

Chapter 3

Lớp và đối tượng

CT176 – LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Mục tiêu

Chương này nhằm giới thiệu cách thức xây dựng lớp, khởi tạo - khai thác đối tượng và cơ bản về cách xử lý **ngoại lệ** trong Java

Nội dung

- Tạo lớp và đối tượng
- Khởi tạo và hủy đối tượng
- Xử lý ngoại lệ

Đối tượng và lớp

Đối tượng và lớp (object & class)

- Đối tượng:

- Đối tượng là một thực thể phần mềm, hay là sự mô hình hóa của một sự vật hay khái niệm trong thực tế
- Chương trình là một tập các **đối tượng** “tương tác” lẫn nhau
- Một đối tượng **tồn tại trong bộ nhớ** của chương trình, thực hiện 1 **tác vụ** hay **chức năng** nào đó
- Mỗi đối tượng có 2 thành phần:
 - **Dữ liệu (thuộc tính, attribute)**: mô tả cho **trạng thái** của đối tượng
 - **Hành vi (phương thức, method)**: thể hiện **khả năng** của đối tượng
- Đối tượng cũng là 1 **biến** (variable) trong chương trình

Đối tượng và lớp (object & class)

- Lớp:

- Là **kiểu** (type/datatype) của đối tượng: mỗi đối tượng phải thuộc 1 kiểu (loại) nào đó
- Là một **đặc tả** (specification) của một kiểu dữ liệu mới, nó mô tả các thành phần của kiểu dữ liệu đó:
 - Các **dữ liệu** mà một đối tượng thuộc lớp đó có thể lưu trữ
 - Các **phương thức** của các đối tượng thuộc lớp đó
- Có thể hiểu, lớp là một khuôn mẫu để tạo ra các đối tượng.
- Ví dụ: String, Scanner là các lớp được định nghĩa sẵn của Java
 - Các đối tượng thuộc lớp String có thể chứa các chuỗi ký tự (String s = "Hello World"); và các thao tác trên chuỗi ký tự (s.toLowerCase(); s.toUpperCase();)

Đối tượng và lớp (object & class)

- Biến kiểu dữ liệu cơ bản:
 - Chứa dữ liệu
 - Tác động lên dữ liệu: gọi các hàm
- Biến đối tượng:
 - Chứa dữ liệu + các thao tác (phương thức) trên dữ liệu
 - Tác động lên dữ liệu của đối tượng: gọi các phương thức của chính đối tượng

```
int i = 5;  
  
Math.sqrt(i);
```

```
String s = "Hellu World";  
  
s = s.replace("u", "o");  
  
int len = s.length();
```

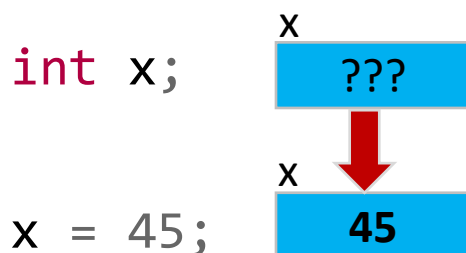
Khai báo & tạo đối tượng

- **Khai báo đối tượng:** `<tên lớp> <tên đối tượng>;`
- Đối tượng cũng là biến \Rightarrow cú pháp giống khai báo biến, trong đó tên lớp đóng vai trò kiểu dữ liệu.
- **Tạo đối tượng:** dùng toán tử `new`: `new <tên lớp>;`
- Một biến đối tượng được gọi là **biến tham chiếu** (reference variable):
 - Bản thân biến **không chứa dữ liệu**, chỉ là một **tham chiếu** đến đối tượng
 - Hay nói cách khác, biến tham chiếu chứa **“địa chỉ”** của đối tượng mà nó tham chiếu đến
 - Muốn cho biến tham chiếu đến đối tượng, dùng phép gán `=`

Khai báo và tạo đối tượng

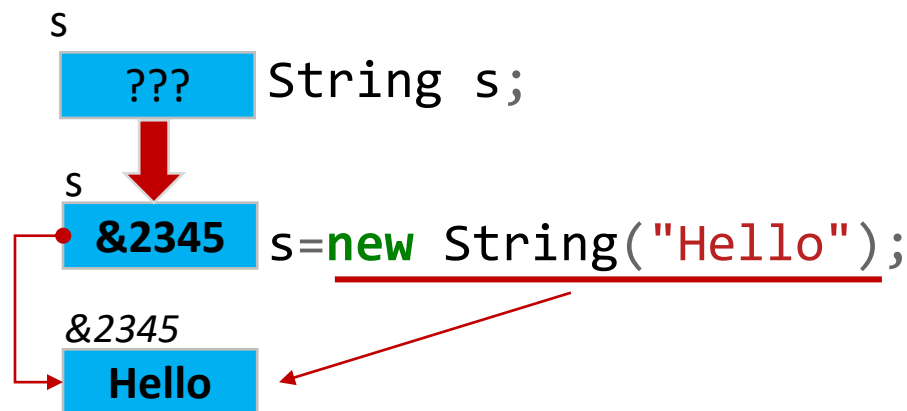
• Biến kiểu nguyên thủy (*primitive datatype variable*)

- Lưu trực tiếp dữ liệu vào vùng nó được cấp phát



• Biến tham chiếu (*reference variable*)

- Lưu địa chỉ của đối tượng mà nó tham chiếu đến



Có thể khai báo ngắn gọn:

`String s = new String("Hello");`

Truy xuất các thành phần của đối tượng

- Một đối tượng được truy xuất thông qua biến tham chiếu đến nó
- Cú pháp: `<biến tham chiếu>.<tên thành phần>`
- Ví dụ:

```
s.replace("u", "o");  
s.toUpperCase();
```
- Lưu ý:
 - Từ bên ngoài lớp, chỉ được truy xuất đến các thành phần `public` hoặc `default` (không khai báo thuộc tính truy cập)
 - Trong cùng một lớp, 1 phương thức có thể truy xuất tất cả các thành phần khác

Tạo lớp căn bản

- Cú pháp tạo lớp:

Phạm vi truy cập
của lớp:
public hoặc
không ghi gì cả
(default)

```
[phạm vi truy cập] class <tên lớp> {  
    // thuộc tính  
  
    // phương thức  
}
```

- Trong một tập tin có thể định nghĩa nhiều lớp, tuy nhiên chỉ được có nhiều nhất một lớp có phạm vi truy cập là **public**
- Các phạm vi truy cập có liên quan đến khái niệm gói (package) nên sẽ được đề cập sau
- Tên lớp: đặt theo qui tắc đặt tên định danh. Thông thường đặt theo qui tắc title case (chữ cái đầu tiên của mỗi từ được viết hoa)

Thuộc tính của lớp

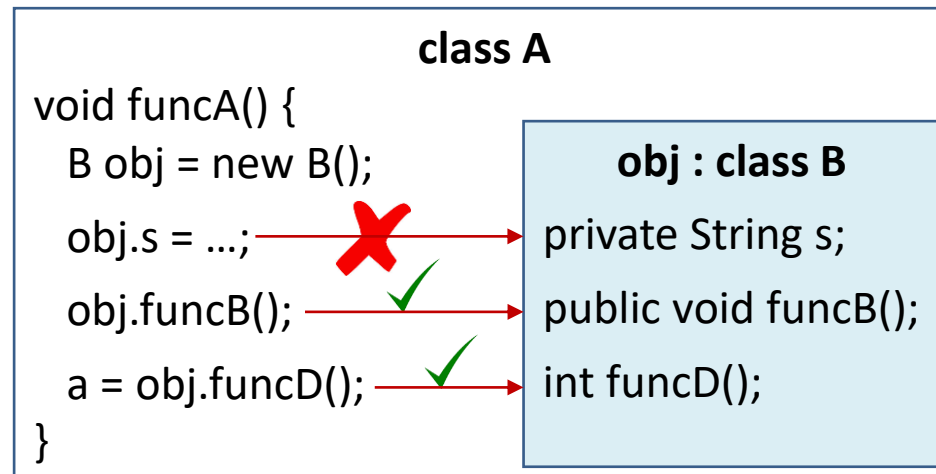
- **Thuộc tính**: tương tự cú pháp khai báo biến

[**phạm vi truy cập**] <**kiểu dữ liệu**> <tên thuộc tính>;

