

LẬP TRÌNH OOP VỚI JAVA

KẾ THỪA

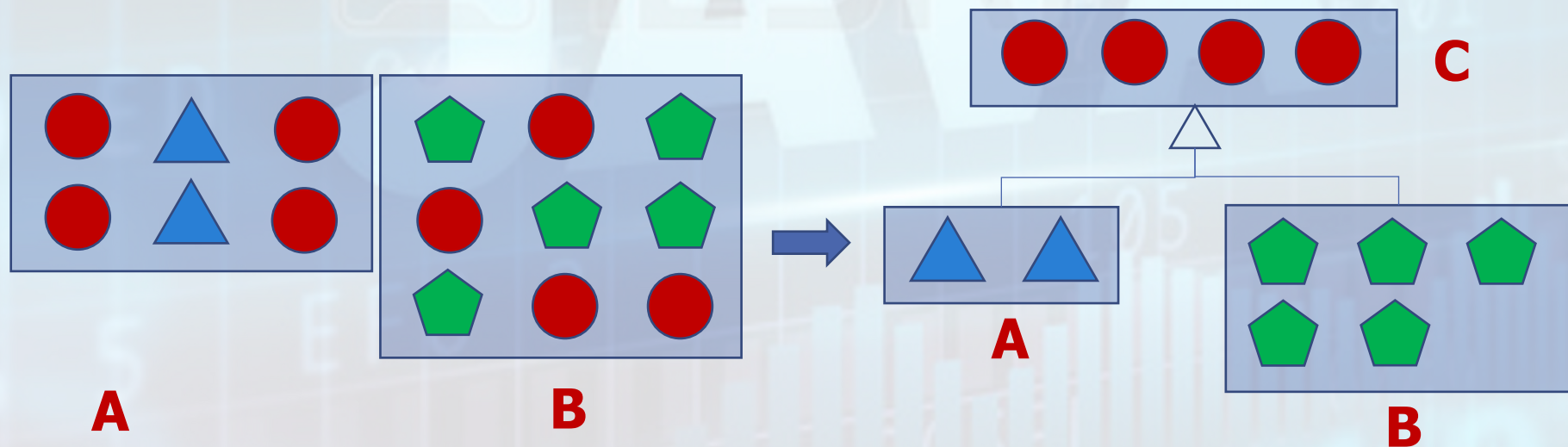
ThS. Dương Hữu Thành
Khoa CNTT, Đại học Mở Tp.HCM
thanh.dh@ou.edu.vn



Nội dung chính

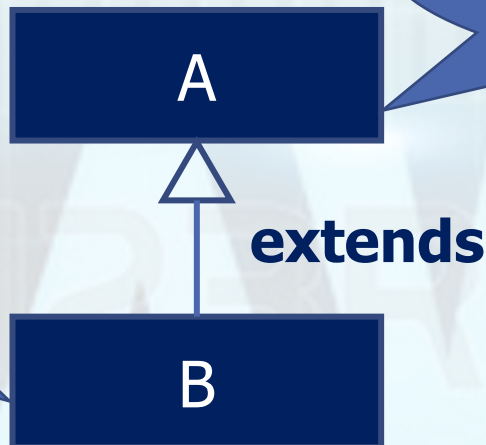
- 1. Kế thừa**
- 2. Phương thức khởi tạo trong kế thừa**
- 3. Phạm vi truy cập protected**
- 4. Từ khoá super**
- 5. Ghi đè**
- 6. Liên kết động (Dynamic Binding)**
- 7. Toán tử instanceof**
- 8. Lớp Object**
- 9. Xử lý ngoại lệ**

- Kế thừa (inheritance) có nghĩa là một lớp **đạt được** các phương thức và thuộc tính của lớp khác.
- Kế thừa thường thực thi bằng **cách gom nhóm** các thành phần **giống nhau** của các lớp để tạo thành lớp mới và các lớp ban đầu kế thừa lớp vừa tạo.



- Ký hiệu UML

Lớp dẫn xuất
(derived class)
hoặc lớp con
(subclass)

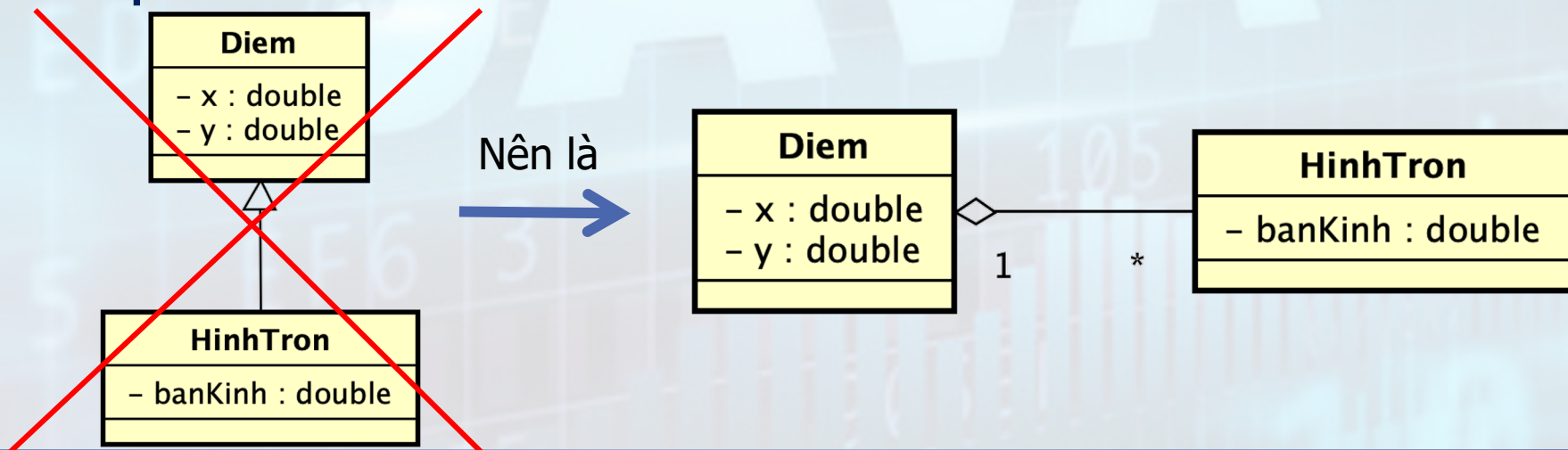


Lớp cơ sở (base
class) hoặc lớp cha
(superclass)

