

# LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

## Bài 5: Ngoại lệ (Exception)

Trinh Thi Van Anh – PTIT

# Nội dung

- Ngoại lệ
- Bắt và xử lý ngoại lệ
- Ủy nhiệm ngoại lệ
- Tự định nghĩa ngoại lệ
- Ví dụ (person)
- Ví dụ (validate)

# Xét ví dụ

```
public class HelloException {  
    public static void main(String[] args) {  
        System.out.println("Two");  
        // Phép chia này hoàn toàn không có vấn đề.  
        int value = 10 / 2;  
        System.out.println("One");  
        // Phép chia này có vấn đề, chia cho 0.  
        // Lỗi đã xảy ra tại đây.  
        value = 10 / 0;  
        // Và dòng code dưới đây sẽ không được thực hiện.  
        System.out.println("Let's go!");  
    }  
}
```

# Output

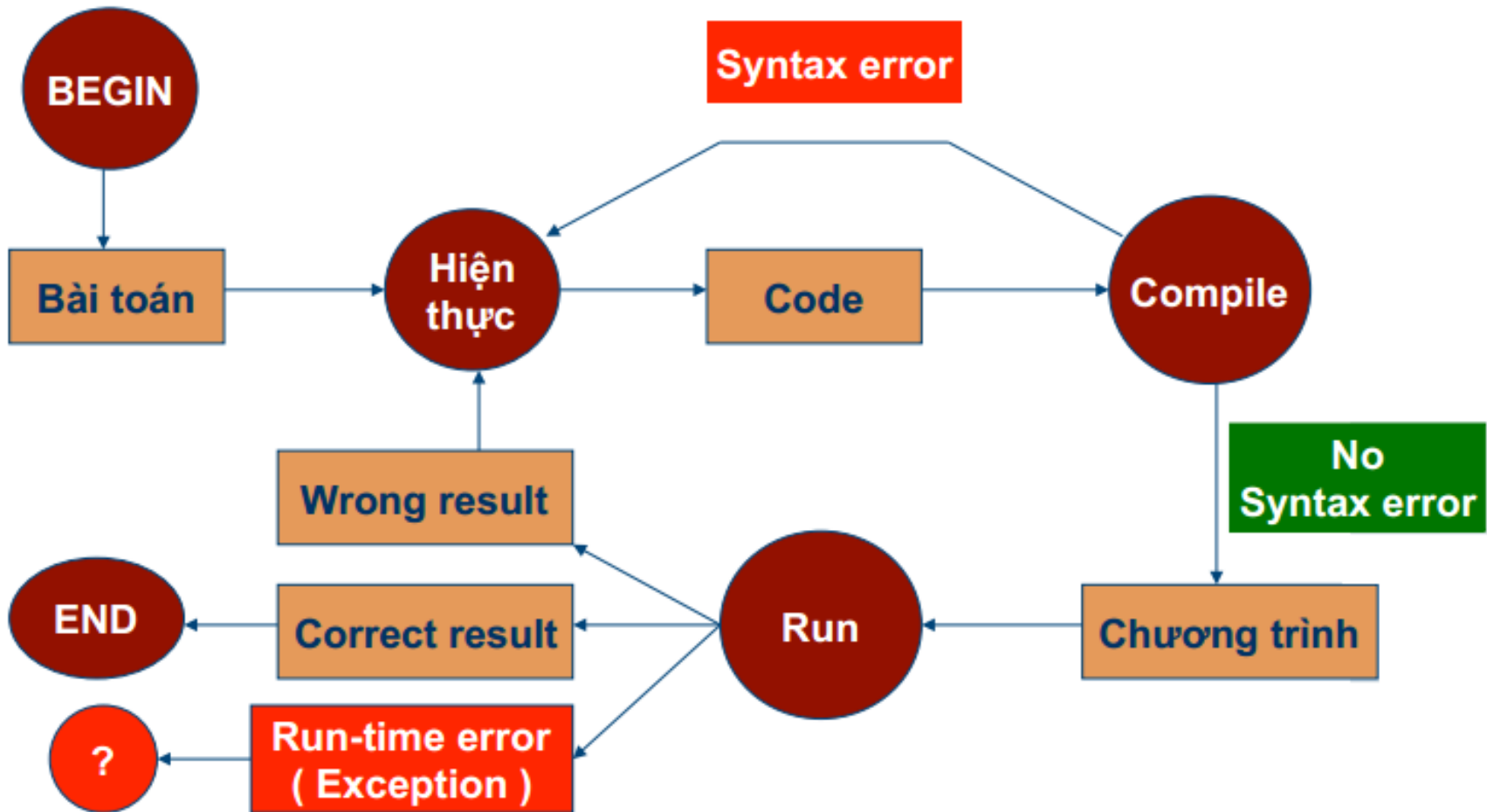
```
Output - Vudu-Bai5 (run) ✖
run:
Two
Exception in thread "main" java.lang.ArithmeticException: / by zero
One
|      at demo.HelloException.main(HelloException.java:20)
|      C:\Users\vip\AppData\Local\NetBeans\Cache\8.1\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

# Sửa lại

