Hướng đối tượng trong Java

Nội dung

- Các khái niệm cơ bản về lớp, đối tượng.
- Lớp và đối tượng trong java.
- Tính đóng gói.
- Tính kế thừa.
- Tính đa hình.
- Interface.

Các khái niệm cơ bản

- **Đối tượng (object)**: trong thế giới thực khái niệm đối tượng có thể xem như một thực thể: người, vật, bảng dữ liệu,…
  - ✓ Đối tượng giúp hiểu rõ thế giới thực
  - ✓ Cơ sở cho việc cài đặt trên máy tính
  - ✓ Mỗi đối tượng có định danh, thuộc tính, hành vi

  Ví dụ: đối tượng sinh viên

  MSSV: “TH0701001”; Tên sinh viên: “Nguyễn Văn A”
- **Hệ thống các đối tượng**: là 1 tập hợp các đối tượng
  - ✓ Mỗi đối tượng đảm trách 1 công việc
  - ✓ Các đối tượng có thể trao đổi thông tin với nhau
  - ✓ Các đối tượng có thể xử lý song song, hay phân tán

Các khái niệm cơ bản

- **Lớp (class)**: là khuôn mẫu (template) để sinh ra đối tượng. Lớp là sự trừu tượng hóa của tập các đối tượng có các thuộc tính, hành vi tương tự nhau, và được gom chung lại thành 1 lớp.
- ***Ví dụ***: lớp các đối tượng ***Sinhviên***
  - ✓ Sinh viên "Nguyễn Văn A", mã số TH0701001 → 1 đối tượng thuộc lớp ***Sinhviên***
  - ✓ Sinh viên "Nguyễn Văn B", mã số TH0701002 → là 1 đối tượng thuộc lớp ***Sinhviên***

- **Đối tượng (object) của lớp**: một đối tượng cụ thể thuộc 1 lớp là 1 thể hiện cụ thể của 1 lớp đó.

Lớp và đối tượng trong Java

Khai báo lớp

class *<ClassName>*

{

<danh sách thuộc tính>

<các khởi tạo>

<danh sách các phươngthức>

}

Lớp và đối tượng trong Java (tt)

Thuộc tính: các đặc điểm mang giá trị của đối tượng, là vùng dữ liệu được khai báo bên trong lớp

class *<ClassName>*

{

<Tiền tố> <kiểu dữ liệu> <tên thuộc tính>;

}

Kiểm soát truy cập đối với thuộc tính

* **public**: có thể truy xuất từ bất kỳ 1 lớp khác.
* **protected**: có thể truy xuất được từ những lớp con.
* **private**: không thể truy xuất từ 1 lớp khác.

Lớp và đối tượng trong Java (tt)

Phương thức: chức năng xử lý, hành vi của các đối tượng.

```
class <ClassName> {
    …
    <Tiền tố> <kiểu trả về> <tên phương thức>(<các đối số>){
     …
    }
}
```

Lớp và đối tượng trong Java (tt)

- **public**: có thể truy cập được từ bên ngoài lớp khai báo.
- **protected**: có thể truy cập được từ lớp khai báo và các lớp dẫn xuất (lớp con).
- **private**: chỉ được truy cập bên trong lớp khai báo.
- **static**: phương thức lớp dùng chung cho tất cả các thể hiện của lớp, có thể được thực hiện kể cả khi không có đối tượng của lớp
- **final**: không được khai báo chồng ở các lớp dẫn xuất.
- **abstract**: không có phần source code, sẽ được cài đặt trong các lớp dẫn xuất.
- **synchronized**: dùng để ngăn những tác động của các đối tượng khác lên đối tượng đang xét trong khi đang đồng bộ hóa. Dùng trong lập trình multithreads.

