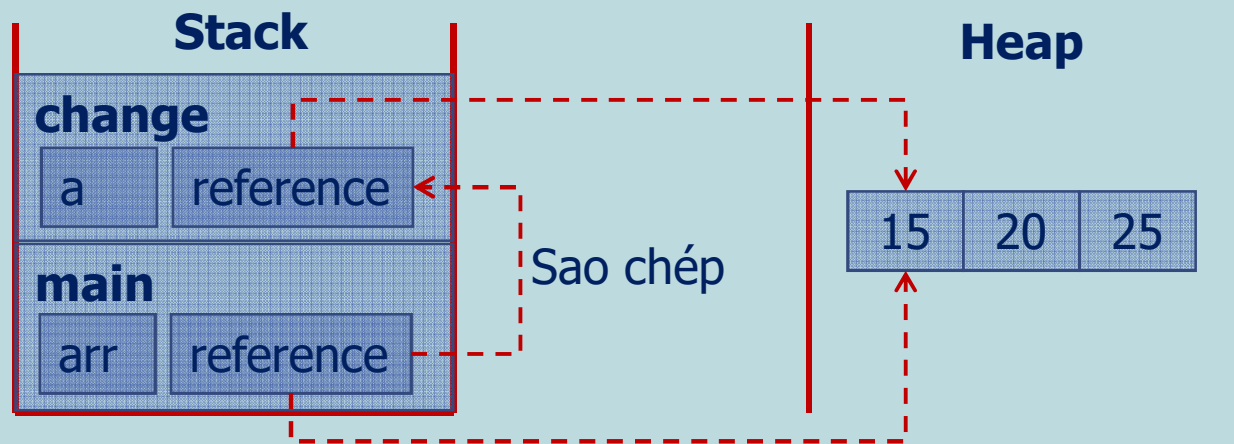
Truyền mảng vào phương thức

- Cho biết kết quả xuất ra màn hình đoạn chương trình?

```
public class Example {  
    public static void change(int[] a, int y) {  
        y = 999;  
        a[0] = 999;  
    }  
    public static void main(String[] args) {  
        int x = 10;  
        int[] arr = {15, 20, 25};  
        change(arr, x);  
        System.out.println(x);  
        System.out.println(arr[0]);  
    }  
}
```

Truyền mảng vào phương thức

```
public class Example {  
    public static void change(int[] a) {  
        ...  
    }  
    public static void main(String[] args) {  
        int[] arr = {15, 20, 25};  
        change(arr);  
        ...  
    }  
}
```



Truyền mảng vào phương thức

▪ Cho biết kết quả của đoạn chương trình?

```
public class Demo {  
    public static void change(int a[], int b) {  
        int[] t = {1, 2, 3};  
        a = t;  
        b = 5;  
    }  
    public static void main(String[] args) {  
        int[] a = {4, 5, 6};  
        int b = 10;  
        change(a, b);  
        System.out.println(a[0]);  
        System.out.println(b);  
    }  
}
```

Truyền mảng vào phương thức

- Cho biết kết quả của đoạn chương trình?

```
public class Demo {  
    public static void change(int a[]) {  
        a[0] = 9;  
        int[] t = {1, 2, 3};  
        a = t;  
        a[0] = 99;  
    }  
    public static void main(String[] args) {  
        int[] a = {4, 5, 6};  
        change(a);  
        System.out.println(a[0]);  
    }  
}
```

Tham chiếu this

- Mỗi phương thức ngầm định đối số đầu tiên là **this** trỏ đến đối tượng hiện tại.
- Ví dụ

```
public class PhanSo {  
    private int tuSo;  
    private int mauSo;  
    ...  
    public PhanSo(int tu, int mau) {  
        this.tuSo = tu;  
        this.mauSo = mau;  
    }  
}
```

Tham chiếu this

- Ví dụ thiết kế phương thức thực hiện phép cộng hai phân số.

```
public class PhanSo {  
    private int tuSo;  
    private int mauSo;  
    ...  
    public PhanSo Cong(PhanSo p) {  
        int tu = this.tuSo*p.mauSo+p.tuSo*this.mauSo;  
        int mau = this.mauSo*p.mauSo;  
  
        return new PhanSo(tu, mau);  
    }  
}
```