```
public class HelloCatchException {  
    public static void main(String[]  
args) {  
        System.out.println("Two");  
        // Phép chia này hoàn toàn  
        //không có vấn đề.  
        int value = 10 / 2;  
        System.out.println("One");  
    }  
}
```

```
try { // Phép chia này có vấn đề, chia cho 0.  
    // Lỗi đã xảy ra tại đây.  
    value = 10 / 0;  
    // Dòng code này sẽ không được chạy.  
    System.out.println("Value =" + value);  
} catch (ArithmeticException e) {  
    // Các dòng lệnh trong catch được thực thi  
    System.out.println("Error: " +  
e.getMessage()) ;  
    // Các dòng lệnh trong catch được thực thi  
    System.out.println("Ignore...") ;  
    }  
    // Dòng lệnh này được thực hiện.  
    System.out.println("Let's go!");  
}}
```

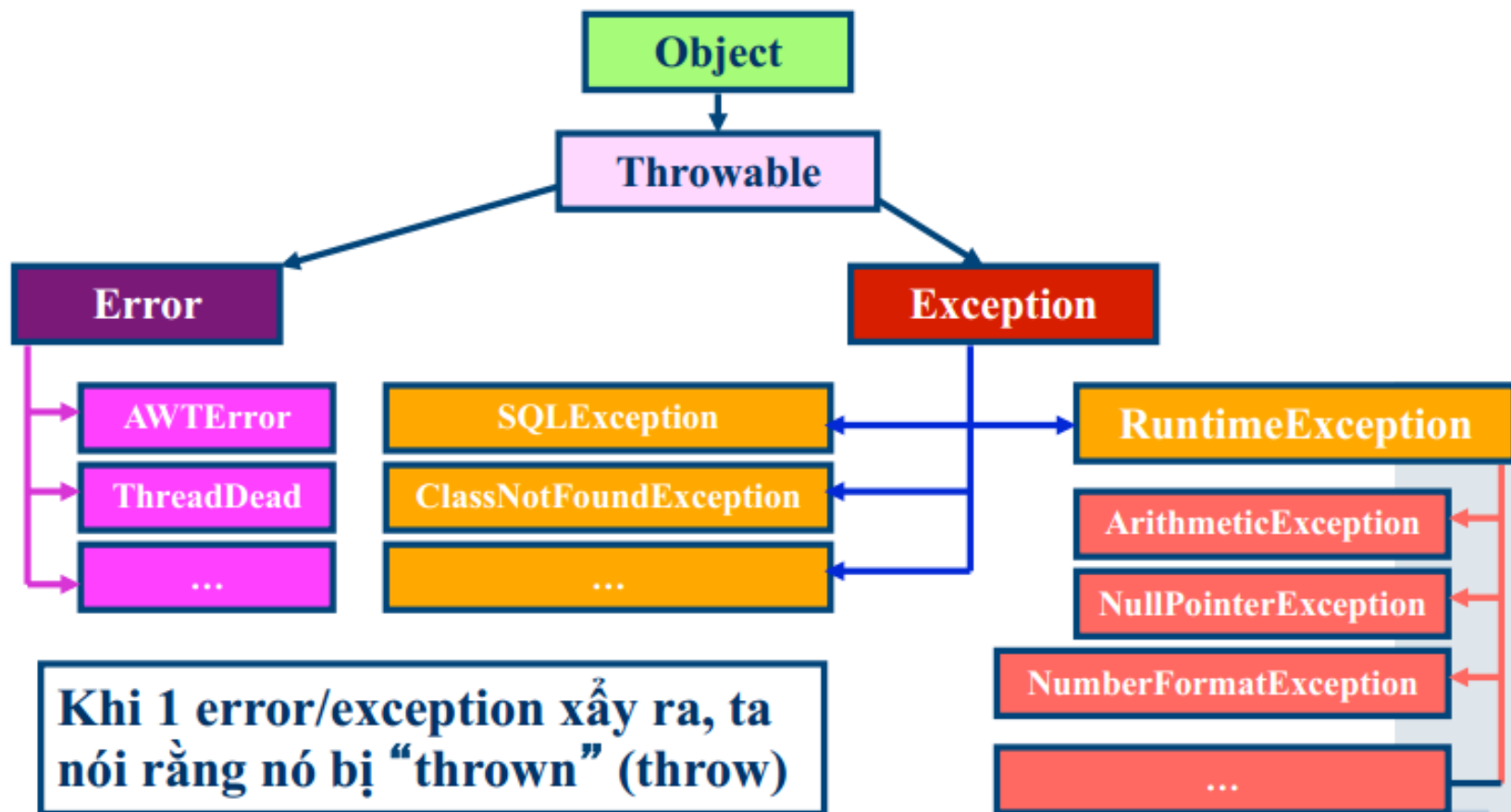
# Các loại lỗi của chương trình



- Compile-time error = Syntax error
- ! Run-time error = Exception, tình huống bất bình thường đã xảy ra trong khi chương trình thực thi.
- Khi có Exception:
  - Có thể là máy bị treo (halt).
  - Chương trình ngắt đột ngột, điều khiển trả về cho OS, OS thu hồi bộ nhớ của chương trình (ném ra ngoài).
- Cần có cơ chế điều khiển tình huống này.



# Sơ đồ phân cấp của Exception



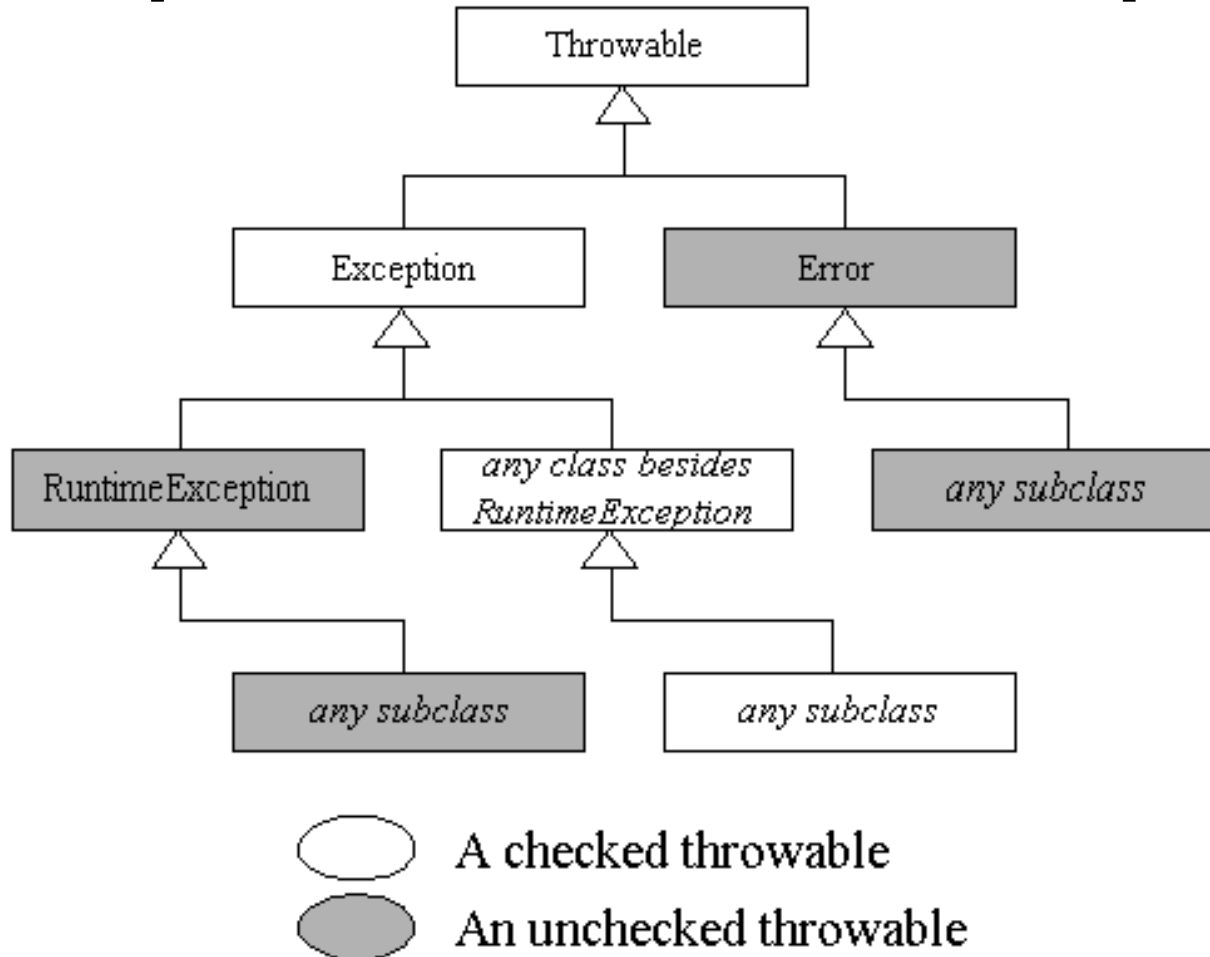
# Sơ đồ phân cấp

- Đây là mô hình sơ đồ phân cấp của Exception trong java.
  - Class ở mức cao nhất là **Throwable**