- **Phạm vi truy cập**: xác định thuộc tính có thể được truy cập từ đâu, chỉ bên trong lớp hay có thể từ bên ngoài lớp. Gồm 4 mức độ: **private**, không ghi gì cả (**default**), **protected** và **public**.
- **Kiểu dữ liệu**: có thể là một kiểu dữ liệu nguyên thủy hay lớp.
- **Tên thuộc tính**: theo qui tắc và qui ước đặt tên biến
- **Lưu ý**: các thuộc tính thường được khai báo là **private** và việc truy xuất các thuộc tính sẽ thông qua các phương thức

Phạm vi truy cập của thành phần của lớp

- Chỉ định phạm vi truy cập tới các thành phần của lớp:
 - `public`: có thể truy cập từ bên trong lớp lẫn bên ngoài lớp
 - `private`: chỉ có thể được truy cập từ bên trong lớp
 - `protected`: có thể được truy cập từ bên trong lớp và các lớp con (trong thừa kế)
 - `default` (không khai báo): trong cùng gói (package), tương đương `public`; khác gói, tương đương `private`



Phương thức của lớp

- **Phương thức**: tương tự cú pháp tạo hàm

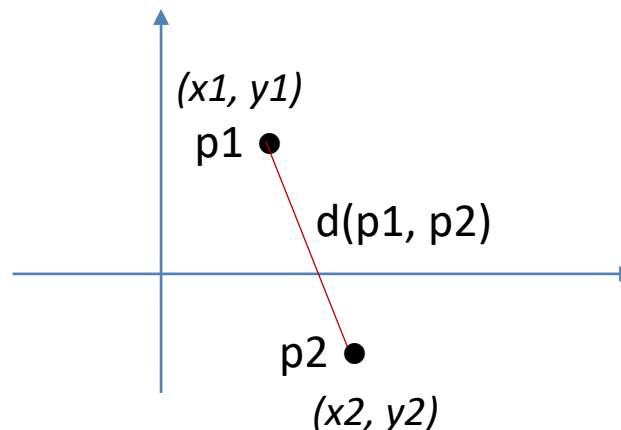
```
[phạm vi truy cập] <kiểu dữ liệu trả về> <tên phương thức> ([các tham số]) {  
    //thân phương thức (hàm)  
}
```

- **Phạm vi truy cập** (access modifier): qui định phương thức có thể được truy cập từ đâu (tương tự như thuộc tính)
- **Kiểu dữ liệu trả về** (return type): kiểu dữ liệu của giá trị trả về của hàm, `void` nếu hàm không trả giá trị về
- Danh sách tham số (parameters): các dữ liệu được truyền vào cho phương thức
- Thân hàm: các lệnh thực hiện tác vụ của hàm

Lớp Diem2D

- Ví dụ 1: Tạo một lớp điểm trong không gian hai chiều (2D)
 - Dữ liệu: mỗi điểm gồm tọa độ x, y
 - Phương thức: thiết đặt giá trị x, y; hiển thị tọa độ ra màn hình; lấy giá trị của x, lấy giá trị của y, nhập giá trị cho x, y; tính khoảng cách đến 1 điểm khác;...

Diem2D
- x: int - y: int
+ ganXY(int, int): void + hienthi(): void + layX(): int + layY(): int + nhap(): void + khoangcach(Diem2D): float ...



Lớp Diem2D

```
public class Diem2D {  
    //thuộc tính  
    private int x;  
    private int y;  
    //phương thức  
    public void ganXY(int h, int t) {  
        /* thân hàm */  
    }  
  
    public void hienthi() {  
        /* thân hàm */  
    }  
  
    public float khoangcach(Diem2D d) {  
        /* thân hàm */  
    }  
    //... Tương tự cho các phương thức khác  
}
```

Diem2D
- x: int - y: int
+ ganXY(int, int): void + hienthi(): void + layX(): int + layY(): int + nhap(): void + khoangcach(Diem2D): float ...

Định nghĩa phương thức

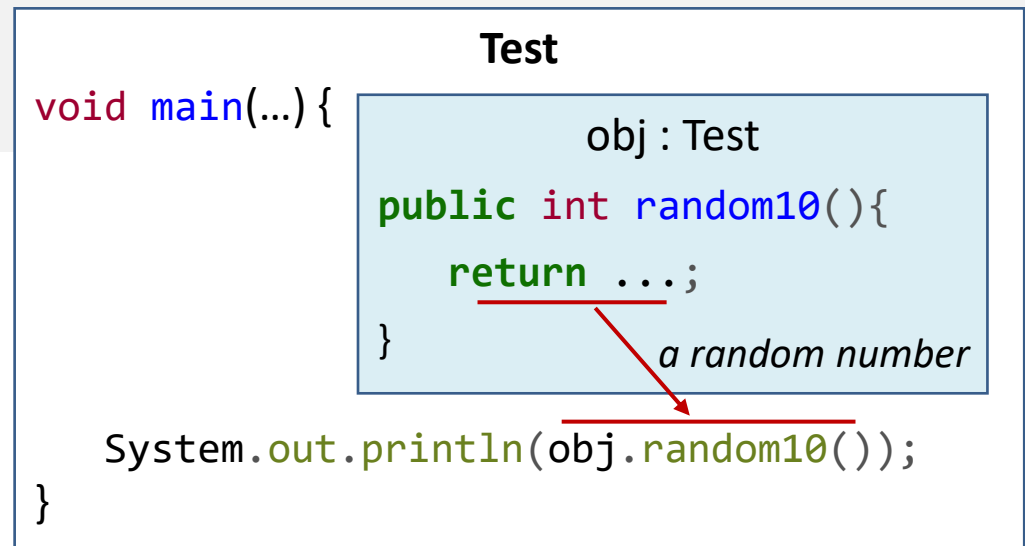
- **Kiểu dữ liệu trả về (return type):**

- Phương thức: thực hiện một tác vụ (task) nào đó
⇒ có thể trả về 1 giá trị
- Lệnh `return` được dùng để trả về 1 giá trị cho lời gọi hàm
- Hàm có thể trả về 1 giá trị có kiểu dữ liệu nguyên thủy hoặc một đối tượng
- Nếu phương thức không trả về giá trị, kiểu dữ liệu trả về phải được khai báo là `void`
- Ví dụ:
 - Hàm `Math.sqrt()` tính căn bậc 2 của 1 số, trả về 1 số thực
⇒ kiểu dữ liệu trả về là `double`: `double sqrt(...)`;
 - Hàm `System.out.println()` hiển thị 1 chuỗi ra màn hình
⇒ không có giá trị trả về: `void println(...)`;

Định nghĩa phương thức

```
public class Test {  
    public static void main(String[] args) {  
        Test obj = new Test();  
        System.out.println(obj.random10());  
    }  
  
    //generate a random value in range of [0, 10)  
    public int random10() {  
        int r = (int)(Math.random() * 10);  
        return r;  
    }  
}
```

- Cú pháp lệnh `return`:
return <biểu thức>;



Định nghĩa phương thức

• Tham số của hàm (parameters):

- Một số hàm cần phải được cung cấp dữ liệu để thực hiện tác vụ
- Ví dụ: hàm tính căn bậc 2 `Math.sqrt()` cần phải được truyền vào một giá trị nguyên/thực để nó có thể tính toán
- **Tham số:** các dữ liệu truyền vào cho hàm
 - Cú pháp khai báo 1 đối số: **<kiểu dữ liệu> <tên tham số>**
 - Các tham số cách nhau bằng dấu phẩy
- Về bản chất, **tham số chính là biến cục bộ** của phương thức
- Ví dụ:
 - `double sqrt(double n)`: hàm tính căn bậc hai, nhận vào một số thực cần tính căn bậc hai
 - `void PTB1(int a, int b)`: hàm giải phương trình bậc 1, nhận 2 tham số kiểu `int`, là hai hệ số `a` và `b`

Định nghĩa phương thức

• Tham số của hàm (tt):

- Tham số hình thức: là các tham số trong định nghĩa hay khai báo hàm
- Tham số thực tế: các biến, biểu thức được truyền vào trong lời gọi hàm

tham số hình thức

```
class Math {  
    public static double sqrt(double n) {  
        //định nghĩa hàm  
    }  
}
```

```
class Test {  
    public static void main(String args[]) {  
        double a = 2.0, b;  
        b = Math.sqrt(a);  
    }  
}
```

tham số thực tế

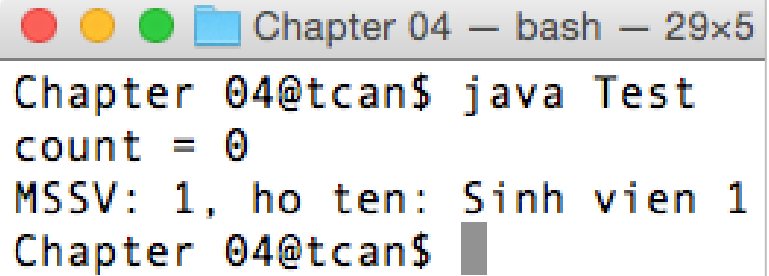
Định nghĩa phương thức

• Tham số của hàm (tt):