Lớp và đối tượng trong Java (tt)

Ví dụ 1: ***class*** *Sinhvien {*

```
// Danh sách thuộc tính
String      maSv, tenSv, dcLienlac;
int    tuoi;
…
// Danh sách các khởi tạo
Sinhvien(){}
Sinhvien (…)      { …}
…
// Danh sách các phương thức
public void capnhatSV (…) {…}
public void xemThongTinSV() {…}
…
}
```

Lớp và đối tượng trong Java (tt)

```
…
// Tạo đối tượng mới thuộc lớp Sinhvien
Sinhvien sv = new Sinhvien();
…
// Gán giá trị cho thuộc tính của đối tượng
sv.maSv = “TH0601001” ;
sv.tenSv = “Nguyen Van A”;
sv.tuoi = “20”;
sv.dcLienlac = “KP6, Linh Trung, Thu Duc”;
…
// Gọi thực hiện phương thức
sv.xemThongTinSV();
```

Lớp và đối tượng trong Java (tt)

Ví dụ 2:

```
class Sinhvien {
    // Danh sách thuộc tính
    private String      maSv;
    String      tenSv, dcLienlac;
    int     tuoi;
    …
}
…
Sinhvien sv = new Sinhvien();
sv.maSv = "TH0601001"; /* Lỗi truy cập thuộc tính
    private từ                    bên ngoài lớp khai báo */
Sv.tenSv = "Nguyen Van A";
…
```

Lớp và đối tượng trong Java (tt)

Khởi tạo (constructor): là một loại phương thức đặc biệt của lớp, dùng để khởi tạo một đối tượng.

- Dùng để khởi tạo giá trị cho các thuộc tính của đối tượng.
- Cùng tên với lớp.
- Không có giá trị trả về.
- Có thể có tham số hoặc không.

Lưu ý: Mỗi lớp sẽ có 1 constructor mặc định (nếu ta không khai báo constructor nào). Ngược lại nếu ta có khai báo 1 constructor khác thì constructor mặc định chỉ dùng được khi khai báo tường minh.

Lớp và đối tượng trong Java (tt)

Ví dụ 1

```
class Sinhvien
{
    …
    // Không có định nghĩa constructor nào
}
…
// Dùng constructor mặc định
Sinhvien sv = new Sinhvien();
```

Lớp và đối tượng trong Java (tt)

Ví dụ 2:

```
class Sinhvien
{
    …
    // không có constructor mặc định
    Sinhvien(<các đối số>) {…}
}
…
Sinhvien sv = new Sinhvien();
// lỗi biên dịch

class Sinhvien
{
    …
    // khai báo constructor mặc định
    Sinhvien(){}
    Sinhvien(<các đối số>) {…}
}
…
Sinhvien sv = new Sinhvien();
```

Lớp và đối tượng trong Java (tt)

Phương thức khai báo chồng (overloading method): Việc khai báo trong một lớp nhiều phương thức có cùng tên nhưng khác tham số (khác kiểu dữ liệu, khác số lượng tham số) gọi là khai báo chồng phương thức.

Ví dụ: *class Sinhvien*

```
{…
 public void xemThongTinSV() {…}
 public void xemThongTinSV(String psMaSv)
 {…}
}
```

Lớp và đối tượng trong Java (tt)

Tham chiếu *this*: là một biến ẩn tồn tại trong tất cả các lớp, ***this*** được sử dụng trong khi chạy và tham khảo đến bản thân lớp chứa nó.

Ví dụ: *class Sinhvien {*

```
String  maSv, tenSv, dcLienlac;
int tuoi;
…
public void xemThongTinSV() {
        System.out.println(this.maSv);
        System.out.println(this.tenSv);
        …
 }
}
```

Tính đóng gói

- **Đóng gói:** nhóm những gì có liên quan với nhau vào thành một và có thể sử dụng một cái tên để gọi.

  Ví dụ:

  - ✓ Các phương thức đóng gói các câu lệnh.
  - ✓ Đối tượng đóng gói dữ liệu và các hành vi/phương thức liên quan.

  (Đối tượng = Dữ liệu + Hành vi/Phương thức)

Tính đóng gói

Đóng gói dùng để che dấu một phần hoặc tất cả thông tin, chi tiết cài đặt bên trong với bên ngoài.