Tham chiếu this

- Trong Java cho phép sử dụng **this** để gọi phương thức khởi tạo trong một phương thức khởi tạo khác.

```
public class PhanSo {  
    private int tuSo, mauSo;  
    public PhanSo(int tu, int mau) {  
        this.tuSo = tu;  
        this.mauSo = mau;  
    }  
    public PhanSo() {  
        this(0, 1);  
    }  
}
```

Tham chiếu this

- Cho biết kết quả xuất ra màn hình đoạn chương trình?

```
class A {  
    private int x;  
    public A(int v) {  
        x += v;  
    }  
    public A() {  
        this(2);  
    }  
    public int getX() {  
        return x;  
    }  
}
```

```
A a1 = new A();  
A a2 = new A(3);  
System.out.println(a1.getX());  
System.out.println(a2.getX());
```

Tham chiếu this

- Cho biết kết quả xuất ra của các khối lệnh (1) và (2)

```
class A {  
    private int x, y;  
    private A(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    public A() {  
        this(6, 8);  
    }  
    public void hienThi() {  
        System.out.println(x + y);  
    }  
}
```

// (1)

```
A a1 = new A(2, 3);  
a1.hienThi();
```

// (2)

```
A a2 = new A();  
a2.hienThi();
```

Thuộc tính và phương thức tĩnh

- Thuộc tính tĩnh và phương thức tĩnh **dùng chung** cho tất cả đối tượng lớp.
- Chúng được gọi sử dụng thông qua tên lớp mà **không cần tạo đối tượng**.
- Trong phương thức tĩnh **chỉ truy xuất** được các thành viên tĩnh của lớp.
- Để khai báo thuộc tính hay phương thức tĩnh dùng từ khoá **static** trước kiểu dữ liệu.

Thuộc tính và phương thức tĩnh

- Ví dụ

```
class PhanSo {  
    private static int soLuongPs = 0;  
  
    public static int timUcLn(int a, int b) {  
        // ...  
    }  
}
```

- Ta gọi phương thức `timUcLn()` bằng cách

```
int ucln = PhanSo.timUcLn(12, 8);
```

Thuộc tính và phương thức tĩnh

- Cho biết kết quả của đoạn chương trình?

```
class PhanSo {  
    private static int soLuongPs = 0;  
    public PhanSo() {  
        soLuongPs++;  
    }  
    public void hienThi() {  
        System.out.print(soLuongPs);  
    }  
}
```

```
PhanSo p1 = new PhanSo();  
PhanSo p2 = new PhanSo();  
p2.hienThi();
```

Thuộc tính và phương thức tĩnh

- Cho biết kết quả của đoạn chương trình?

```
class A {  
    private int a, b;  
    public A(int a, int b) {  
        this.a = a;  
        this.b = b;  
    }  
    public static int cong() {  
        return this.a + this.b;  
    }  
}
```

```
A a = new A(2, 3);  
System.out.print(a.cong());
```

Thuộc tính và phương thức tĩnh

▪ Cho biết kết quả đoạn chương trình?

```
class Test {  
    private static int n;  
    public Test() {  
        n = 3;  
    }  
    public void inc() {  
        n++;  
    }  
    public static int getN() {  
        return n;  
    }  
}
```

```
Test t1 = new Test();  
t1.inc();  
t1.inc();  
Test t2 = new Test();  
t2.inc();  
System.out.print(Test.getN());
```


Initialization block

- Để khởi động giá trị thuộc tính có thể
 - hoặc gán lúc khai báo,
 - hoặc trong phương thức khởi tạo,
 - hoặc initialization block.
- **Initialization block:** khối này được gọi mỗi khi một đối tượng được tạo (**gọi trước cả các phương thức khởi tạo**).

Initialization block

▪ Ví dụ

```
class NhanVien {  
    private static int dem = 1;  
    private int mssv;  
    private String hoTen;  
    {  
        mssv = dem;  
        dem++;  
    }  
  
    public NhanVien(String ht) {  
        this.hoTen = ht;  
    }  
}
```

Initialization block

- Với các thuộc tính tĩnh thì phải sử dụng khối tĩnh, khối này được gọi khi **lớp lần đầu được nạp**.
- Ví dụ

```
class NhanVien {  
    private static int dem;  
    static {  
        Random rand = new Random();  
        dem = (int)(rand.nextDouble() * 100);  
    }  
}
```