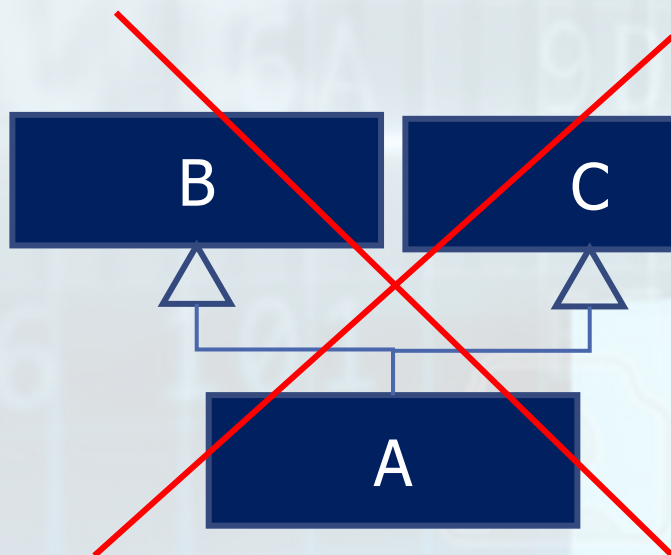
- Từ khoá **extends**

```
class B extends A {  
  
}
```

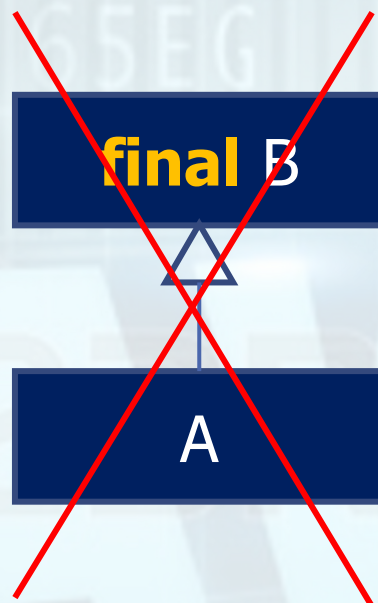

- Chú ý: ta thiết lập mối quan hệ kế thừa khi trả lời được câu hỏi: **B là một loại (is-a) của A**
 - Hình tròn (B) là một loại của Ellipse (A).
 - Hình vuông (B) là một loại của hình chữ nhật (A).
- Ví dụ **thiết kế sai**: hình tròn được xác định bởi tọa độ tâm và bán kính.



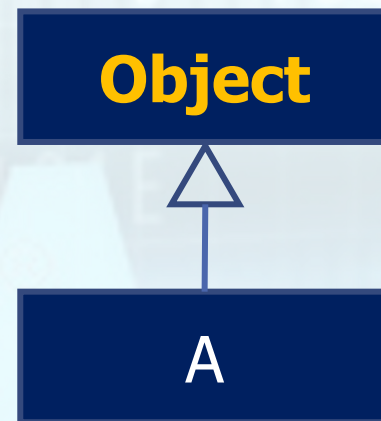
- Hệ thống quản lý thông tin sinh viên (SV) và giảng viên (GV). Trong đó:
 - Sinh viên bao gồm: MSSV, họ tên, quê quán, ngày sinh, giới tính, ngày nhập học. SV có thể xem thông tin cá nhân, đánh giá giảng viên, xem thời khoá biểu và đăng ký môn học.
 - Giảng viên bao gồm: mã giảng viên, họ tên, quê quán, giới tính, ngày vào trường, học vị, bậc lương, hệ số lương. GV được phép xem thông tin cá nhân, xem thu nhập, đăng ký giờ dạy, tra cứu danh bạ của trường.
- **Viết các lớp cần thiết cho mô tả trên?**



Java **không** hỗ trợ đa kế thừa



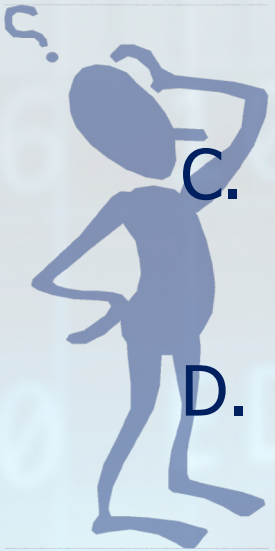
Không được phép kế thừa lớp **final**



Lớp không khai báo kế thừa tường minh, **mặc định** kế thừa lớp **Object**

Phát biểu nào sau đây đúng?

- A. Một lớp có thể kế thừa nhiều lớp cùng lúc.
- B. Phương thức tĩnh không được phép kế thừa ở lớp con.
- C. Phương thức private của lớp cha không được kế thừa ở lớp con.
- D. Khi lớp B kế thừa lớp A, nó còn ngầm định kế thừa lớp Object.




CÂU HỎI

Phương thức khởi tạo trong kế thừa

- Phương thức khởi tạo **không** được kế thừa ở lớp con.
- Thứ tự các phương thức khởi tạo được gọi là từ **các lớp cha trước, rồi đến lớp con**.
- Các phương thức khởi tạo lớp con phải gọi phương thức khởi tạo lớp cha, nếu không Java sẽ **ngầm định** gọi phương thức khởi tạo không tham số của lớp cha.