Ví dụ: khai báo các lớp thuộc cùng gói trong java

package *<tên gói>; // khai báo trước khi khai báo lớp*

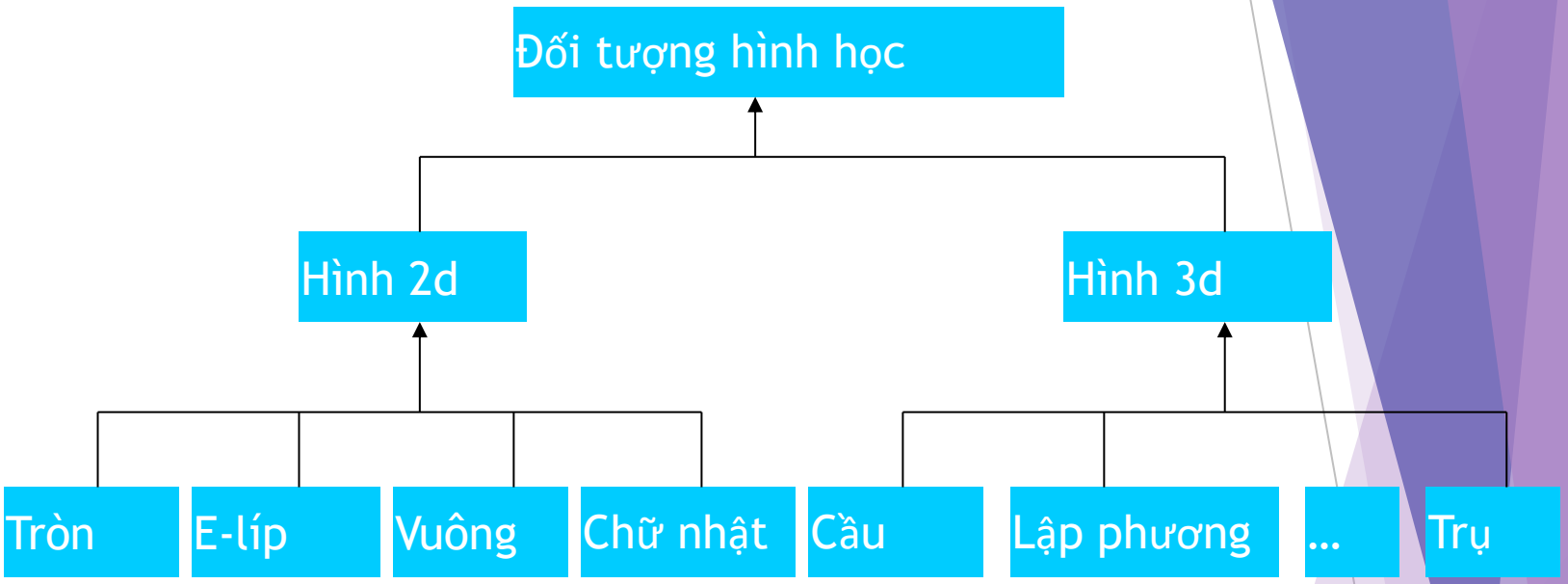
class *<tên lớp>*

{

…

}

Tính kế thừa



- Thừa hưởng các thuộc tính và phương thức đã có
- Bổ sung, chi tiết hóa cho phù hợp với mục đích sử dụng mới
  - ✓ **Thuộc tính**: thêm mới
  - ✓ **Phương thức**: thêm mới hay hiệu chỉnh

Tính kế thừa (tt)

Lớp dẫn xuất hay lớp con (SubClass)

Lớp cơ sở hay lớp cha (SuperClass)

Lớp con có thể kế thừa tất cả hay một phần các thành phần dữ liệu (thuộc tính), phương thức của lớp cha (public, protected, default)

Dùng từ khóa ***extends***.