Nạp chồng

- Nạp chồng (**overloading**) là khả năng định nghĩa nhiều phương thức **cùng tên**, nhưng **khác về danh sách tham số đầu vào**.

```
public double tinghDienTichTG(double canh,  
                                double chieuCao) {  
    return 0.5*canh*chieuCao;  
}  
  
public double tinghDienTichTG(double a,  
                                double b, double c) {  
    double p = (a+b+c)/2;  
    return Math.sqrt(p*(p-a)*(p-b)*(p-c));  
}
```

Nạp chồng

- Cho biết lỗi trong đoạn chương trình sau?

```
class Test {  
    public static void method(int x) {  
  
    }  
  
    public static int method(int y) {  
        return y;  
    }  
  
    public static float method(float y) {  
        return y;  
    }  
}
```

Ký hiệu UML biểu diễn lớp

Tên lớp
Các thuộc tính
Các phương thức

Bình thường: lớp bình thường

In nghiêng: lớp trừu tượng

Gạch chân: đối tượng

Bình thường: thuộc tính bình thường

Gạch chân: thuộc tính tĩnh

Bình thường: phương thức bình thường

In nghiêng: phương thức trừu tượng

Gạch chân: phương thức tĩnh

Ký hiệu trước thuộc tính và phương thức:

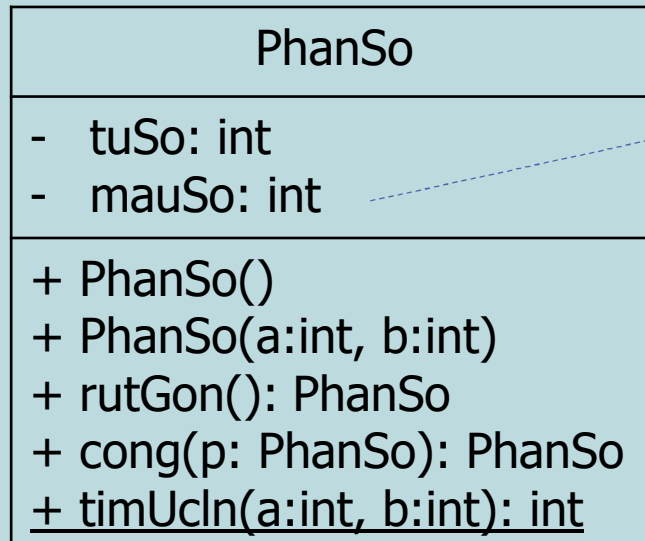
+: thuộc tính/phương thức public.

-: thuộc tính/ phương thức private.

#: thuộc tính/phương thức protected.

Ký hiệu UML biểu diễn lớp

▪ Ví dụ

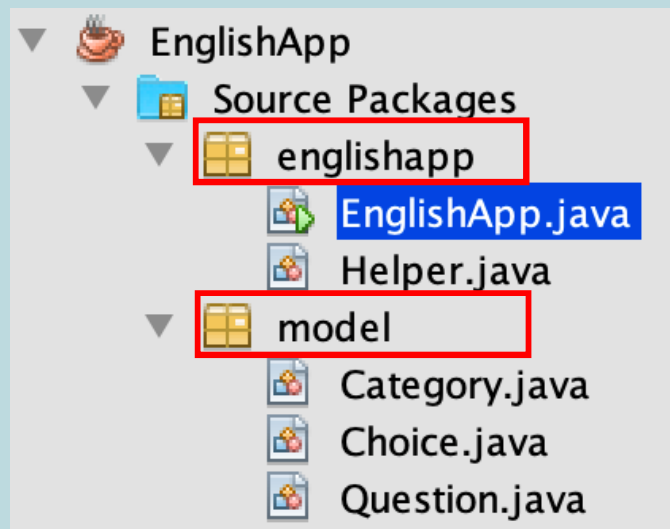


Mẫu số phải khác 0

Ký hiệu dùng để ghi chú trong UML

Gói

- Gói (package) dùng **gom nhóm** các lớp, giao diện và các gói con khác.
- Mục đích chính sử dụng gói là đảm bảo tính **duy nhất** trong tên lớp.



Gói

- Các gói chuẩn trong Java nằm trong hai gói **java** và **javax**.
- Các thuộc tính và phương thức nếu không chỉ định phạm vi truy cập tường minh, sẽ có phạm vi truy cập mặc định là **default**, tức là được phép truy cập trong phạm vi lớp đó và các lớp trong cùng gói.