Phương thức khởi tạo trong kế thừa

- Cho biết kết quả xuất ra đoạn chương trình?



```
class A {  
    public A() {  
        System.out.print("A");  
    }  
}  
class B extends A {  
    public B() {  
        System.out.print("B");  
    }  
}  
class C extends B {  
    public C() {  
        System.out.print("C");  
    }  
}
```

```
C c = new C();
```

CÂU HỎI

Phương thức khởi tạo trong kế thừa

- Đoạn chương trình sau có lỗi gì và nêu cách khắc phục?




```
class A {  
    public A(int v) {  
        System.out.print("A");  
    }  
}  
class B extends A {  
    public B() {  
        System.out.println("B");  
    }  
}
```

```
B b = new B();
```

Phương thức khởi tạo trong kế thừa

- Cho biết kết quả xuất ra đoạn chương trình?




```
class A {  
    public A() {  
        System.out.println("A()");  
    }  
    public A(int v) {  
        System.out.println("A(int)");  
    }  
}  
class B extends A {  
    public B(int v) {  
        System.out.println("B(int)");  
    }  
}
```

```
B b = new B(5);
```


Phương thức khởi tạo trong kế thừa

- Cho biết kết quả đoạn chương trình sau và giải thích?



```
final class A {  
    protected final void show() {  
        System.out.print("A");  
    }  
}  
class B extends A {  
    public void show(String b) {  
        System.out.print(b);  
    }  
}
```

```
B b = new B();  
b.show();
```

Phạm vi truy cập **protected**

- Thuộc tính hay phương thức được khai báo phạm vi **protected** thì chúng được phép truy cập trong lớp, các lớp cùng gói và các lớp con kế thừa lớp chứa chúng.

```
class A {  
    protected int x;  
    public void show() {  
        System.out.println(x);  
    }  
}
```

```
class B extends A {  
    public void handle() {  
        this.x++;  
    }  
}
```




Phạm vi truy cập protected

Access Modifier

private

public

protected

default

Phạm vi truy cập

Bên trong lớp



Lớp khác, cùng gói



Lớp khác, khác gói




*Lớp con kế thừa,
khác gói*



Phạm vi truy cập protected

- Cho biết đoạn chương trình có lỗi không? Nếu không cho biết kết quả xuất ra?



```
class A {  
    protected void show() {  
        System.out.print("A");  
    }  
    private void show(String c) {  
        System.out.print("B");  
    }  
}  
  
class B extends A {  
    public void show(String c) {  
        System.out.print("C");  
        show();  
    }  
}
```

```
B b = new B();  
b.show();
```

```
B b = new B();  
b.show("");
```

```
A a = new B();  
a.show("");
```

?

I

Từ khoá super

- Lớp con muốn truy xuất vào các thành phần của lớp cha sử dụng từ khóa **super**.
- Gọi phương thức khởi tạo lớp cha


```
super ( [CácĐốiSố] );
```

- Gọi phương thức và thuộc tính lớp cha

```
super.tênThuộcTính;
```

```
super.tênPhươngThứcLớpCha ( [CácĐốiSố] );
```


- Cho biết kết quả xuất ra đoạn chương trình?



```
class A {  
    public A(String s) {  
        System.out.println(s);  
    }  
}  
class B extends A {  
    public B(String name) {  
        name = "Hello " + name;  
        super(name);  
        System.out.println(name);  
    }  
}
```

```
B b = new B("Java");
```


- Cho biết kết quả đoạn chương trình?



```
class A {  
    public A() {  
        System.out.print("A");  
    }  
    public A(String c) {  
        this();  
        System.out.print(c);  
    }  
}  
class B extends A {  
    public B() {  
        super("T");  
        System.out.print("B");  
    }  
}
```

```
B b = new B();
```

?

HỎI

- Lớp con có thể ghi đè (overriding) các phương thức của lớp cha **cho phép để định nghĩa hành vi riêng** của nó.

```
class English {  
    protected void hello(String name) {  
        System.out.print("Hello " + name);  
    }  
}
```

```
class Vietnamese extends English {  
    public void hello(String name) {  
        System.out.print("Chào " + name);  
    }  
}
```


- Một số chú ý khi ghi đè
 - Tên phương thức, danh sách tham số và kiểu trả về **phải giống** với phương thức định ghi đè của lớp cha.
 - Phạm vi truy cập **không được phép** giới hạn hơn phạm vi truy cập của phương thức định ghi đè.
 - **Không thể** ghi đè phương thức **final**.

- Cho biết kết quả xuất ra đoạn chương trình?

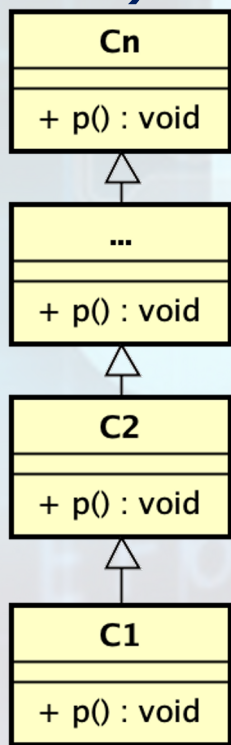


```
class A {  
    public void hienThi() {  
        System.out.print("A");  
    }  
}  
class B extends A {  
    protected void hienThi() {  
        System.out.print("B");  
    }  
}
```

```
B b = new B();  
b.hienThi();
```


Dynamic Binding

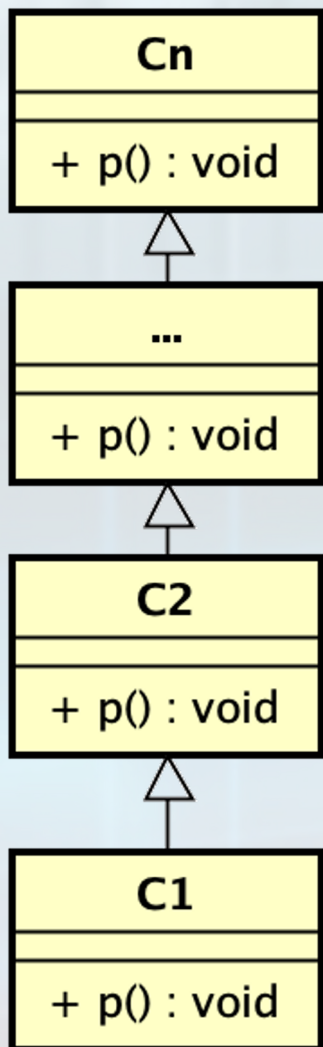
- Một phương thức sẽ được thực thi ở nhiều lớp trong **chuỗi quan hệ kế thừa**. Khi đó JVM sẽ quyết định thực thi của lớp nào được gọi **lúc thực thi** (runtime).



```
Cn c = new C1 ();  
c.p();
```

Phương thức p()
của lớp nào
được thực thi?