Ví dụ: *class nguoi { …*

}

class sinhvien ***extends*** *nguoi { …*

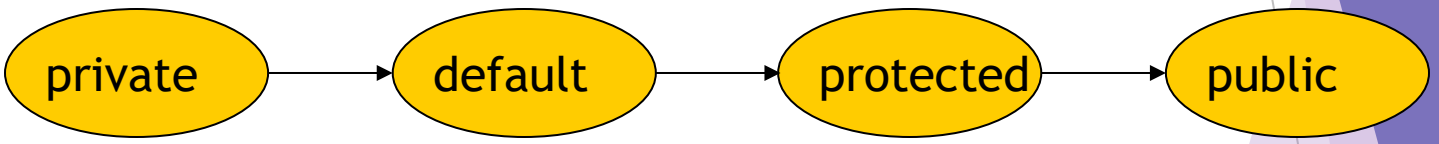
}

Lưu ý: default không phải là 1 từ khóa

Tính kế thừa (tt)

- **Phương thức định nghĩa lại (Overriding Method)**

- Được định nghĩa trong lớp con
- Có tên, kiểu trả về & các đối số giống với phương thức của lớp cha
- Có kiểu, phạm vi truy cập "lớn hơn" phương thức trong lớp cha



Tính kế thừa (tt)

- ***Ví dụ:***

```
abstract class Hinhhoc { …
        public float tinhdientich() {
                return 0;
        }
        …
}

class HinhVuong extends Hinhhoc {
        private int canh;
        public float tinhdientich() {
                return canh*canh;
        }
        …
}
```

*Chỉ có thể **public** do phương thức tinhdientich() của lớp cha là **public***

Tính kế thừa (tt)

```
class HinhChuNhat extends HinhVuong
{
        private int cd;
        private int cr;
        public float tinhdientich() {
                return cd*cr;
        }
        …
}
```

*Chỉ có thể **public** do phương thức tinhdientich() của lớp cha là **public***

Tính kế thừa (tt)

- **Lớp nội:** là lớp được khai báo bên trong 1 lớp khác. Lớp nội thể hiện tính đóng gói cao và có thể truy xuất trực tiếp biến của lớp cha.

Ví dụ:

```
public class A {
    // …
        int <field_1>
        static class B {
            // …
            int <field_2>
            public B(int par_1)
            {
                    field_2 = par_1 + field_1;
            }
        }
}
```

Tính kế thừa (tt)

- **Lớp final:** là lớp không cho phép các lớp khác dẫn xuất từ nó hay lớp final không thể có lớp con.

  Định nghĩa dùng từ khóa **final**

  *public **final** class A*

  {

  …

  }

Tính kế thừa (tt)

- **Lớp trừu tượng:** là lớp dùng để thể hiện sự trừu tượng hóa ở mức cao.
- ***Ví dụ***: lớp "Đối tượng hình học", "Hình 2D", "Hình 3D"

  (Ví dụ định nghĩa lớp các đối tượng hình học cơ bản)
- **Phương thức finalize() của lớp Object - *protected void finalize()***: được "Bộ thu gom rác" gọi tự động khi nhận ra không còn tham chiếu nào đến đối tượng đang xét.

Tính đa hình

Ví dụ:

```
class A_Object {
    // …
    void method_1() { // …
    }
}

class B_Object extends
    A_Object {
    // …
    void method_1() {     // …
    }
}
```

Tính đa hình (tt)

```
class C {
        public static void main(String[] args) {
                A_Object arr_Object[] = new A_Object[2];
                        B_Object var_1 = new
B_Object();
                arr_Object[0] = var_1;
                A_Object var_2;
                for (int i=0; i<2; i++) {
                        var_2 = arr_Object[i];
                        var_2.method_1();
                }
}
```

Phần tử đầu tiên của mảng arr_Object[0] tham chiếu đến 1 đối tượng kiểu Object dẫn xuất từ A_Object

- Với i = 0 thì biến var_2 có kiểu là B_Object, và lệnh var_2.method_1() sẽ gọi thực hiện phương thức method_1 của lớp B_Object.
- Với i = 1 thì biến var_2 có kiểu là A_Object, và lệnh var_2.method_1() sẽ gọi thực hiện phương thức method_1 của lớp A_Object.