  - Hai class con trực tiếp là **Error** và **Exception**
- Trong nhánh Exception có một nhánh con **RuntimeException** là các ngoại lệ sẽ không được java kiểm tra trong thời điểm biên dịch.
- **Error**: Khi liên kết động thất bại, hoặc trong máy ảo xảy ra một vấn đề nghiêm trọng, nó sẽ ném ra một Error. Các chương trình Java điển hình không nên bắt lỗi (Error). Ngoài ra, nó không chắc rằng các chương trình Java điển hình sẽ bao giờ ném lỗi

- **Exception:** Hầu hết các chương trình ném và bắt các đối tượng xuất phát từ lớp ngoại lệ. Trường hợp ngoại lệ cho thấy một vấn đề xảy ra nhưng vấn đề không phải là một vấn đề mang tính hệ thống nghiêm trọng. Hầu hết các chương trình bạn viết sẽ ném và bắt ngoại lệ.
- Một class ngoại lệ con có ý nghĩa đặc biệt trong ngôn ngữ Java: **RuntimeException**.
- Class **RuntimeException** đại diện cho trường hợp ngoại lệ xảy ra trong thời gian chạy chương trình. Một ví dụ về một ngoại lệ thời gian chạy là **NullPointerException**, xảy ra khi một bạn truy cập vào method hoặc field một đối tượng thông qua một tham chiếu null. Với các ngoại lệ kiểu này người ta thường kiểm tra để đảm bảo nó sẽ không xảy ra hơn là tìm bắt nó.

# Các loại Exception trong Java

- Java có 2 loại Exception là **Checked exceptions** và **Unchecked exceptions**.



- **checked exceptions:** Là loại exception xảy ra trong lúc compile time, nó cũng có thể được gọi là compile time exceptions. Loại exception này không thể bỏ qua được trong quá trình compile, bắt buộc ta phải handle nó.