- Tham số kiểu dữ liệu nguyên thủy:
 - **Giá trị** của tham số thực tế được truyền vào cho tham số hình thức \Rightarrow **truyền bằng giá trị**
 - Thay đổi giá trị t/số hình thức **không làm thay đổi** giá trị của t/số thực tế
- Tham số kiểu dữ liệu tham chiếu (đối tượng):
 - **Địa chỉ** của đối tượng được tham chiếu bởi t/số thực tế được truyền vào cho t/số hình thức (cũng là một tham chiếu) \Rightarrow **truyền bằng tham chiếu**
 - T/số hình thức & t/số thực tế tham chiếu đến cùng 1 đối tượng \Rightarrow **có thể thay đổi** giá trị (thuộc tính) của đối tượng được tham chiếu bởi tham số thực tế thông qua tham số hình thức \Rightarrow là một cách để **truyền giá trị trả về** cho lời gọi hàm

Định nghĩa phương thức

```
public class Test {  
    public static void main(String[] args) {  
        int count = 0;  
        Sinhvien sv1 = new Sinhvien();  
        test(count, sv1);  
        System.out.println("count = " + count);  
        sv1.hienthi();  
    }  
  
    public static void test(int c, Sinhvien sv) {  
        c = c + 1;  
        sv.mssv = c;  
        sv.hoten = "Sinh vien " + c;  
    }  
}
```



Chapter 04 — bash — 29x5

```
Chapter 04@tcan$ java Test  
count = 0  
MSSV: 1, ho ten: Sinh vien 1  
Chapter 04@tcan$
```

```
class Sinhvien {  
    public int mssv;  
    public String hoten;  
  
    public void hienthi() {  
        System.out.println("MSSV: " + mssv +  
            ", ho ten: " + hoten);  
    }  
}
```

Định nghĩa phương thức

```

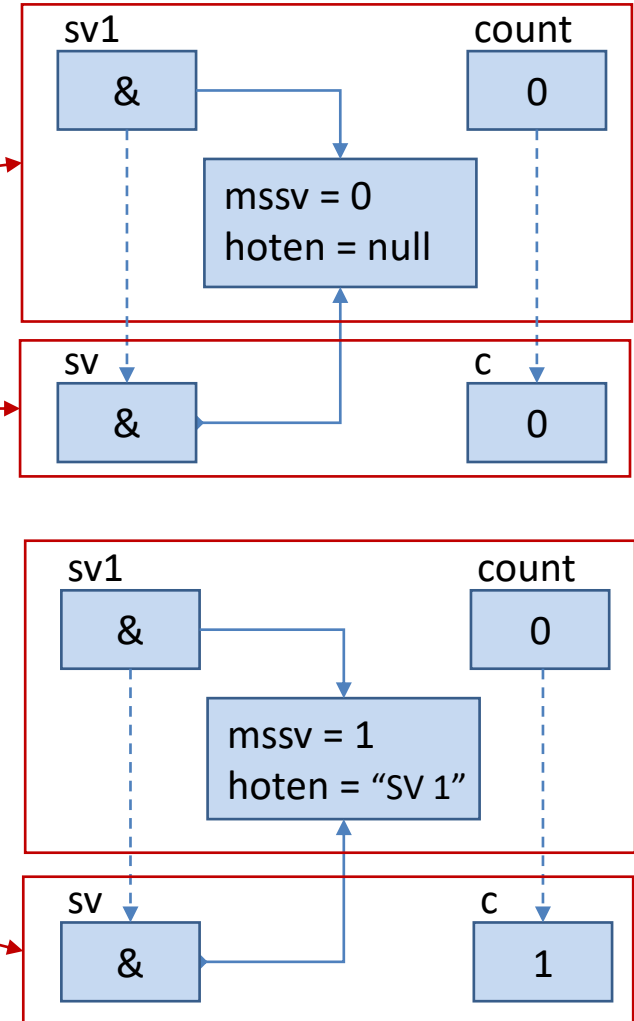
class Test {
    public static void main(String args[]) {
        int count = 0;
        Sinhvien sv1 = new Sinhvien();

        test(count, sv1);

        System.out.println("count = " + count);
        sv1.hienthi();
    }

    public static void test(int c, Sinhvien sv)
    {
        c = c + 1;
        sv.mssv = c;
        sv.hoten = "SV " + c;
    }
}

```



Định nghĩa phương thức

- Ưu điểm của truyền bằng tham chiếu:
 - Tiết kiệm bộ nhớ: tham số kiểu tham chiếu chỉ chứa địa chỉ được truyền vào, không phụ thuộc kích thước của đối tượng
 - Có thể sử dụng để trả về nhiều hơn một giá trị cho lời gọi phương thức
 - Tăng tốc độ gọi hàm: chỉ sao chép tham chiếu chứ không sao chép toàn bộ đối tượng (dữ liệu)
- Lưu ý: việc thay đổi tham chiếu của tham số hình thức không làm thay đổi tham chiếu của tham số thực tế

Định nghĩa phương thức

```
public class Test {  
    public static void main(String[] args) {  
        int count = 0;  
        Sinhvien sv1 = new Sinhvien();  
        test(count, sv1);  
        System.out.println("count = " + count);  
        sv1.hienthi();  
    }  
  
    public static void test(int c, Sinhvien sv) {  
        c = c + 1;  
        sv = new Sinhvien();  
        sv.mssv = c;  
        sv.hoten = "Sinh vien " + c;  
    }  
}
```



```
Chapter 04 — bash — 26x5  
Chapter 04@tcan$ java Test  
count = 0  
MSSV: 0, ho ten: null  
Chapter 04@tcan$
```

Tái định nghĩa phương thức

- Tái định nghĩa phương thức (method overloading): định nghĩa các phương thức trùng tên (cùng chức năng)
- Các phương thức trùng tên phải khác nhau về tham số
 - Khác nhau về số tham số
 - Khác nhau về kiểu tham số
 - Khác nhau về thứ tự các tham số

```
public class Diem2D {  
    public void hienthi() {  
        System.out.println("(" + x + ", " + y + ")");  
    }  
  
    public void hienthi(String s) {  
        System.out.println(s + "(" + x + ", " + y + ")");  
    }  
}
```

Tái định nghĩa phương thức

Hợp lệ:

```
public void func1();  
public void func1(int);  
public void func1(long);  
public void func1(String, int);  
public void func1(int, String);
```

Không hợp lệ:

```
public void func2(String);  
public void func2(String);  
public int func2(String);  
  
public func3(int, long);  
public func3(int, long);
```

Khởi tạo và hủy đối tượng

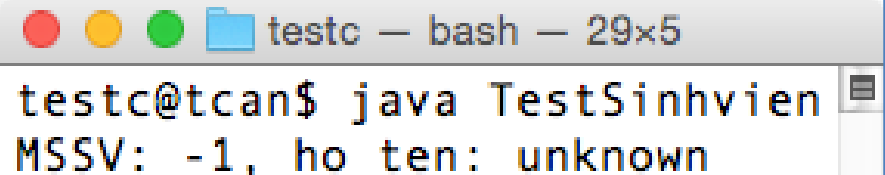
Hàm xây dựng (constructor)

- Là một phương thức đặc biệt của lớp, dùng để khởi tạo đối tượng (khởi tạo các dữ liệu thành viên)
- Có 2 loại:
 - Hàm khởi tạo không có tham số, còn được gọi là hàm xây dựng mặc nhiên (default constructor)
 - Các hàm khởi tạo có tham số
- Cú pháp:
 - Phạm vi truy xuất: public
 - Tên hàm: trùng với tên lớp
 - Kiểu dữ liệu trả về: không có, kể cả void

Hàm xây dựng (constructor)

```
public class Sinhvien {  
    private int mssv;  
    private String hoten;  
  
    public Sinhvien() {  
        mssv = -1;  
        hoten = "unknown";  
    }  
  
    public void hienthi() {  
        System.out.println("MSSV: " + mssv +  
            ", ho ten: " + hoten);  
    }  
}
```

```
class Test {  
    public static void main(String[] args) {  
        Sinhvien sv1 = new Sinhvien();  
        sv1.hienthi();  
    }  
}
```



testc — bash — 29x5

```
testc@tcan$ java TestSinhvien  
MSSV: -1, ho ten: unknown
```