Giao tiếp - Interface

- **Interface:** giao tiếp của một lớp, **là phần đặc tả** (không có phần cài đặt cụ thể) của lớp, nó chứa các khai báo phương thức và thuộc tính để bên ngoài có thể truy xuất được. (java, C#, …)
  - ✓ Lớp sẽ cài đặt các phương thức trong interface.
  - ✓ Trong lập trình hiện đại các đối tượng không đưa ra cách truy cập cho một lớp, thay vào đó cung cấp các interface. Người lập trình dựa vào interface để gọi các dịch vụ mà lớp cung cấp.
  - ✓ Thuộc tính của interface là các hằng và các phương thức của giao tiếp là trừu tượng (mặc dù không có từ khóa abstract).

Giao tiếp – Interface (tt)

- ***Ví dụ:***

```
// Định nghĩa một interface Shape trong tập tin shape.java
public interface Shape
{
        // Tính diện tích
        public abstract double area();
        // Tính thể tích
        public abstract double volume();
        // trả về tên của shape
        public abstract String getName();
}
```

Giao tiếp – Interface (tt)

// Lớp *Point* cài đặt/hiện thực *interface* tên *shape*.

```
// Định nghĩa lớp Point trong tập tin Point.java
public class Point extends Object implements Shape {
 protected int x, y; // Tọa độ x, y của 1 điểm
 // constructor không tham số.
 public Point() {
      setPoint( 0, 0 );
 }
    // constructor có tham số.
    public Point(int xCoordinate, int yCoordinate) {
        setPoint( xCoordinate, yCoordinate );
    }
```

Giao tiếp – Interface (tt)

```
// gán tọa độ x, y cho 1 điểm
public void setPoint( int xCoordinate, int yCoordinate ) {
        x = xCoordinate;
        y = yCoordinate;
}
// lấy tọa độ x của 1 điểm
public int getX() {
        return x;
}
// lấy tọa độ y của 1 điểm
public int getY() {
        return y;
}
```

Giao tiếp – Interface (tt)

```
// Thể hiện tọa độ của 1 điểm dưới dạng chuỗi
public String toString() {
 return "[" + x + ", " + y + "]";
}
// Tính diện tích
public double area() {
 return 0.0;
}
// Tính thể tích
public double volume() {
 return 0.0;
}
```

Giao tiếp – Interface (tt)

```
// trả về tên của đối tượng shape
public String getName() {
  return "Point";
}
 } // end class Point
```

Giao tiếp – Interface (tt)

- **Kế thừa giao diện**

```
public interface InterfaceName extends
    interface1, interface2, interface3
{
    // …
}
```

Quản lý Exceptions

Nội dung

- Giới thiệu về Exception
- Kiểm soát Exception
- Ví dụ minh họa
- Thư viện phân cấp các lớp Exception

Giới thiệu về Exception

Ví dụ 1:

```
…
int x = 10;
int y = 0;
float z = x/y;
System.out.print("Ket qua la:" + z);
…
```

Dòng lệnh thứ 3 có lỗi chia cho 0, vì vậy đoạn chương trình kết thúc và dòng lệnh thứ 4 xuất kết quả ra màn hình không thực hiện được.

Giới thiệu về Exception (tt)

Ví dụ 2:

```
…
void docfile(String filename)
{       …
        FileInputStream fin = new FileInputStream(filename);
        …
}
```

Dòng lệnh trên có khả năng xảy ra lỗi đọc file (chẳng hạn khi file không có trên đĩa)

Giới thiệu về Exception (tt)

- **Exception**
  - ✓ Dấu hiệu của lỗi trong khi thực hiện chương trình
  - ✓ *ví dụ*: lỗi chia cho **0**, đọc file không có trên đĩa, …
- **Quản lý Exception (Expcetion handling)**
  - ✓ Kiểm soát được lỗi từ những thành phần chương trình
  - ✓ Quản lý Exception theo 1 cách thống nhất trong những project lớn
  - ✓ Hạn chế, bỏ bớt những đoạn source code kiểm tra lỗi trong chương trình.