Gói

- Một lớp sử dụng lớp thuộc gói khác thì
 - Dùng đầy đủ tên lớp

```
java.util.Date today = new java.util.Date();
```

- Dùng lệnh **import** chỉ đến lớp trong gói.

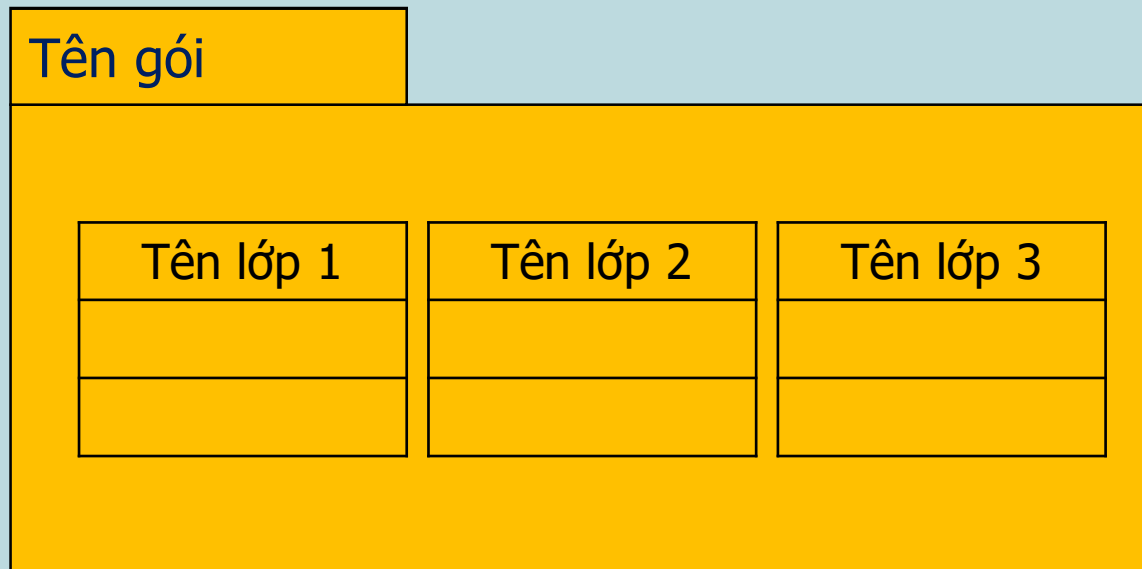
```
import java.util.Date;  
Date today = new Date();
```

- Dùng ký hiệu ***** để import các lớp của gói

```
import java.util.*;
```

Gói

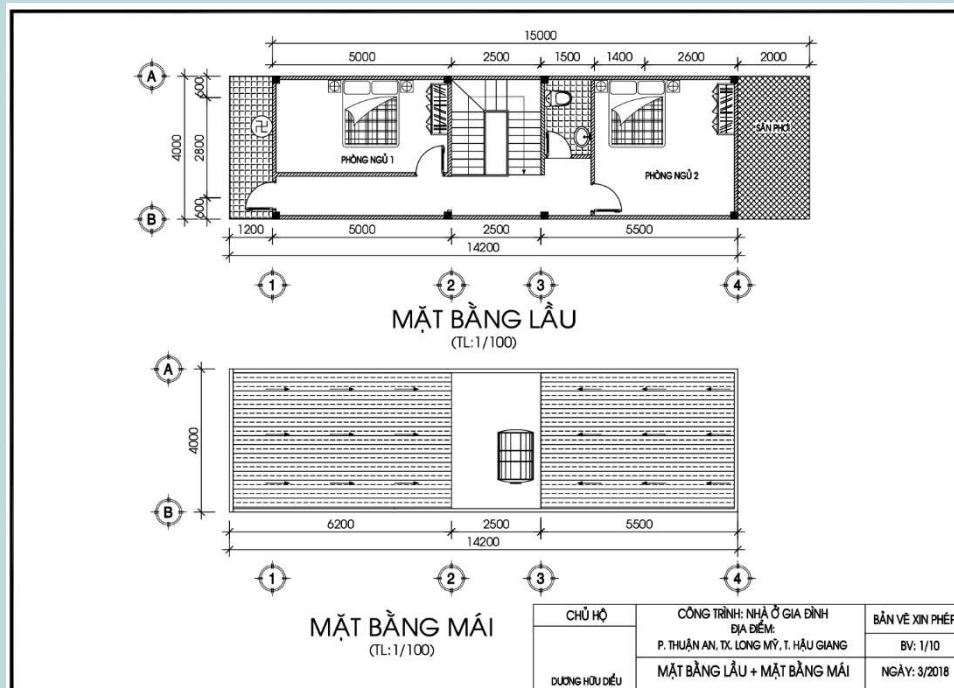
- Ký hiệu UML biểu diễn gói



Phát biểu nào sau đây đúng?