Hàm xây dựng (constructor)

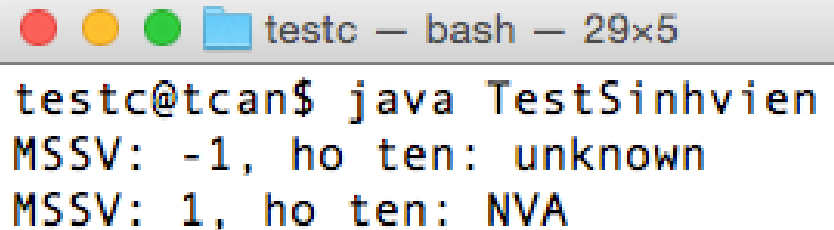
- Một số tính chất:

- Hàm xây dựng sẽ **tự động được gọi** khi đối tượng được tạo ra
- Hàm xây dựng cũng là 1 phương thức của lớp nên nó có thể được **tái định nghĩa** (lưu ý qui tắc tái định nghĩa phương thức)
- Trong trường hợp có nhiều hàm xây dựng, hàm nào được gọi là tùy vào danh sách tham số truyền vào trong câu lệnh tạo đối tượng
- Lớp có bao nhiêu hàm xây dựng, có bấy nhiêu cách để tạo đối tượng
- Hàm xây dựng không thể được gọi trực tiếp

Hàm xây dựng (constructor)

```
class Sinhvien {  
    private int mssv;  
    private String hoten;  
  
    public Sinhvien() {  
        mssv = -1;  
        hoten = "unknown";  
    }  
  
    public Sinhvien(int ms, String ht) {  
        mssv = ms;  
        hoten = ht;  
    }  
  
    public void hienthi() {  
        System.out.println("MSSV: " + mssv +  
            ", ho ten: " + hoten);  
    }  
}
```

```
class TestSinhvien {  
    public static void main(String[] args) {  
        Sinhvien sv1 = new Sinhvien();  
        Sinhvien sv2 = new Sinhvien(1, "NVA");  
        sv1.hienthi();  
        sv2.hienthi();  
    }  
}
```

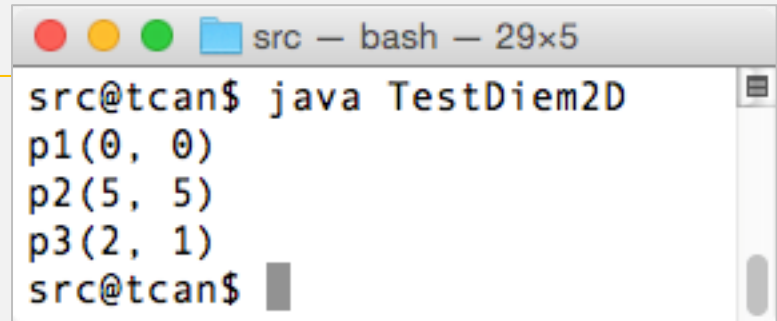


```
testc@tcan$ java TestSinhvien  
MSSV: -1, ho ten: unknown  
MSSV: 1, ho ten: NVA
```


Hàm xây dựng (constructor)

```
class Diem2D {  
    private int x, y;  
  
    public Diem2D() {  
        x = y = 0;  
    }  
  
    public Diem2D(int th) {  
        x = y = th;  
    }  
  
    public Diem2D(int h, int t) {  
        x = h;    y = t;  
    }  
  
    public void hienthi(String s) {  
        System.out.println(s + "(" + x + ", " + y + ")");  
    }  
}
```

```
class TestDiem2D {  
    public static void main(String args[]) {  
        Diem2D p1 = new Diem2D();  
        Diem2D p2 = new Diem2D(5);  
        Diem2D p3 = new Diem2D(1, 2);  
  
        p1.hienthi("p1");  
        p2.hienthi("p2");  
        p3.hienthi("p3");  
    }  
}
```

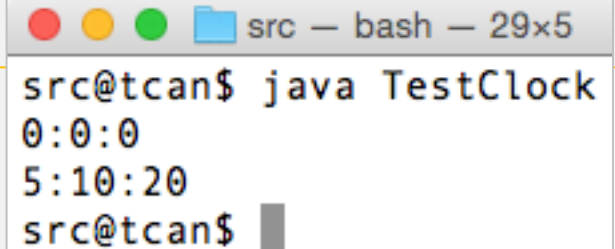


```
src@tcan$ java TestDiem2D  
p1(0, 0)  
p2(5, 5)  
p3(2, 1)  
src@tcan$
```

Hàm xây dựng (constructor)

```
class Clock {  
    private int h, m, s;  
  
    public Clock() {  
        h = m = s = 0;  
    }  
  
    public Clock(int g, int p, int gi) {  
        h = g; m = p; s = gi;  
    }  
  
    public void setTime(int g, int p, int gi) {  
        h = g; m = p; s = gi;  
    }  
  
    public void display() {  
        System.out.println(h + ":" + m + ":" + s);  
    }  
}
```

```
public class TestClock {  
    public static void main(String args[]) {  
        Clock c1 = new Clock();  
        Clock c2 = new Clock(5, 10, 20);  
        c1.display();  
        c2.display();  
    }  
}
```



```
src — bash — 29x5  
src@tcan$ java TestClock  
0:0:0  
5:10:20  
src@tcan$
```

Hàm xây dựng sao chép (copy constructor)

- Là hàm xây dựng với đối số là một đối tượng cùng lớp
- Được sử dụng để tạo 1 đối tượng mới “giống” với 1 đối tượng đã có sẵn
- Cú pháp: `public Classname(Classname obj);`
 - Đây cũng là hàm xây dựng nên cú pháp cũng theo qui tắc hàm xây dựng: tên hàm trùng tên lớp, không có giá trị trả về
 - Hàm luôn có 1 đối số, chính là 1 đối tượng cùng lớp

```
//hàm xây dựng sao chép của Lớp Clock  
public Clock(Clock c) {  
    h = c.h;  
    m = c.m;  
    s = c.s;  
}
```

```
Clock c1 = new Clock(5, 10, 20);  
Clock c2 = new Clock(c1);
```

Phép gán đối tượng (object assignment)

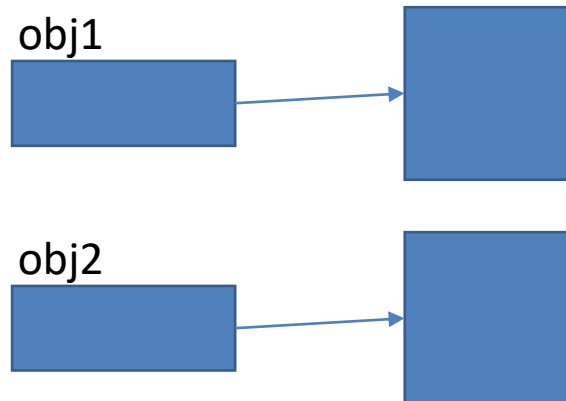
- Phép gán đối tượng còn được gọi là phép sao chép cạn (shadow copy):

- Cái được **sao chép** chính là **tham chiếu**: phép gán

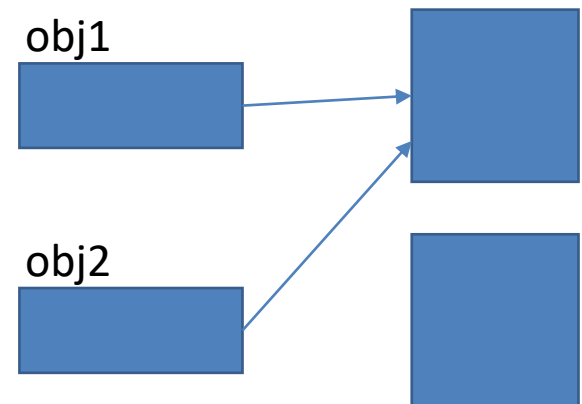
$obj1 = obj2$

làm cho $obj1$ và $obj2$ **tham chiếu đến cùng 1 đối tượng**

- Điều này có thể dẫn đến các lỗi ngoài mong đợi trong chương trình: $obj1$ và $obj2$ luôn có cùng 1 giá trị