Kiểm soát Exception

Ví dụ 1:

```
…
try {
      int x = 10;
      int y = 0;
      float z = x/y;
      System.out.print("Ket qua la:" + z);
}
catch(ArithmeticException e) {
    System.out.println(“Loi tinh toan so hoc”)
}
…
```

Kiểm soát Exception (tt)

Ví dụ 2:

```
…
void docfile(String  filename) throws IOException  {
    …
    FileInputStream fin = new FileInputStream(filename);
    …
}
```

Kiểm soát Exception (tt)

Hoặc

```
…
void docfile(String  filename) {  …
      try {
    …
          FileInputStream fin = new
    FileInputStream(filename);
    …
      }
      catch (IOException e) {
          System.out.println("Loi doc file");
      }
}
```

Kiểm soát Exception (tt)

- Khi có lỗi phương thức sẽ ném ra một exception
- Việc kiểm soát exception giúp chương trình kiểm soát được những trường hợp ngoại lệ và xử lý lỗi.
- Những lỗi không kiểm soát được sẽ có những ảnh hưởng bất lợi trong chương trình.
- Dùng từ khóa **throws** để chỉ định những loại exception mà phương thức có thể ném ra.

▶ <tiền tố> <tên phương thức>(<đối số>) **throws** <các exceptions>

Kiểm soát Exception (tt)

- Đoạn code có thể sinh ra lỗi cần đặt trong khối lệnh bắt đầu bằng **try**.
- Đoạn code để kiểm tra, xử lý trong trường hợp có lỗi xảy ra đặt trong khối lệnh **catch**.

```
try {
    // Đoạn mã có thể sinh ra lỗi …
}
catch (<Kiểu Exception>){
    // Đoạn mã kiểm soát lỗi
}
```

Kiểm soát Exception (tt)

- Khối lệnh đặt trong **finally** luôn được thực thi cho dù có Exception hay không.
- Thường dùng để giải phóng tài nguyên

```
try { // Đoạn mã có thể sinh ra lỗi …
}
Catch (<Kiểu Exception>) { // Đoạn mã kiểm soát lỗi
}
finally {
    // Đoạn mã luôn luôn được thực thi
}
```

Kiểm soát Exception (tt)

```
try {
        // Khối lệnh trước dòng lệnh sinh ra lỗi
        // Dòng lệnh sinh ra lỗi (Exception)
        …
}
catch (<Kiểu Exception>){
        // Đoạn mã kiểm soát lỗi
}
finally { …
}
```

Khối lệnh sau dòng lệnh sinh ra lỗi sẽ bị bỏ qua và không thực hiện khi có exception