- A. Phạm vi truy cập mặc định của các thuộc tính và phương thức là private.**
- B. Phương thức thông thường không được phép truy cập các thành viên tĩnh của lớp.**
- C. Các phương thức nạp chồng phải cùng tên, khác kiểu dữ liệu trả về.**
- D. Phương thức khởi tạo không thể khai báo là tĩnh.**
- E. Trong một lớp muốn sử dụng lớp khác phải import lớp đó trước khi sử dụng.**

Quan hệ giữa lớp và đối tượng



Lớp



Đối tượng



Đối tượng



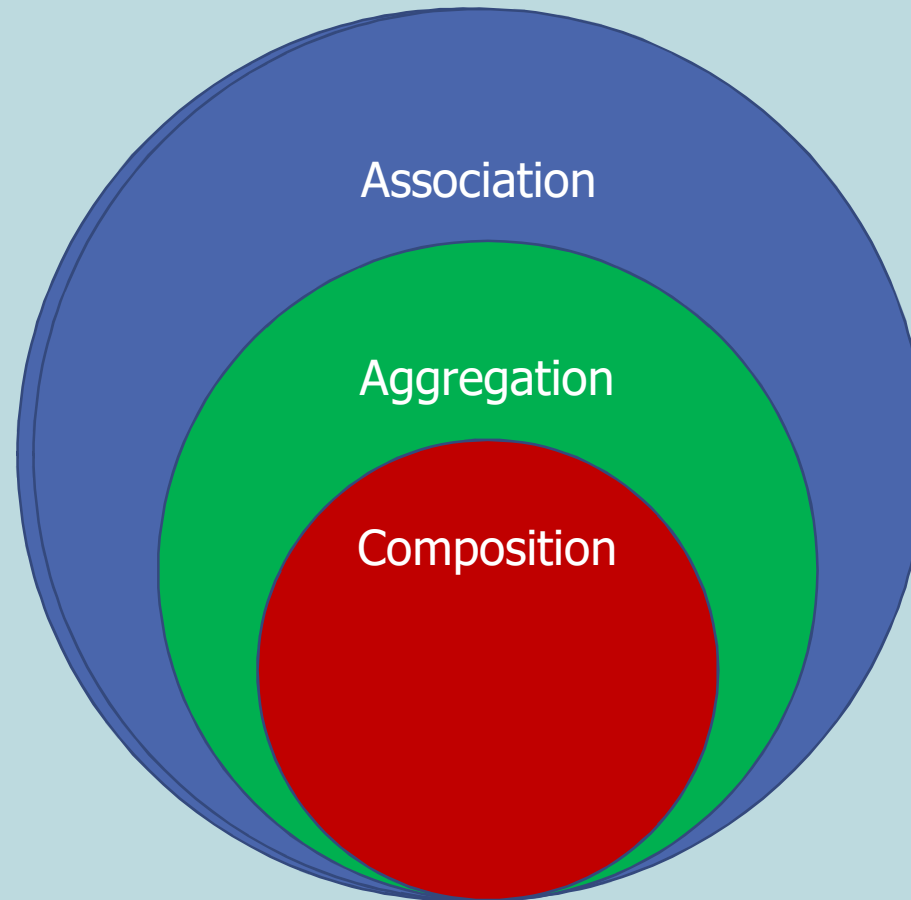
Đối tượng

Nguồn ảnh: internet

Quan hệ giữa lớp và đối tượng

- Quan hệ giữa lớp và đối tượng tương tự như
 - Quan hệ giữa **kiểu dữ liệu** và **biến**.
 - **Công thức** làm bánh và **cái bánh** cụ thể.
 - **Bảng vẽ** xây dựng nhà và **ngôi nhà** cụ thể.
- Đối tượng là một **thể hiện** cụ thể của lớp, lớp là **sự trừu tượng hóa** các đối tượng.
- Các đối tượng của cùng một lớp sẽ có **cùng** các thuộc tính và phương thức.