$obj2 = obj1$



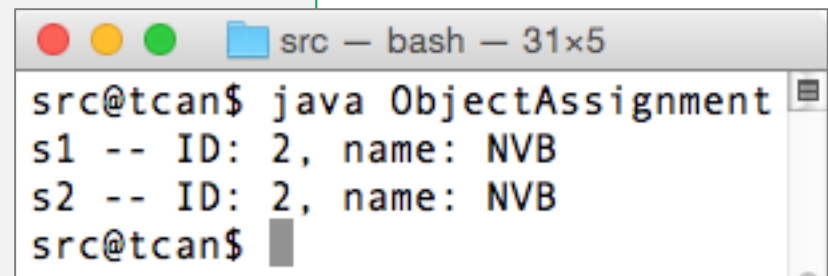
Phép gán đối tượng (object assignment)

```
class Student {  
    int ID;  
    String name;  
  
    public Student(int i, String n) {  
        ID = i;  
        name = n;  
    }  
}
```

```
class ObjectAssignment {  
    public static void main(String []args) {  
        Student s1 = new Student(1, "NVA"), s2;  
        s2 = s1;  
        s2.setInfo(2, "NVB");  
        s1.display("s1");  
        s2.display("s2");  
    }  
}
```

```
    public void display(String s) {  
        System.out.println(s + " -- ID: " + ID +  
            ", name: " + name);  
    }  
}
```

```
    public void setInfo(int i, String n) {  
        ID = i;  
        name = n;  
    }  
}
```



```
src — bash — 31x5  
src@tcan$ java ObjectAssignment  
s1 -- ID: 2, name: NVB  
s2 -- ID: 2, name: NVB  
src@tcan$
```

Hàm hủy (desctructor)

- Là một phương thức đặc biệt của đối tượng, sẽ tự động được gọi bởi bộ thu hồi rác (garbage collector) khi ra khỏi phạm vi của đối tượng
- Trong Java, hàm hủy được gọi là **finalizer** (hàm kết thúc)
- Cú pháp hàm kết thúc:
 - Không có đối số và kiểu dữ liệu trả về, kể cả `void`
 - Một lớp chỉ được phép có tối đa 1 hàm kết thúc
 - Tên của hàm kết thúc: `finalizer`
- Trong Java, thông thường hàm này ít được sử dụng

Thêm về lớp và đối tượng *(more on class & object)*

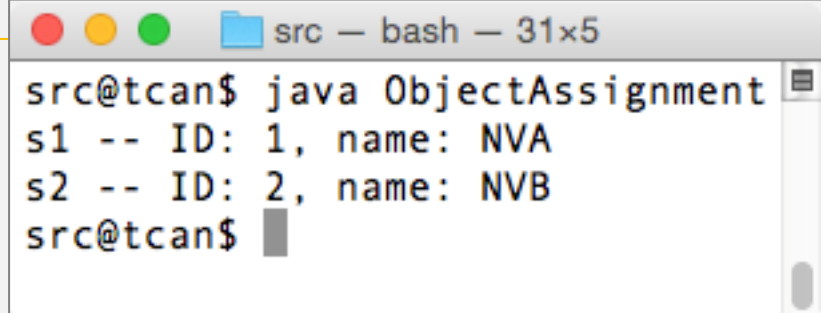
Sao chép sâu (deep copy)

- Muốn thực hiện sao chép dữ liệu của đối tượng thay vì tham chiếu, ta phải tự viết các phương thức để sao chép dữ liệu.
- Thao tác **sao chép dữ liệu** của đối tượng được gọi là **sao chép sâu** (deep copy)
- Ví dụ: để thực hiện sao chép hai đối tượng `s1` và `s2` trong ví dụ trước, ta phải viết thêm phương thức `makeCopy()` như sau trong lớp `Student`:
 - `void makeCopy(Student);`
 - Để sao chép `s1` cho `s2`, ta gọi: `s2.makeCopy(s1)`

Sao chép sâu (deep copy)

```
class Student {  
  
    //dữ liệu  
    //các phương thức  
  
    public void makeCopy(Student s) {  
        ID = s.ID;  
        name = s.name;  
    }  
}
```

```
class ObjectAssignment {  
    public static void main(String []args) {  
        Student s1 = new Student(1, "NVA");  
        Student s2 = new Student();  
  
        s2.makeCopy(s1);  
        s2.setInfo(2, "NVB");  
  
        s1.display("s1");  
        s2.display("s2");  
    }  
}
```

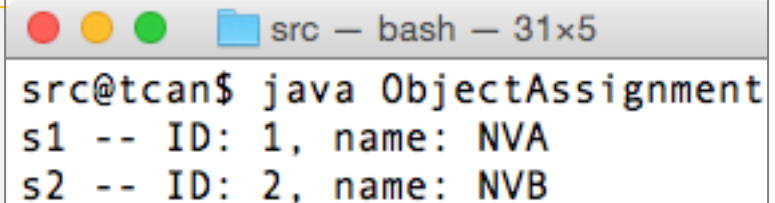


```
src@tcn$ java ObjectAssignment  
s1 -- ID: 1, name: NVA  
s2 -- ID: 2, name: NVB  
src@tcn$
```

Sao chép sâu (deep copy)

- Một phương pháp khác để sao chép sâu dùng chính phép gán `=` là tạo một đối tượng mới giống với đối tượng cần được sao chép, sau đó sử dụng phép gán

```
class ObjectAssignment {  
    public static void main(String []args) {  
        Student s1 = new Student(1, "NVA");  
        Student s2 = new Student();  
  
        s2 = s1.getCopy();  
        s2.setInfo(2, "NVB");  
  
        s1.display("s1");  
        s2.display("s2");  
    }  
}
```




```
src@tcan$ java ObjectAssignment  
s1 -- ID: 1, name: NVA  
s2 -- ID: 2, name: NVB
```

```
class Student {  
  
    //dữ liệu  
    //các phương thức  
  
    public Student getCopy() {  
        Student t = new Student(ID, name);  
        return t;  
    }  
}
```

Thành phần tĩnh (static members)

- Lớp là khuôn mẫu để tạo ra đối tượng
 - Các thành phần khai báo trong lớp chỉ tồn tại khi đối tượng được tạo ra
 - Mỗi đối tượng sẽ “sở hữu” các thành phần riêng của mình
⇒ có bao nhiêu đối tượng được tạo ra thì có bấy nhiêu “tập” thành phần trên bộ nhớ

Student		<u>st1:Student</u>		<u>st2:Student</u>	
- int ID		- ID	1	- ID	2
- String name		- name	NVA	- name	NVB
+ Student() + Student() + display(): void + setInfo(int, String): void		+ Student() + Student() + display(): void + setInfo(int, String): void		+ Student() + Student() + display(): void + setInfo(int, String): void	

Thành phần tĩnh (static members)

- Thành phần tĩnh:
 - Còn được gọi là thành phần “của lớp”
 - Tồn tại độc lập với các đối tượng: có bao nhiêu đối tượng được tạo ra thì vẫn chỉ có 1 bản (copy) của thành phần tĩnh trong bộ nhớ
 - Thành phần tĩnh được truy xuất trực tiếp thông qua lớp
`<tên lớp>.<tên thành phần>`
 - Các phương thức tĩnh chỉ được truy xuất các thành phần tĩnh
 - Các phương thức không tĩnh có thể truy xuất tất cả các thành phần tĩnh lẫn không tĩnh
 - Các thành phần tĩnh sẽ được khởi tạo các giá trị mặc nhiên của kiểu dữ liệu của thành phần tĩnh