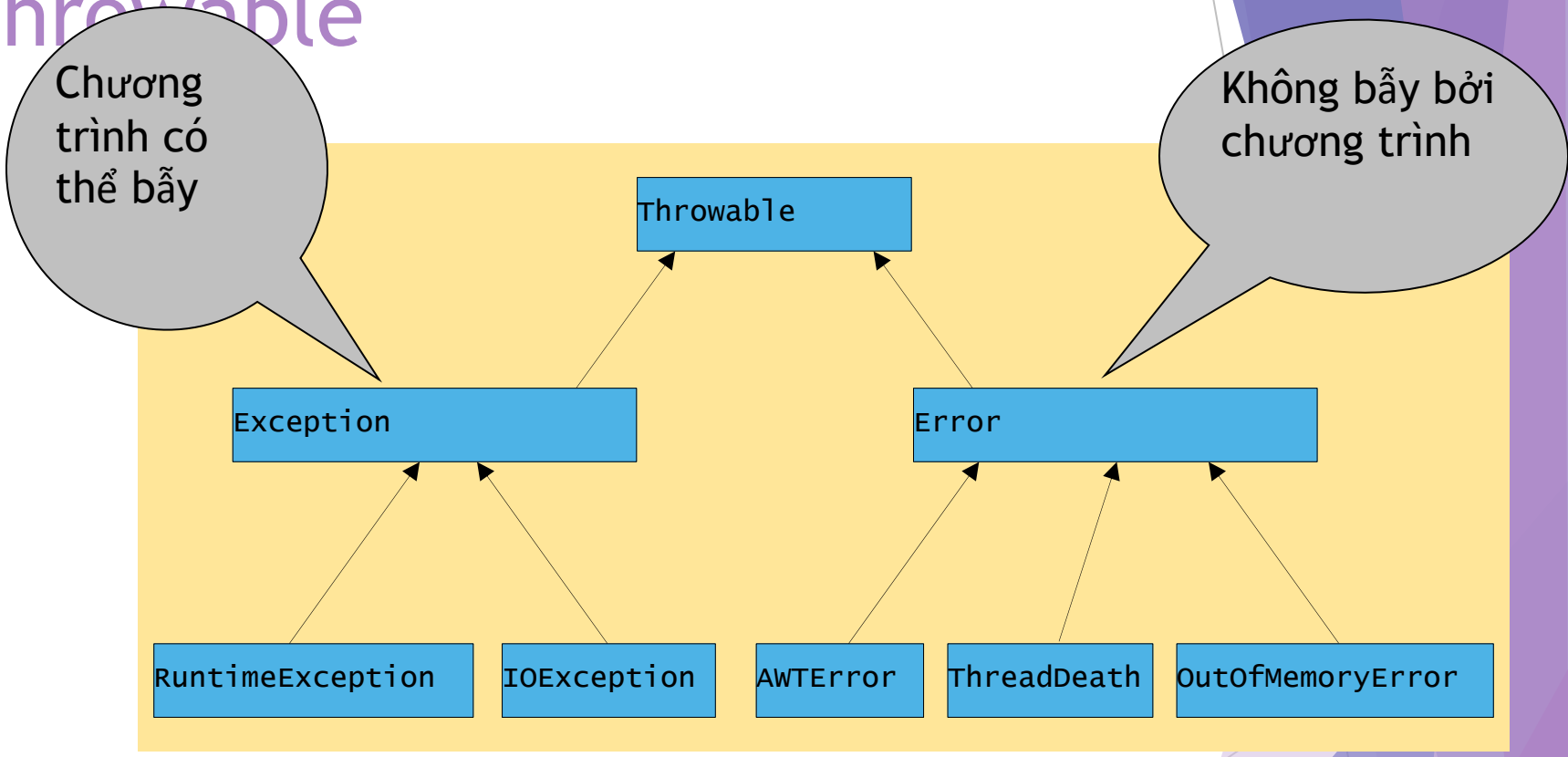
Ví dụ kiểm soát Exception chia cho 0

```
import java.io.*;
public class MainClass {
public static void main(String[] args) {
        try {
        int num_1, num_2;
        BufferedReader in = new BufferedReader(new
        InputStreamReader(System.in));
        System.out.print("\n Nhap so thu 1:");
        num_1 = Integer.parseInt(in.readLine());
        System.out.print("\n Nhap so thu 2:");
        num_2 = Integer.parseInt(in.readLine());
        float rs = num_1/num_2;
        System.out.print("\n Ket qua:" + rs);
      }
```

Ví dụ kiểm soát Exception chia cho 0 (tt)

```
        catch (ArithmeticException e) {
           System.out.print("Loi chia cho 0");
        }
        catch (IOException e) {
           System.out.print("Loi xuat nhap");
        }
        catch(Exception e) {
           System.out.print("Loi khac");
        }
        System.out.print(“Kiem soat duoc loi hay
           khong co loi”);
    }
}
```

Phân cấp thư viện của lớp Throwable



- Có thể định nghĩa các exception mới bằng cách dẫn xuất (**extends**) từ những lớp Exception đang có.

Q/A