- Ví dụ: IOException, FileNotFoundException....
- **Unchecked exceptions:** Là loại exception xảy ra tại thời điểm thực thi chương trình, nó cũng có thể gọi là runtime exceptions đó là programming bugs, lỗi logic của chương trình... Loại exception này được bỏ qua trong quá trình compile, không bắt buộc ta phải handle nó.
- Ví dụ: NumberFormatException, ArrayIndexOutOfBoundsException...

- Có một số điểm cần lưu ý
  - RuntimeException và các lớp con của nó là một tập hợp con của *Unchecked Exceptions*. Và vì vậy ***RuntimeException và Unchecked Exceptions không phải là hai từ đồng nghĩa***
  - Lớp **Error** và các lớp con của nó cũng không được kiểm tra tại thời điểm biên dịch nên cũng được coi là Unchecked Exceptions
- Một số phương thức quan trọng trong lớp Throwable
  - **getMessage()** : Trả về kiểu chuỗi lấy ra thông tin của lỗi, dùng trong khối catch(). Nếu getMessage() trả về null tức là ngoại lệ đó không có thông tin được lưu dưới dạng chuỗi trong đó.
  - **toString()**: Trả về chuỗi ghi tên của loại exception cùng với kết quả của getMessage()
  - **printStackTrace()** : In ra kết quả của toString() hiển thị thông điệp lỗi và chi tiết lỗi trên dòng nào, dùng trong khối catch, trả về kiểu void.

# Cách bắt lỗi trong Java

- Có 2 cách xử lý lỗi trong Java là xử lý lỗi trực tiếp bằng **try-catch** và xử lý lỗi gián tiếp bằng **throws**
- Cú pháp: Single catch block