Quan hệ giữa hai lớp

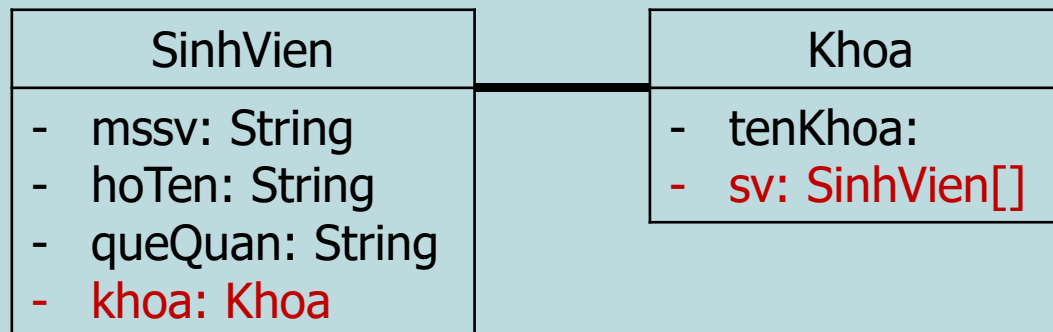


Quan hệ giữa hai lớp

- Quan hệ Association

- Lớp A **có thuộc tính** kiểu lớp B hoặc lớp B **có thuộc tính** kiểu lớp A

- Ký hiệu UML



Quan hệ giữa hai lớp

- Quan hệ Aggregation (**has-a**)
 - Lớp A và lớp B **đã có** quan hệ Association.
 - Lớp A **có thuộc tính** kiểu lớp B, nếu đối tượng a của lớp A bị hủy thì đối tượng b (thuộc tính của đối tượng a) của lớp B vẫn **có thể tồn tại**.
 - Ký hiệu UML