Dynamic Binding



Kiểu khai báo
(Declared Type)


Kiểu thực sự (Actual Type): lớp thực
sự mà đối tượng tham chiếu đến

```
Cn c = new C1 ();
c.p ();
```

➔ Phương thức `p()` được quyết định bởi kiểu thực sự lúc thực thi ➔ **dynamic binding**

➔ Tổng quát: `c` là một thể hiện của `C1`, `C2`, ..., `Cn` trong quan hệ kế thừa như hình. Khi `c` gọi phương thức `p()`, JVM sẽ tìm theo thứ tự từ `C1` đến `Cn`, một khi tìm thấy nó sẽ thực thi và kết thúc quá trình tìm kiếm.

- Cho biết kết quả xuất ra đoạn chương trình?



```
class A {  
    public void show() {  
        System.out.println("A");  
    }  
}  
class B extends A {  
    public void show() {  
        super.show();  
        System.out.println("B");  
    }  
}  
class C extends B {  
    public void show() {  
        System.out.println("C");  
        super.show();  
    }  
}
```

```
A a = new C();  
a.show();
```

?

HỎI

- Cho biết kết quả xuất ra đoạn chương trình?



```
class A {  
    private void hienThi() {  
        System.out.print("A");  
    }  
}  
class B extends A {  
    public void hienThi() {  
        super.hienThi();  
        System.out.print("B");  
    }  
}
```

```
B b = new B();  
b.hienThi();
```


- Cho lớp A và lớp B kế thừa lớp A

```
class A {  
    public void show() {  
        System.out.print("A");  
    }  
}
```



- Viết lớp B và hoàn thiện đoạn chương trình sau để kết quả xuất ra là "ABA"

```
A a = ...;  
a.show();
```

CÂU HỎI



Ghi đề


- Cho biết kết quả của đoạn chương trình?

```
class A {  
    private String getName() {  
        return "A";  
    }  
    public void show() {  
        System.out.println(this.getName());  
    }  
}  
class B extends A {  
    public String getName() {  
        return "B";  
    }  
}
```

```
A a = new A();  
a.show();  
B b = new B();  
b.show();
```

CÂU HỎI


- Cho biết kết quả đoạn chương trình sau và giải thích?



```
class A {  
    protected final void show() {  
        System.out.print("A");  
    }  
}  
class B extends A {  
    public void show() {  
        System.out.print("B");  
    }  
}
```

```
B b = new B();  
b.show();
```

- Cho biết kết quả xuất ra đoạn chương trình?



```
class A {  
    protected final void show() {  
        System.out.print("A");  
    }  
}  
class B extends A {  
    public void show(String text) {  
        System.out.print(text);  
    }  
}
```

```
B b = new B();  
b.show();
```


- Cho biết kết quả xuất ra đoạn chương trình?



```
class A {  
    public static void show() {  
        System.out.println("A");  
    }  
}  
class B extends A {  
    public static void show() {  
        System.out.println("B");  
    }  
}
```

```
A a = new B();  
a.show();
```

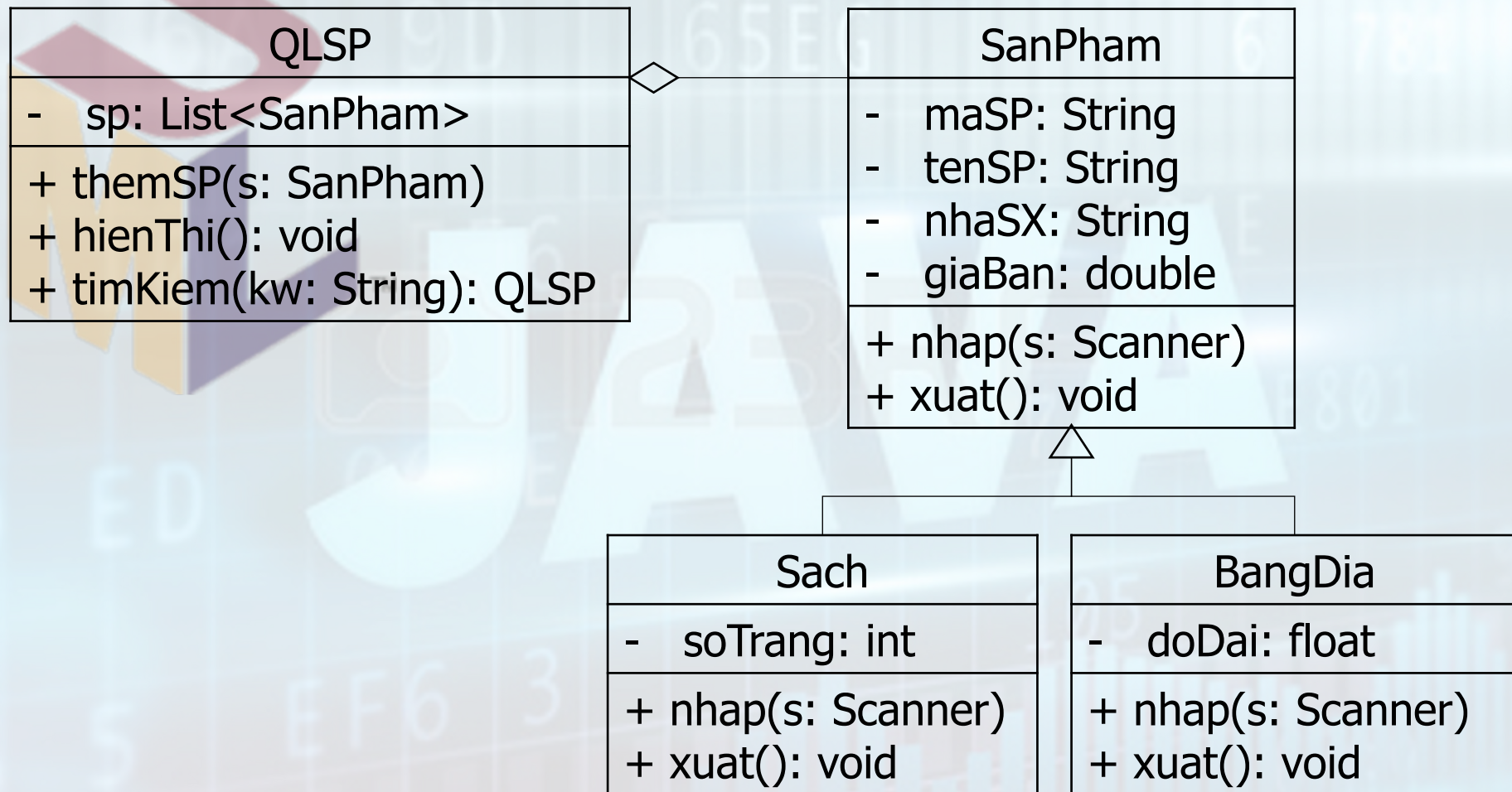


Cho biết kết quả của chương trình?

```
package a;
public class A {
    protected static int count = 0;
    void inc() {
        count++;
    }
    public static void main(String[] args) {
        A a = new b.B();
        a.inc();
        System.out.println(A.count);
    }
}
```

```
package b;
public class B extends a.A {
    protected void inc() {
        count += 2;
    }
}
```