```
try {  
    <các lệnh có khả năng gây lỗi>  
}  
catch ( ExceptionName e) {  
    <catch block>  
}
```

## ■ Multiple catch blocks

```
try {  
    // Làm gì đó tại đây  
} catch (Exception1 e) {  
    // Làm gì đó tại đây  
} catch (Exception2 e) {  
    // Làm gì đó tại đây  
} finally {  
    // Khởi finally luôn luôn được thực thi  
    // Làm gì đó tại đây.  
}
```



# Ví dụ

```
public class Exception1 {  
    public static void main(String[] args) {  
        String sNum = "CTB";  
        String sDate = "10/03/2016";  
        int num = Integer.parseInt(sNum);  
        SimpleDateFormat f = new  
SimpleDateFormat("dd/MM/yyyy");  
        Date d = f.parse(sDate);  
    }  
}
```

- Khi chuyển chuỗi sang số nguyên trình biên dịch không hề báo lỗi mặc dù chuỗi "CTB" không thể chuyển sang số nguyên – Unchecked Exceptions
- Khi chuyển chuỗi sang thời gian trình biên dịch thông báo lỗi mặc dù chuỗi "10/3/2016" hoàn toàn phù hợp với định dạng khiến chương trình không dịch được – Checked Exceptions

# Ví dụ try...catch

```
public class Excepton2 {  
    public static void main(String[] args) {  
        String s = "Mon LTHDT";  
        try{  
            System.out.println("Truoc");  
            System.out.println(s.substring(50));  
        } catch (StringIndexOutOfBoundsException  
e) {  
  
            System.out.println("Loi:"+e.toString());  
        }  
        System.out.println("Sau");  
    }  
}
```

# Unchecked exception

```
public class Exception3 {  
    public static void main(String[] args) {  
        String s = "Mon LTHDT";  
        int x=12, y=0;  
        try{  
            System.out.println("Truoc");  
            System.out.println(s.substring(50));  
            System.out.println("x/y = "+ x/y);  
        } catch (StringIndexOutOfBoundsException  
e) {  
            System.out.println("Loi:"+e.toString());  
        } catch (ArithmeticException e) {  
            System.out.println("Loi: "+  
e.getMessage()); }  
    }  
}
```

```
catch (Exception e) {  
    e.printStackTrace();  
} finally {  
    System.out.println("Luon thuc  
hien");  
}  
System.out.println("Sau");  
}}
```

# Checked Exceptions

```
public class Exception4 {  
    public static void main(String[]  
args) {  
        File file = new File("vidu.txt");  
        try{  
            file.createNewFile();  
        } catch (IOException ex) {  
            System.out.println("Lỗi!!!!");  
        }  
    }  
}
```

# Chú ý

- “Try” có thể không có “Catch” nhưng khi đứng một mình nhất định phải có “Finally”
- “Catch” nhất định phải có “Try” Đối với một “Try” có thể có nhiều “Catch” nhưng không được “Catch ngược”
  - Nguyên tắc khai báo **catch** là cái **ở trên** phải là **con** của **cái dưới**
- **Vì:** catch thứ 3 là cha của tất cả các exception, ta khai báo về đầu thì tất cả những thằng sau chả bao giờ được sử dụng nữa nên nó sẽ báo lỗi (vì có ngoại lệ thì nó sẽ nhảy luôn vào cái đầu tiên)

```
try{
    System.out.println("Truoc");
    System.out.println(s.substring(50));
    System.out.println("x/y = "+ x/y);
} catch (Exception e) {
    e.printStackTrace();
} catch (StringIndexOutOfBoundsException e) {
    System.out.println("Loi:"+e.toString());
} catch (ArithmeticException e) {
    System.out.println("Loi: "+
        e.getMessage());
} finally{
    System.out.println("Luon thuc hien");
}
System.out.println("Sau");
```

# try-catch-finally