Thành phần tĩnh (static members)

```
class ClockStatic {
    private static int count;
    private int h, m, s;

    public ClockStatic() {
        h = m = s = 0;
        count++;
    }

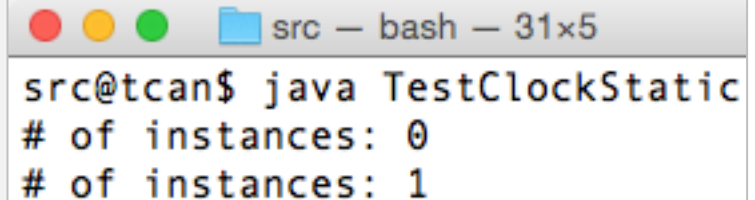
    public ClockStatic(int g, int p, int gi) {
        h = g; m = p; s = gi;
        count++;
    }

    static public void noOfInstances() {
        System.out.println("# of instances: " +
            ClockStatic.noOfInstances());
    }

    public void display() {
        System.out.println(h + ":" + m + ":" + s);
    }
}
```

```
class TestClockStatic {
    public static void main(String args[]) {
        ClockStatic.noOfInstances();

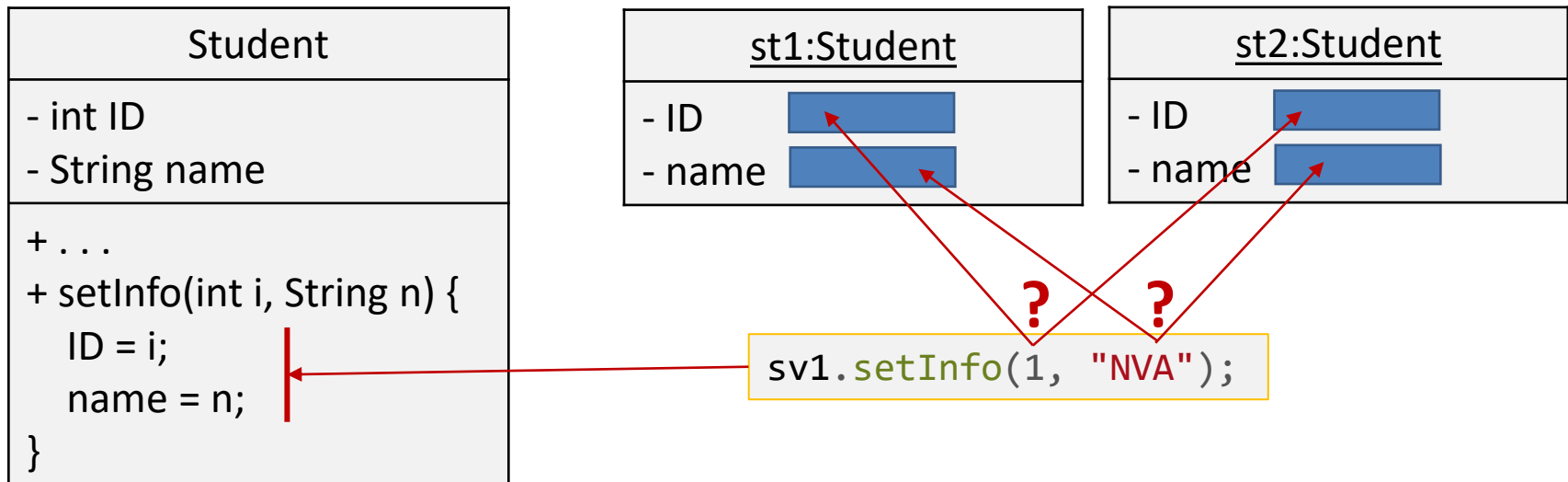
        ClockStatic c1 = new ClockStatic();
        ClockStatic.noOfInstances();
    }
}
```



```
src — bash — 31x5
src@tcan$ java TestClockStatic
# of instances: 0
# of instances: 1
```

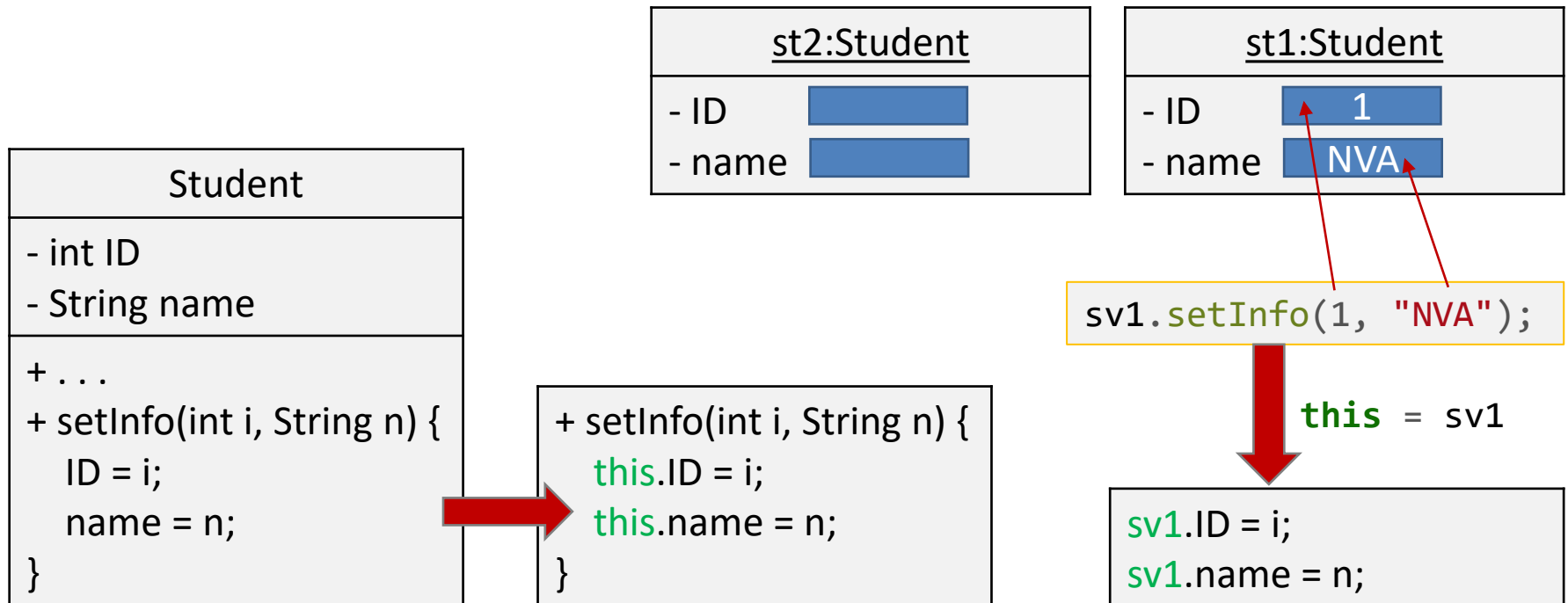
Biến tham chiếu `this`

- Đây là một **biến tham chiếu** đặc biệt, tồn tại trong tất cả **các phương thức không tĩnh** của lớp
- Biến này tham chiếu đến đối tượng đang gọi phương thức



Biến tham chiếu `this`

- Trong tất cả các phương thức **không tĩnh**, tất cả các lệnh truy xuất vào các dữ liệu thành viên không tĩnh sẽ **được tự động thêm vào** tham chiếu `this`:



Biến tham chiếu `this`

- Biến tham chiếu `this` có thể được sử dụng một cách tường minh để **tránh trùng tên**

```
class Student {  
    private int ID;  
    private String name;
```

```
//...
```

```
public void setInfo(int ID, String name) {  
    ID = ID;  
    name = name;  
}
```

```
public void setInfo(int ID, String name) {  
    this.ID = ID;  
    this.name = name;  
}
```

Hai tham số của phương thức sẽ che đi các dữ liệu thành viên

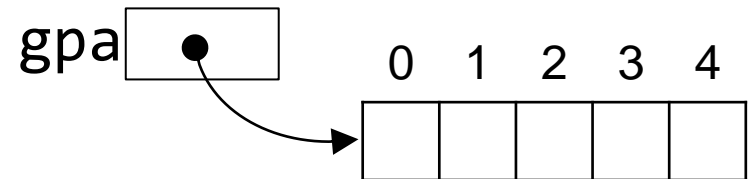
Mảng đối tượng

Mảng (array)

- Là một dãy các phần tử (giá trị) có cùng kiểu dữ liệu
- Là một biến **kiểu tham chiếu**:

- Khai báo: `<kiểu dữ liệu> <tên mảng>[];`
- Tạo mảng: `<tên mảng> = new <kiểu dữ liệu>;`

```
double gpa[];  
gpa = new double[5];
```



- Có thể vừa khai báo vừa khởi tạo

```
int []number = {2, 4, 6, 8};  
String []monthName = {"Jan", "Feb", "March",...};
```