- Một công ty chuyên sản xuất sách và băng đĩa phục vụ học tập trong nhà trường phổ thông. Mỗi sản phẩm có mã sản phẩm, tên sản phẩm, nhà sản xuất, giá bán. Ngoài ra:
 - Sách thêm thông tin số trang.
 - Băng đĩa thêm thông tin độ dài thời gian phát.
- Viết chương quản lý các sản phẩm công ty.
 - Nhập và xuất sản phẩm.
 - Tìm kiếm sản phẩm.
 - Sắp xếp các sản phẩm giảm theo giá bán.




```
class SanPham {  
    private String maSP;  
    private String tenSP;  
    private String nhaSX;  
    private double giaBan;  
  
    public void nhap(Scanner scanner) {  
        // nhập maSP, tenSP, nhaSX, giaBan  
    }  
    public void xuat() {  
        // xuất maSP, tenSP, nhaSX, giaBan  
    }  
    // Các phương thức getter và setter  
}
```

```
class Sach extends SanPham {  
    private int soTrang;  
  
    @Override  
    public void nhap(Scanner scanner) {  
        super.nhap(scanner);  
        // Nhập số trang  
    }  
    @Override  
    public void xuất() {  
        super.xuat();  
        // Xuất số trang  
    }  
    // Các phương thức getter và setter  
}
```



```
class BangDia extends SanPham {  
    private float doDai;  
  
    @Override  
    public void nhap(Scanner scanner) {  
        super.nhap(scanner);  
        // Nhập độ dài  
    }  
  
    @Override  
    public void xuat() {  
        super.xuat();  
        // Xuất độ dài  
    }  
    // Các phương thức getter và setter  
}
```



Ví dụ

```
class QLSP {  
    private List<SanPham> sp = new ArrayList<>();  
    public void themSP(SanPham s) {  
        this.sp.add(s);  
    }  
    public void hienThi() {  
        this.sp.forEach(s -> s.xuat());  
    }  
    public QLSP timKiem(String kw) {  
        QLSP ql = new QLSP();  
        this.sp.stream().filter(s->s.getTenSP().contains(kw))  
            .forEach(s->ql.themSP(s));  
        return ql;  
    }  
}
```


- Phương thức sắp xếp trong lớp QLSP

```
class QLSP {  
    // ...  
    public void sapXepGiamTheoGia() {  
        this.sp.sort((sp1, sp2) -> {  
            double k = sp1.getGiaBan() - sp2.getGiaBan();  
            return -((k > 0) ? 1:(k < 0)?-1:0);  
        });  
    }  
}
```



Ví dụ

```
QLSP ql = new QLSP();
try (Scanner scanner = new Scanner(System.in)) {
    SanPham sp1 = new Sach();
    sp1.nhap(scanner);
    SanPham sp2 = new BangDia();
    sp2.nhap(scanner);
    ql.themSP(sp1);
    ql.themSP(sp2);
    ql.hienThi();

    String kw = scanner.nextLine();
    QLSP kq = ql.timKiem(kw);
    kq.hienThi();

    ql.sapXepGiamTheoGia();
    ql.hienThi();
}
```


- Tiếp tục ví dụ trên, sau một thời gian phát triển, công ty kinh doanh thêm văn phòng phẩm với các thuộc tính mã sản phẩm, tên sản phẩm, nhà sản xuất, giá bán và công dụng của nó.
- Viết tiếp lớp **VanPhongPham** bổ sung vào chương trình trên để công ty quản lý loại sản phẩm này với các chức năng hiện tại, nhưng không thay đổi mã nguồn đã có.



CÂU HỎI

Phát biểu nào sau đây sai?

- A. Không thể ghi đè phương thức khởi tạo.
- B. Phương thức private không thể được viết đè (overriding).
- C. Phương thức static không thể được viết đè (overriding).
- D. Phương thức static của lớp cha không được kế thừa ở lớp con.
- E. Trong Java, một lớp có tối đa một lớp cha trực tiếp.

Toán tử instanceof

- Toán tử **instanceof** dùng kiểm tra một đối tượng là thể hiện của lớp nào.

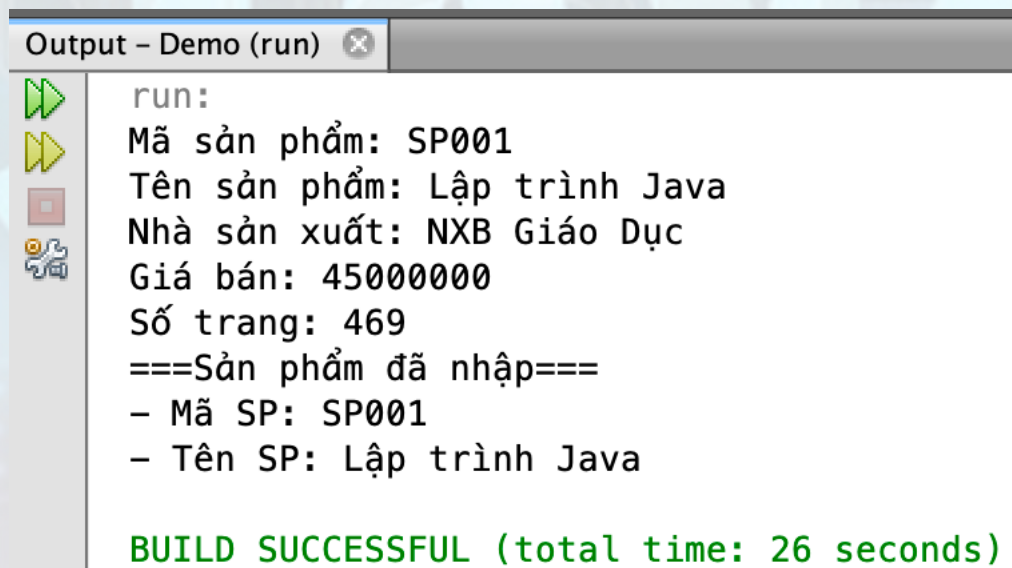
```
SanPham s1 = new Sach();  
  
if (s1 instanceof Sach)  
    System.out.println("true");  
else  
    System.out.println("false");
```