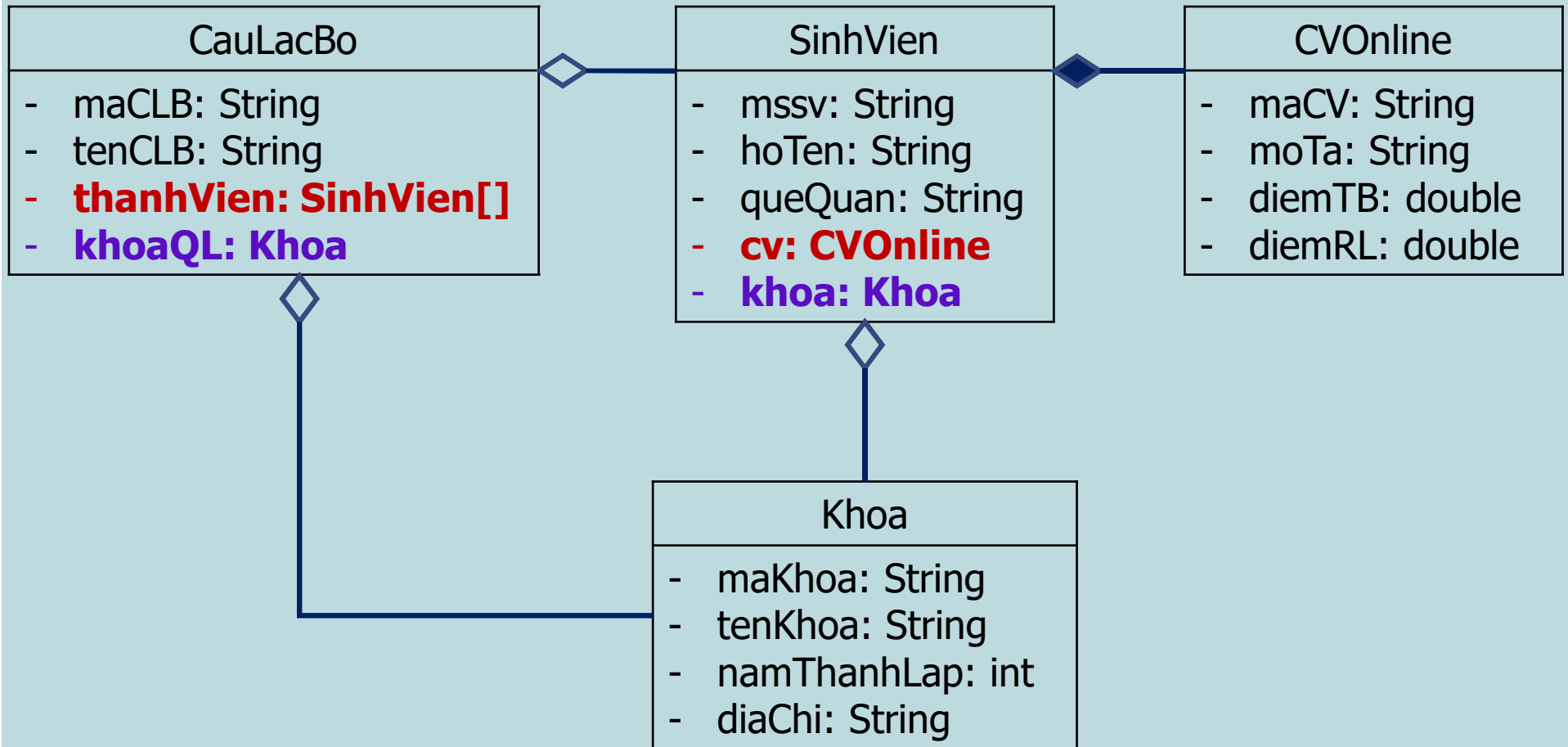
Quan hệ giữa hai lớp

- Quan hệ Composition

- Lớp A và lớp B **đã có** quan hệ Association.
- Lớp A **có thuộc tính** kiểu lớp B, nếu đối tượng a của lớp A bị hủy thì đối tượng b (thuộc tính của đối tượng a) của lớp B **không thể** tồn tại.
- Ký hiệu UML



Quan hệ giữa hai lớp



Quan hệ giữa các lớp

- Hệ thống quản lý câu hỏi trắc nghiệm quản lý câu hỏi theo danh mục, mỗi câu hỏi gồm: nội dung câu hỏi, các phương án lựa chọn câu hỏi và danh mục của câu hỏi.
- Chú ý: một câu hỏi có thể từ 1 đến nhiều phương án lựa chọn và trong đó có ít nhất một phương án đúng.
- Thiết kế các lớp và các mối quan hệ giữa các lớp thoả mãn yêu cầu trên?

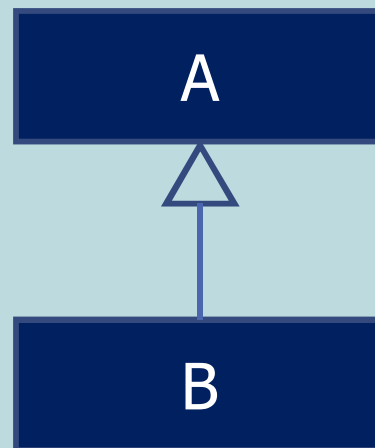
Quan hệ giữa hai lớp

- Quan hệ Dependency (**uses-a**)
 - Lớp A và lớp B **không có** quan hệ Association. Đối tượng kiểu lớp B có thể là **đối số** hoặc **kết quả trả về** hoặc **biến cục bộ** trong các phương thức của lớp A.
 - Ký hiệu UML



Quan hệ giữa hai lớp

- Quan hệ kế thừa (**is-a**)
 - Lớp B kế thừa từ lớp A: lớp B là trường hợp đặc biệt của lớp A, lớp A là trường hợp tổng quát của lớp B.
 - Ký hiệu UML



Lớp trong

- Java cho phép định nghĩa một lớp khác **bên trong** một lớp (**inner class**).
- Lớp trong **có thể truy xuất** đến các thuộc tính và phương thức của lớp ngoài chứa nó.

```
public class OuterClass {  
    [<Các thuộc tính>]  
    [<Các phương thức>]  
    class InnerClass {  
        [...]  
    }  
}
```

Immutable class

- Lớp immutable dùng tạo các đối tượng immutable, nội dung những đối tượng này sẽ **không đổi**.
- Một lớp immutable
 - Tất cả các thuộc tính phải là private và final.
 - Không có phương thức nào trả về tham chiếu có thể thay đổi dữ liệu của thuộc tính.