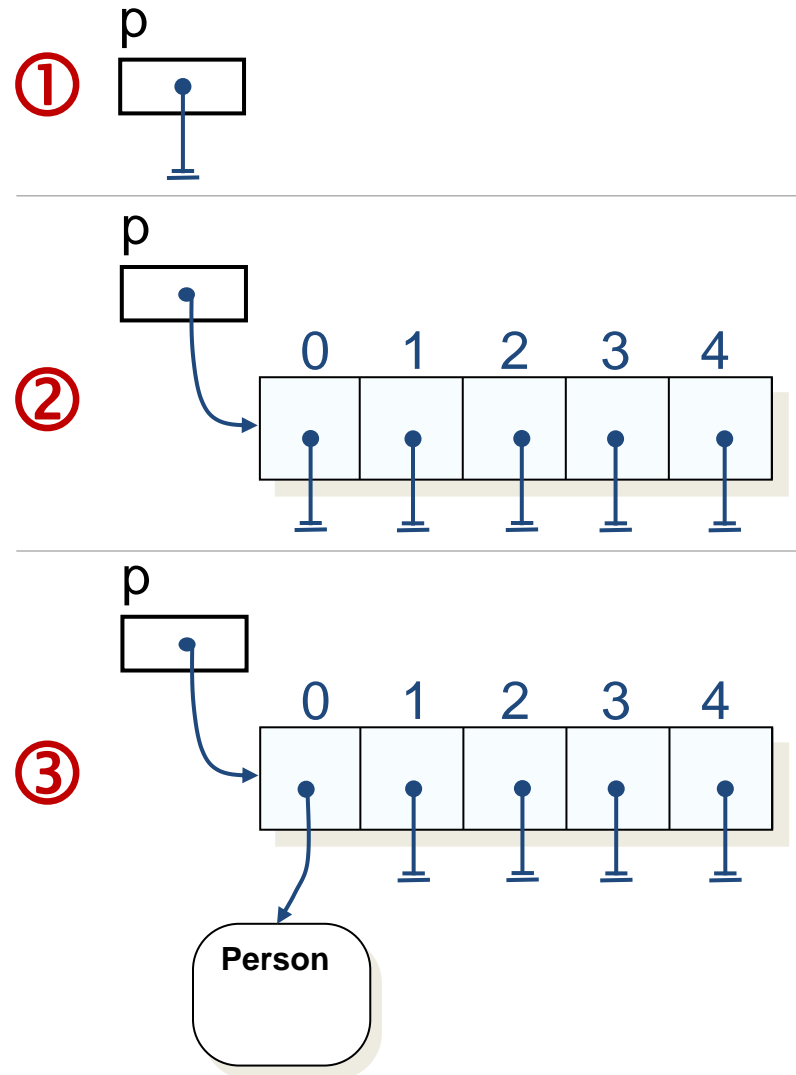
Mảng (array)

• Mảng các đối tượng:

```
Person []prs;           ①
prs = new Person[5];    ②
prs[0] = new Person();  ③
//...
```

```
prs[0].setName("Tommy");
prs[0].setAge(20);
```

```
for (Person t : prs) {
    System.out.println(t.getName());
}
```



Xử lý ngoại lệ

Xử lý ngoại lệ (exception handling)

- Ngoại lệ: là một lỗi xảy ra khi thực thi chương trình:
 - Ví dụ: chia cho 0, mở tập tin không tồn tại,...
- Khi một lỗi xảy ra, một **đối tượng** ngoại lệ sẽ sinh ra và chương trình phải “bắt” và xử lý ngoại lệ
- Bắt ngoại lệ: dùng câu lệnh `try...catch...finally`
- Xử lý ngoại lệ:
 - Truyền tiếp ngoại lệ: “quảng” ngoại lệ cho nơi khác trong chương trình xử lý
 - Khắc phục lỗi để chương trình có thể tiếp tục

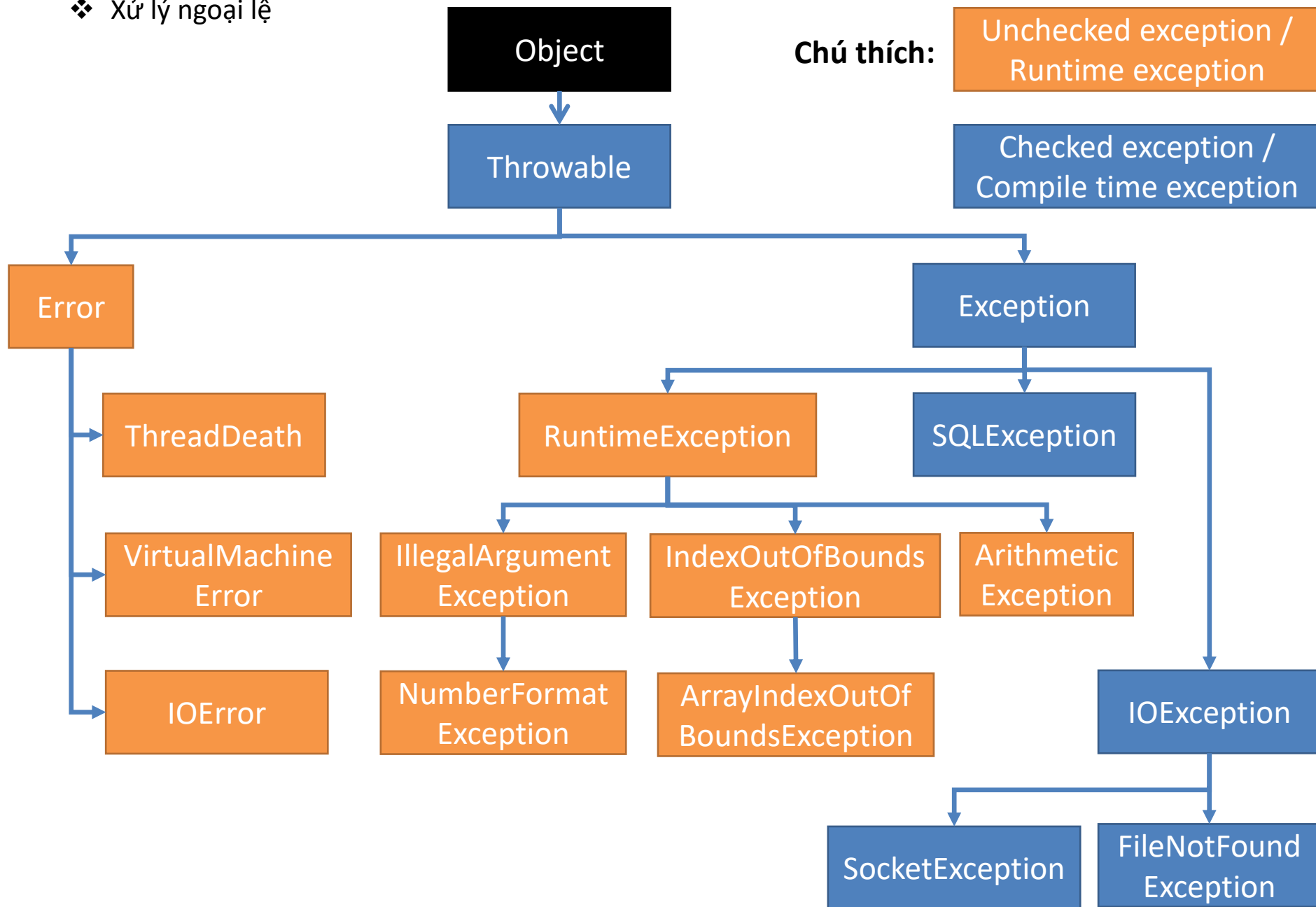
Xử lý ngoại lệ (exception handling)

- Bắt ngoại lệ:

```
try {  
    //các câu lệnh có khả năng  
    //sinh ra lỗi  
}  
catch (<loại ngoại lệ 1>) {  
    //xử lý ngoại lệ 1  
}  
...  
catch (<loại ngoại lệ n>) {  
    //xử lý ngoại lệ n  
}  
finally {  
    //các lệnh sẽ được thực hiện  
    //cả trong trường hợp có và  
    //không có ngoại lệ xảy ra  
}
```

- Lưu ý:

- Khối lệnh **finally** là không bắt buộc
- Các loại ngoại lệ trong Java được tổ chức theo dạng cây phân cấp với gốc cây phân cấp là **Exception**



Xử lý ngoại lệ (exception handling)