- Các phương thức hữu dụng lớp **Object**
 - **toString()**: trả về chuỗi đại diện cho đối tượng.
 - **equals(Object obj)**: kiểm tra hai đối tượng có bằng nhau.
 - **hashCode()**: trả về mã băm của đối tượng.
 - **clone()**: sao chép đối tượng ra đối tượng mới.
 - **getClass()**: trả về lớp mà đối tượng được tạo ra.

- Lớp **SanPham** ghi đè phương thức **toString()**

```
class SanPham {  
    // ...  
    @Override  
    public String toString() {  
        return String.format("- Mã SP: %s\n- Tên SP: %s\n",  
                               this.maSP, this.tenSP);  
    }  
}
```

```
public static void main(String[] args) {  
    Scanner scanner = new Scanner(System.in);  
    SanPham sp = new Sach();  
    sp.nhap(scanner);  
    System.out.println(sp);  
}
```



```
Output - Demo (run) x  
run:  
Mã sản phẩm: SP001  
Tên sản phẩm: Lập trình Java  
Nhà sản xuất: NXB Giáo Dục  
Giá bán: 45000000  
Số trang: 469  
===Sản phẩm đã nhập===  
- Mã SP: SP001  
- Tên SP: Lập trình Java  
BUILD SUCCESSFUL (total time: 26 seconds)
```


- Ví dụ: phương thức **equals ()** và **hashCode ()**

```
class SanPham {  
    // ...  
    @Override  
    public boolean equals(Object obj) {  
        SanPham sp = (SanPham) obj;  
        return this.maSP.toLowerCase()  
            .equals(sp.maSP.toLowerCase());  
    }  
    @Override  
    public int hashCode() {  
        int hash = 5;  
        hash = 97 * hash + Objects.hashCode(this.maSp);  
        return hash;  
    }  
}
```



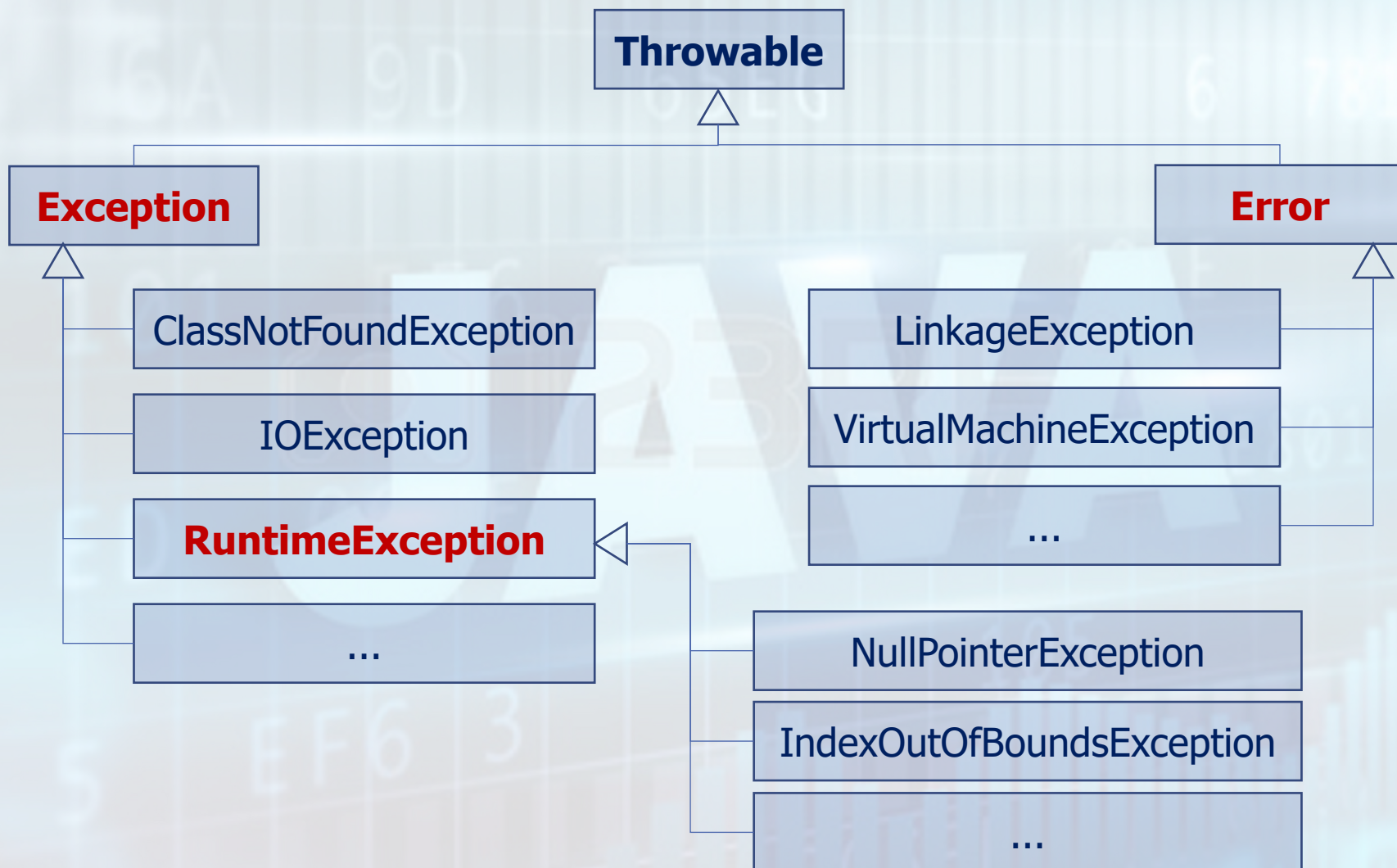
Xử lý ngoại lệ

- Ngoại lệ (exception) là thuật ngữ chỉ tình huống bất thường xảy ra khi đang chạy chương trình.
- Xử lý ngoại lệ nhằm cho phép chương trình xử lý tình huống không mong muốn xảy ra và chương trình tiếp tục thực thi bình thường.