Immutable class

```
class SinhVien {  
    private int maSo;  
    private String ten;  
    public SinhVien(int ms, String t) {  
        this.maSo = ms;  
        this.ten = t;  
    }  
    public int getMaSo() {  
        return maSo;  
    }  
    public String getTen() {  
        return ten;  
    }  
}
```

Lớp ArrayList

- Lớp ArrayList dùng quản lý danh sách các phần tử được **lưu trữ dạng mảng**.
- Để sử dụng ArrayList phải chỉ định kiểu dữ liệu (**tham chiếu**) các phần tử sẽ quản lý.

```
// Tạo ArrayList quản lý danh sách số nguyên  
ArrayList<Integer> numbers = new ArrayList<>();  
  
// Tạo ArrayList quản lý danh sách phân số  
ArrayList<PhanSo> fractions = new ArrayList<>();
```

Một số phương thức ArrayList

```
ArrayList<String> fruits = new ArrayList<>();

// Thêm phần tử vào ArrayList
fruits.add("Apple");
// Thêm danh sách các phần tử vào ArrayList
String[] f = {"Banana", "Lemon", "Apple"};
fruits.addAll(Arrays.asList(f));

// Tìm vị trí đầu tiên xuất hiện 1 phần tử
System.out.println(fruits.indexOf("Apple"));
// Tìm vị trí cuối xuất hiện 1 phần tử.
System.out.println(fruits.lastIndexOf("Apple"));
// Kiểm tra ArrayList chứa 1 phần tử không
System.out.println(fruits.contains("Banana"));
```

Một số phương thức ArrayList

▪ Duyệt các phần tử ArrayList

```
// Sử dụng forEach của ArrayList
fruits.forEach(fr -> System.out.println(fr));

// Sử dụng Iterator
Iterator<String> fr = fruits.iterator();
while (fr.hasNext())
    System.out.println(fr.next());
```

▪ Sắp xếp

```
fruits.sort((String o1, String o2) -> {
    return o1.compareTo(o2);
});
fruits.forEach(fr -> System.out.println(fr));
```

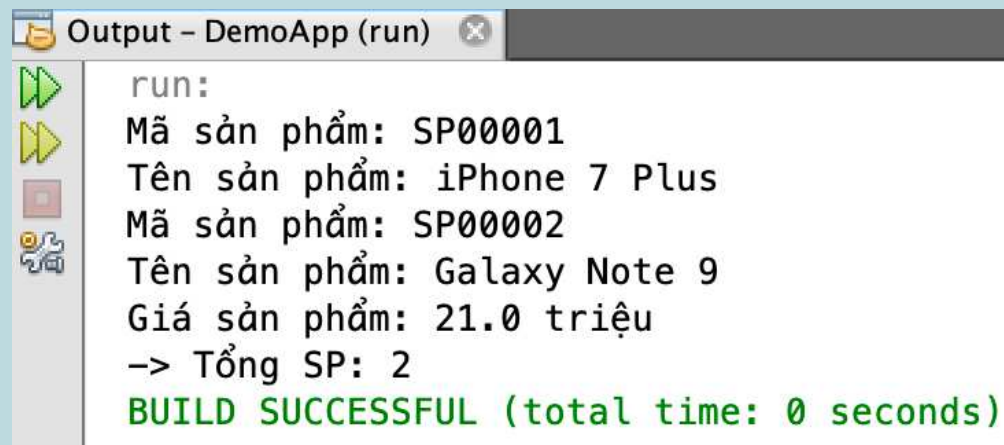
Tổng kết chương 1

1. Cho biết phương thức khởi tạo là gì, được gọi khi nào và đặc điểm của nó trong Java?
2. Cho biết sự khác nhau giữa phương thức tĩnh và phương thức thông thường?
3. Cho biết khối tĩnh (static block) là gì? Nó được gọi thực thi khi nào?
4. Trình bày quan hệ composite giữa hai lớp và cho ví dụ?
5. Lớp không đổi (immutable) là gì? làm cách nào tạo lớp không đổi?

Tổng kết chương 1

- Viết lớp **SanPham** để kết quả đoạn chương trình như hình bên dưới, mã sản phẩm do chương trình tự sinh ra và không đổi.

```
SanPham sp1 = new SanPham("iPhone 7 Plus");  
SanPham sp2 = new SanPham("Galaxy Note 9", 21);  
sp1.show();  
sp2.show();  
System.out.println("-> Tổng SP: " + SanPham.getDem());
```



```
run:  
Mã sản phẩm: SP00001  
Tên sản phẩm: iPhone 7 Plus  
Mã sản phẩm: SP00002  
Tên sản phẩm: Galaxy Note 9  
Giá sản phẩm: 21.0 triệu  
-> Tổng SP: 2  
BUILD SUCCESSFUL (total time: 0 seconds)
```