```
public class Exception5 {  
    public static void main(String[] args) {  
        String sNum="098A1";  
        int num = toInteger(sNum);  
        System.out.println("Gia tri:"+num);  
    }  
    public static int toInteger(String st){  
        System.out.println("Bat dau:");  
        try{  
            int num = Integer.parseInt(st);  
            return num;  
        } catch (NumberFormatException e) {  
            System.out.println(e.toString());  
            return 0;  
        } finally{  
            System.out.println("Ket thuc!!!");  
        }  
    }  
}
```

- Khởi **finally** luôn được thực thi



# Ủy nhiệm ngoại lệ

- Phương thức có thể ủy nhiệm ngoại lệ cho vị trí gọi nó bằng cách: Sử dụng throws ở phần định nghĩa phương thức để báo hiệu
- **Throw:** để quăng ra Exception ở bất kỳ dòng nào trong phương thức (sau đó dùng try-catch để bắt hoặc throws cho cái khác xử lý)
- **Throws:** Chỉ có phương thức mới được sử dụng. Khi một phương thức có throw bên trong mà không bắt lại (try – catch) thì phải ném đi (throws) cho cái khác xử lý.
- **Chú ý:** Nếu phương thức đang ném ra lỗi (throws), mà phương thức khác gọi phương thức đang ném ra lỗi thì phương thức đó vẫn phải ném ra lỗi hoặc xử lý luôn (try – catch)
- Khối “Try-Catch” dùng để xử lý luôn lỗi, còn “Throws” dùng để ném lỗi đi cho cái khác xử lý, và dùng “Throws” khi ta không muốn đặt nhiều khối “Try-Catch” bên trong.

```
■ public static int getEndMember(int[] a) throws
  ArrayIndexOutOfBoundsException {
    return a[a.length-1];
  }
  public static void main(String[] args) {
    int[] a = {1, 2, 0};
    int j;
    try {
      int i = getEndMember(a);
      if (i == 0) {
        throw new ArithmeticException();
      }
      else{
        j = 10/i;
      }
    }
    catch (ArithmeticException e) {
      System.out.println("ArithmeticException");
    }
    catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("ArrayIndexOutOfBoundsExceptionkk
      kk");
    }
  }
```

```
final class Fraction extends Number {
    private int numerator;
    private int denominator;
    public Fraction(int numerator, int
denominator) {
        if(denominator == 0) {
            throw new
IllegalArgumentException("denominator is
zero");
        }
        if(denominator < 0) {
            numerator *= -1;
            denominator *= -1;
        }
        this.numerator = numerator;
        this.denominator = denominator;
    }
}
```

```
public class ExceptionExample { public
    static void main(String[] args) {
        try{ writeToFile();
        }catch (IOException e) {
            System.out.println("Da co loi
xay ra khi thuc hien ghi file" + e);
        }
    }
    private static void writeToFile()
throws IOException {
        FileWriter fileWriter = new
FileWriter("data.txt");
        fileWriter.write("Xu ly ngoai le
trong Java");
        fileWriter.close(); } }
```

- Một phương thức có thể ủy nhiệm nhiều hơn 1 ngoại lệ

```
public void myMethod(int tuoi, String  
    ten)
```

```
throws ArithmeticException,  
    NullPointerException {
```

```
    if (tuoi < 18) {  
        throw new ArithmeticException("Chua  
    du tuoi!");  
    }
```

```
    if (ten == null) {  
        throw new  
    NullPointerException("Thieu ten!");  
    }
```