Xử lý ngoại lệ

- Ngoại lệ được phân thành 3 loại:
 - Lỗi hệ thống (System error) được ném ra từ JVM, đại diện bởi lớp Error.
 - Lỗi chương trình hoặc tình huống tác động từ bên ngoài, đại diện bởi lớp Exception. Các ngoại lệ thuộc nhóm này, Java bắt buộc lập trình viên phải xử lý.
 - Lỗi thực thi (runtime exception) là lỗi do lập trình viên.





Xử lý ngoại lệ

```
method1() {  
    try {  
        // Thực thi phương thức  
    } catch (Exception ex) {  
        // Xử lý ngoại lệ  
    }  
}
```

Bắt (**catch**)
ngoại lệ

Khai báo
(**declare**)
ngoại lệ

```
method2() throws Exception {  
    if (LỗiXảyRa) {  
        throw new Exception();  
    }  
}
```


Ném
(**throws**)
ngoại lệ



Xử lý ngoại lệ

```
try {  
    <try-block>  
} catch (exception-1) {  
    <catch-block>  
} catch (exception-2) {  
    <catch-block>  
} ... catch (exception-n) {  
    <catch-block>  
} finally {  
    <finally-block>  
}
```



- Cho biết kết quả đoạn chương trình?



```
try {  
    System.out.println("A");  
    System.out.println(Integer.parseInt("b"));  
    System.out.println("B");  
} catch (NumberFormatException ex) {  
    System.out.println("ERROR");  
}
```

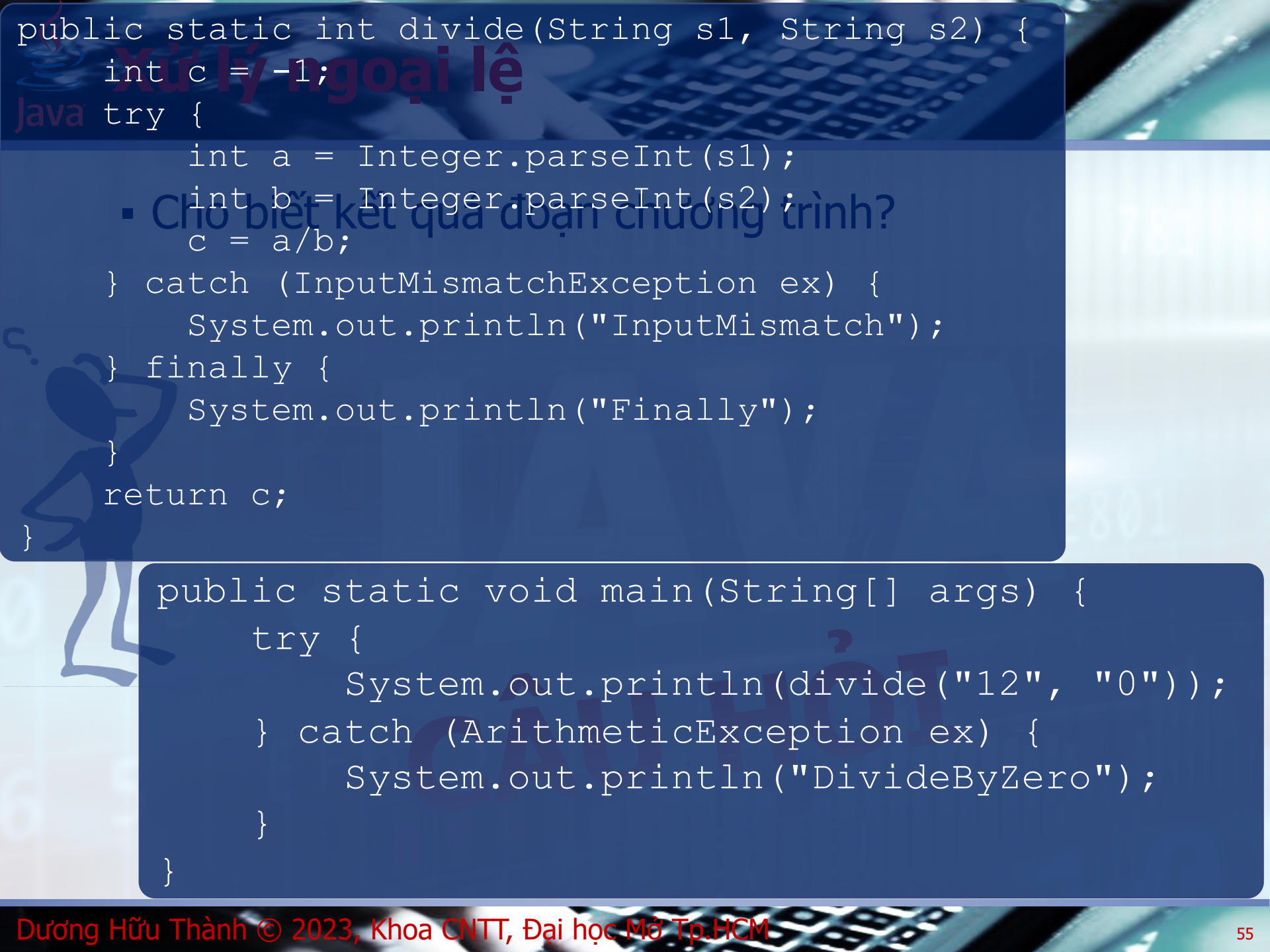
CÂU HỎI

- Cho biết kết quả đoạn chương trình?



```
try {  
    System.out.println("A");  
    System.out.println(10/0);  
    System.out.println("B");  
} catch (NumberFormatException ex) {  
    System.out.println("ERROR");  
} finally {  
    System.out.println("DONE");  
}
```

CÂU HỎI



Xử lý ngoại lệ

```
public static int divide(String s1, String s2) {  
    int c = -1;  
    try {  
        int a = Integer.parseInt(s1);  
        int b = Integer.parseInt(s2);  
        c = a/b;  
    } catch (InputMismatchException ex) {  
        System.out.println("InputMismatch");  
    } finally {  
        System.out.println("Finally");  
    }  
    return c;  
}
```

Cho biết kết quả đoạn chương trình?