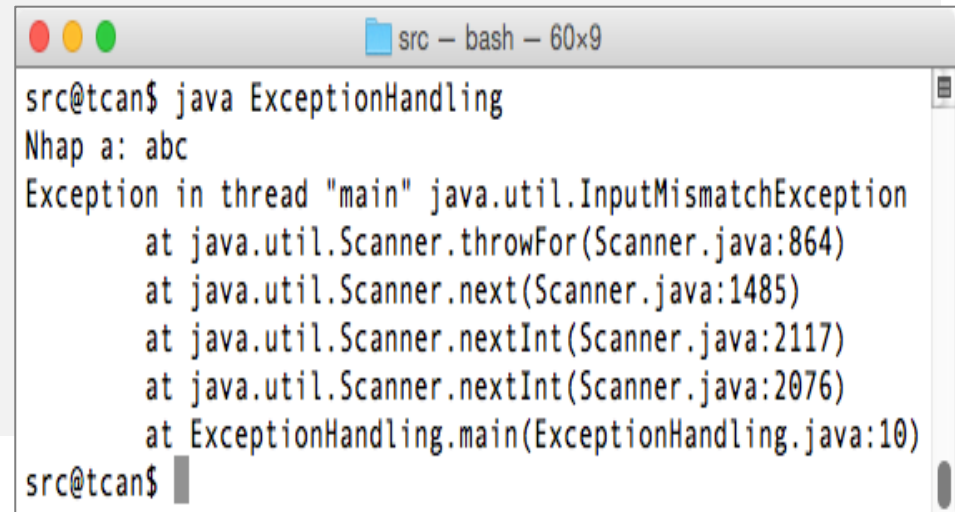
```
import java.util.Scanner;
import java.util.InputMismatchException;

class ExceptionHandling {
    public static void main(String args[]) {
        int a, b, x;
        Scanner s = new Scanner(System.in);

        System.out.print("Nhập a: ");
        a = s.nextInt();
        s.nextLine();

        System.out.print("Nhập b: ");
        b = s.nextInt();
        s.nextLine();

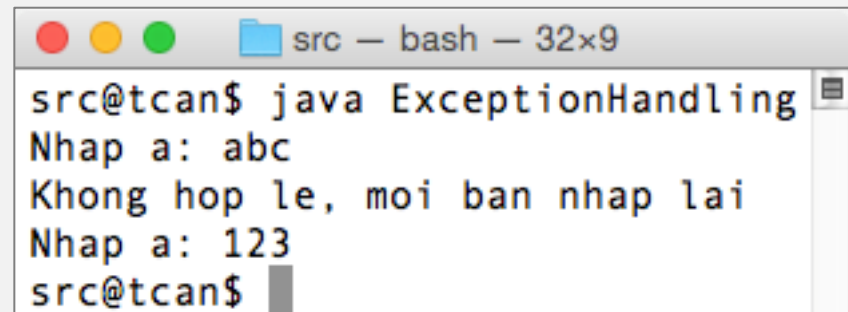
        //...
    }
}
```



```
src - bash - 60x9
src@tcan$ java ExceptionHandling
Nhập a: abc
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:864)
    at java.util.Scanner.next(Scanner.java:1485)
    at java.util.Scanner.nextInt(Scanner.java:2117)
    at java.util.Scanner.nextInt(Scanner.java:2076)
    at ExceptionHandling.main(ExceptionHandling.java:10)
src@tcan$
```


Xử lý ngoại lệ (exception handling)

```
import java.util.Scanner;
import java.util.InputMismatchException;
class ExceptionHandling {
    public static void main(String args[]) {
        int a, b, x;
        Scanner s = new Scanner(System.in);
        while (true) {
            try {
                System.out.print("Nhập a: ");
                a = s.nextInt();
            }
            catch (InputMismatchException e) {
                System.out.println("Không hợp lệ, mời bạn nhập lại");
                continue;
            }
            finally {
                s.nextLine();
            }
            break;
        } //while
        //tiếp tục nhập b, ...
    } //main
}
```



The screenshot shows a terminal window titled "src — bash — 32x9". The prompt is "src@tcan\$". The user enters "java ExceptionHandling". The program outputs "Nhập a: abc". The user then enters "123". The program outputs "Không hợp lệ, mời bạn nhập lại" and then "Nhập a: 123". The prompt "src@tcan\$" is visible again.

Gói trong Java (package)

Gói (package)

- Tổ chức và lưu trữ các lớp trong chương trình theo cấu trúc phân cấp
- Thể hiện ưu điểm khi số lượng lớp trong chương trình lớn
- Hạn chế việc xung đột tên: các lớp trong các gói khác nhau thì có thể trùng tên với nhau
- Cho phép bảo vệ lớp và thành phần của lớp theo đơn vị gói (thay vì chỉ bảo vệ theo đơn vị lớp)

Tạo gói

- Chọn tên gói
 - Mang ý nghĩa gói:
 - Ví dụ: util (tiện ích), graphics (đồ họa),...
 - Định danh được tổ chức hay cá nhân phát triển gói:
 - Ví dụ: tất cả các thư viện do tổ chức Apache Software Foundation phát triển đều được đặt trong gói với tiếp đầu ngữ là *org.apache*
- Tạo cấu trúc thư mục
 - Mỗi gói tương ứng với một thư mục
 - Gói con chính là một thư mục con nằm trong thư mục tương ứng với gói cha
 - IDE tự tạo cấu trúc thư mục hoặc người lập trình tự tạo cấu trúc thư mục

Apache
(apache.org)

Các lớp thông dụng
org.apache.commons.*

Các lớp hỗ trợ tìm kiếm văn bản
org.apache.lucene.*

Các lớp hỗ trợ lập trình phân tán
org.apache.hadoop.*

Các cấu trúc dữ liệu tập hợp
org.apache.commons.collection.*

Các lớp hỗ trợ tính toán, thống kê
org.apache.commons.math.*

/ (thư mục gốc của dự án)

org/

apache/

commons/

các lớp thuộc gói org.apache.commons

collection/

các lớp thuộc gói org.apache.commons.collection

math/

các lớp thuộc gói org.apache.commons.math

lucence/

các lớp thuộc gói org.apache.lucene

hadoop/

các lớp thuộc gói org.apache.hadoop

Sử dụng gói

- *Định danh đầy đủ* của lớp bao gồm *tên gói* và *tên lớp*
 - Ví dụ: lớp Math trong gói java.util thì định danh đầy đủ là java.util.Math
- Khi sử dụng một lớp khác gói
 - Sử dụng *định danh đầy đủ* của lớp: code rườm rà
 - Ví dụ:
java.util.Scanner kb = new java.util.Scanner(System.in);
 - Dùng lệnh *import* để khai báo việc sử dụng những gói khác
 - Ví dụ:
import java.util.Scanner; // import từng lớp
import java.util.*; // import tất cả các lớp của gói

Biên dịch và thực thi chương trình

- IDE tự động cấu hình việc biên dịch và thực thi
- Người lập trình tự biên dịch:
 - Việc biên dịch và thực thi phải thực hiện từ thư mục gốc của dự án

ct176/ (thư mục gốc của dự án)
ch2/
 Diem.java
ch3/
 DoanThang.java
 SDDoanThang.java

```
// Diem.java
package ch2;
public class Diem {
    private int x, y;
    //...
}
```

```
// DoanThang.java
package ch3;
import ch2.Diem;
public class DoanThang {
    private Diem dau, cuoi;
    //...
}
```

```
// SDDoanThang.java
package ch3;
public class SDDoanThang {
    public static void main(String []args) {
        DoanThang d;
        // ...
    }
}
```

Phạm vi truy cập

Phạm vi truy cập	Cùng lớp	Cùng gói	Lớp con	Khác lớp
public	Được	Được	Được	Được
protected	Được	Được	Được	Không
default	Được	Được	Không	Không
private	Được	Không	Không	Không

Tóm tắt

- Tạo lớp
 - Hàm xây dựng: mặc nhiên, có tham số, sao chép
 - Các setter và các getter
 - Hàm nhập, hàm hiển thị
 - Hàm sao chép
 - Tái định nghĩa hàm
 - Từ khóa this, static
- Sử dụng lớp
 - Tạo đối tượng
 - Thao tác trên đối tượng
 - Mảng các đối tượng



Question?

CT176 – LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Phụ lục

Lớp String

- Trữ các chuỗi và cung cấp các thao tác trên chuỗi

```
String s = "Hello";
```

```
String s = new String("Hello");
```

- Không thể thay đổi giá trị của chuỗi (immutable)

- Các phương thức của lớp này thường trả về 1 chuỗi

```
s = s.concat("World");
```

```
s = s.replace("W", " W");
```

Lớp String

- `char charAt(int index)`
- `int length()`
- `String substring(int beginIndex)`
- `String substring(int beginIndex, int endIndex)`
- `boolean contains(CharSequence s)`
- `boolean equals(Object another)`
- `boolean isEmpty()`
- `String concat(String str)`
- `String replace(String oldStr, String newStr)`
- `String trim()`
- `int indexOf(String substring)`
- `int indexOf(String substring, int fromIndex)`
- `String toLowerCase()`
- `String toUpperCase()`

Lớp Math

- Cung cấp các hàm toán học
- Hầu hết các phương thức là tĩnh (static, gọi từ lớp)
- Một số hàm thông dụng như:
 - `double/float/int/long abs(double/float/int/long);`
 - `double log/log10(double);`
 - `long/int round(double/float);`
 - `double sqrt(double);`
 - `double random();`
- Một số hàm khác: `sin`, `cos`, `asin`, `acos`, `exp`, `floor`, `ceil`, `pow`, `min`, `max`