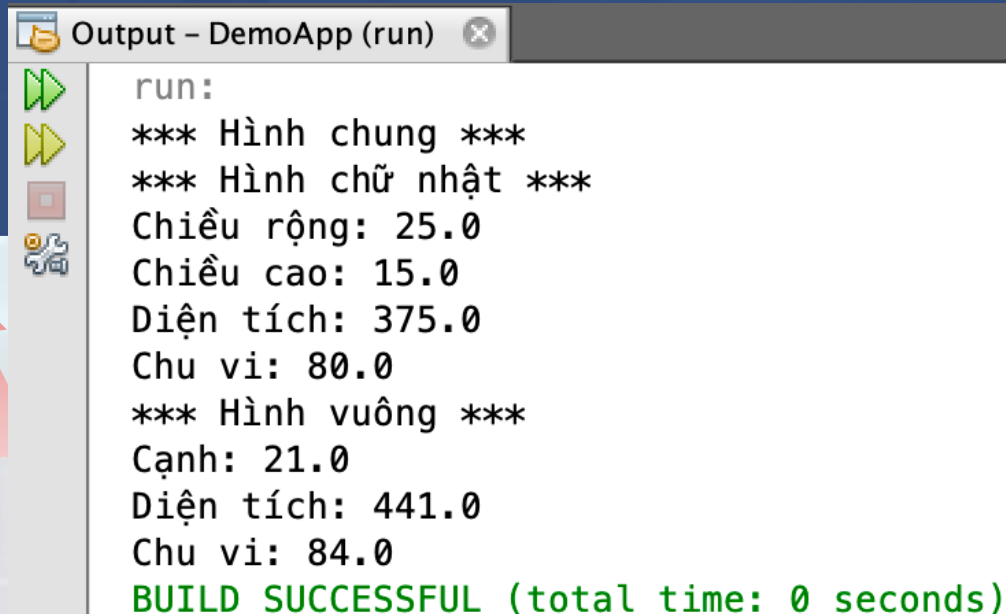
```
public static void main(String[] args) {  
    try {  
        System.out.println(divide("12", "0"));  
    } catch (ArithmeticException ex) {  
        System.out.println("DivideByZero");  
    }  
}
```

1. Phương thức khởi tạo làm việc như thế nào trong quan hệ kế thừa?
2. Phương thức tĩnh làm việc như thế nào trong quan hệ kế thừa?
3. Lớp con có thể truy cập các thành viên có phạm vi truy cập nào của lớp cha?
4. Điểm giống và khác nhau của overriding và overloading?
5. Mặc định một lớp kế thừa lớp nào? Trình bày vài phương thức quan trọng lớp đó?

- Viết các lớp cần thiết để đoạn chương trình sau cho kết quả như trong hình?

```
Shape s0 = new Shape("Hình chung");  
Shape s1 = new Rectangle("Hình chữ nhật", 25, 15);  
Rectangle s2 = new Square("Hình vuông", 21);
```

```
s0.show();  
s1.show();  
s2.show();
```



```
Output - DemoApp (run) x  
run:  
*** Hình chung ***  
*** Hình chữ nhật ***  
Chiều rộng: 25.0  
Chiều cao: 15.0  
Diện tích: 375.0  
Chu vi: 80.0  
*** Hình vuông ***  
Cạnh: 21.0  
Diện tích: 441.0  
Chu vi: 84.0  
BUILD SUCCESSFUL (total time: 0 seconds)
```

```

try {
    TamGiac tg = new TamGiac(1, 1, 3);
    System.out.println(tg);
} catch (InputMismatchException ex) {
    System.out.println("Lỗi: " + ex.getMessage());
} finally {
    TamGiac tg1 = new TamGiac(2, 3, 4);
    TamGiac tg2 = new TamGiacCan(2, 3);
    TamGiac tg3 = new TamGiacDeu(3);

    System.out.println(tg1);
    System.out.println(tg2);
    System.out.println(tg3);
}

```

Viết các lớp

- **TamGiac**
- **TamGiacCan**
- **TamGiacDeu**

để kết quả thực thi đoạn
chương trình như trong hình

```

run:
Lỗi: (1.0, 1.0, 3.0) không phải 3 cạnh tam giác Tam giác thường
-Tam giác thường
-Độ dài ba cạnh (2.0, 3.0, 4.0)
-Diện tích: 2.90
Tam giác cân
-Độ dài ba cạnh (2.0, 2.0, 3.0)
-Diện tích: 1.98
Tam giác đều
-Độ dài ba cạnh (3.0, 3.0, 3.0)
-Diện tích: 3.90
BUILD SUCCESSFUL (total time: 0 seconds)

```


Tổng kết

- Một hệ thống ngân hàng có 2 loại tài khoản là có kỳ hạn và không kỳ hạn. Một tài khoản bao gồm các thông tin số tài khoản, tên tài khoản, điện thoại, email, số tiền, ngày tạo tài khoản, trạng thái. Riêng tài khoản có kỳ hạn có thông tin kỳ hạn và ngày đáo hạn.
- Hệ thống cho phép khách hàng xem thông tin cá nhân, tra cứu tài khoản, tính tiền lãi, rút tiền, chuyển tiền, nộp tiền và hệ thống tự động cập nhật ngày đáo hạn khi đến hạn đối với tài khoản có kỳ hạn.
- **Thiết kế các lớp quản lý các tài khoản.**

Tổng kết

- Thư mục chứa thông tin tên thư mục, ngày tạo. Tập tin cũng là một loại thư mục có thêm thông tin loại tập tin và dung lượng. Thư mục có thể chứa các thư mục con và nhiều tập tin khác. Trong đó:

- Thư mục bắt buộc phải có tên, còn tập tin phải có tên và dung lượng.
- Một thư mục bị xóa thì tất cả tập tin, thư mục con của nó cũng bị xóa theo.

- Thiết kế các lớp **TapTin**, **ThuMuc** và quan hệ giữa chúng theo yêu cầu trên.

Q&A