```
// ...  
}
```

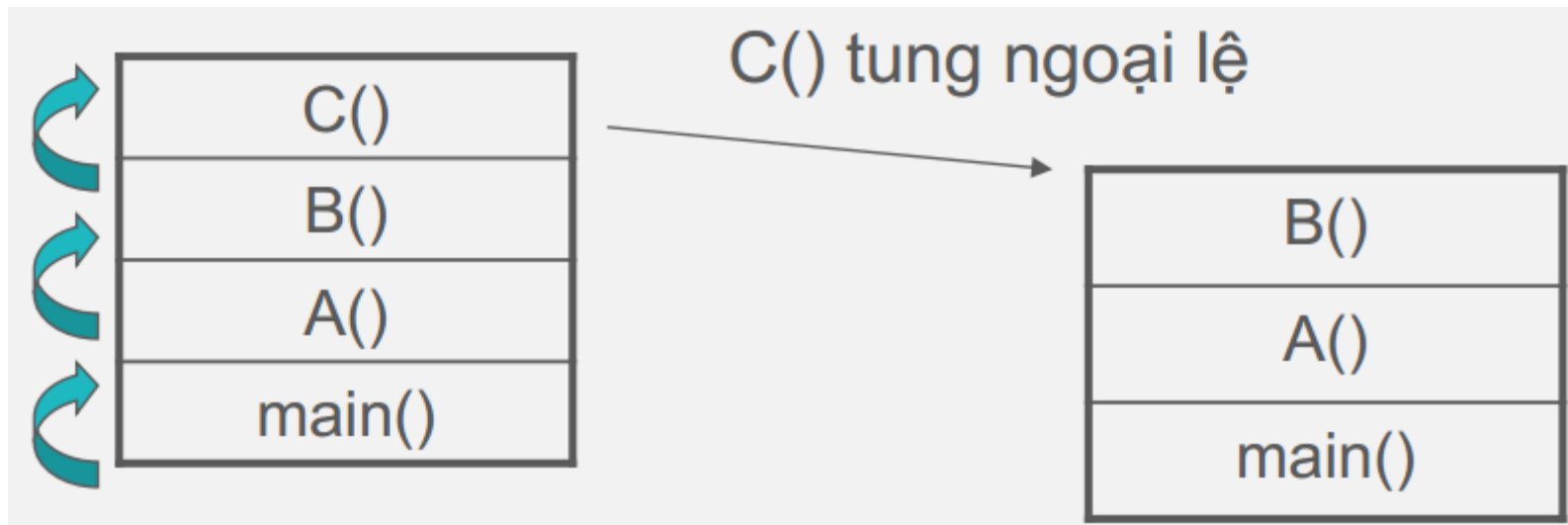
# Một số lớp quản lý lỗi của Java

| class                          | Giải thích                                      |
|--------------------------------|---|
| Exception                      | Lớp nền của các run-time error                  |
| RuntimeException               | Lớp nền của nhiều lớp run-time error            |
| ArithmeticException            | Lỗi do thực thi một phép toán                   |
| IllegalArgumentException       | Lỗi sai đối số của hàm                          |
| ArrayIndexOutOfBoundsException | Lỗi do chỉ số ngoài tầm của mảng                |
| NullPointerException           | Lỗi do truy xuất một đối tượng mà chưa khởi tạo |
| SecurityException              | Lỗi do truy cập bị cấm                          |
| ClassNotFoundException         | Lỗi do không tìm thấy file.class                |

| Lớp                    | Giải thích  |
|------------------------|---|
| NumberFormatException  | Lỗi do không đúng dạng số                             |
| IOException            | Lỗi xuất nhập   |
| FileNotFoundException  | Lỗi do không tìm thấy file                            |
| EOFException           | Lỗi do cố truy cập nội dung 1 file khi đã ở cuối file |
| IllegalAccessException | Lỗi do truy cập 1 class bị cấm                        |
| NoSuchMethodException  | Lỗi do viết sai tên hành vi                           |
| InterruptedException   | Lỗi do ngắt ngang 1 luồng lệnh đang được thực thi     |

# Lan truyền ngoại lệ

- Tình huống:
- Giả sử trong `main()` gọi phương thức `A()`, trong `A()` gọi `B()`, trong `B()` gọi `C()`. Khi đó một ngăn xếp các phương thức được tạo ra.
- Giả sử trong `C()` xảy ra ngoại lệ.





- Nếu C() gặp lỗi và tung ra ngoại lệ nhưng trong C() lại không xử lý ngoại lệ này, thì chỉ còn một nơi có thể xử lý chính là nơi mà C() được gọi, đó là trong phương thức B().
- Nếu trong B() cũng không xử lý thì phải xử lý ngoại lệ này trong A()... Quá trình này gọi là lan truyền ngoại lệ
- Nếu đến main() cũng không xử lý ngoại lệ được tung từ C() thì chương trình sẽ phải dừng lại.

# Kế thừa và ủy nhiệm ngoại lệ

- Khi override một phương thức của lớp cha, phương thức ở lớp con không được phép tung ra các ngoại lệ mới
- Phương thức ghi đè trong lớp con chỉ được phép tung ra các ngoại lệ giống hoặc là lớp con hoặc là tập con của các ngoại lệ được tung ra ở lớp cha.

```
class Disk {  
    void readFile() throws EOFException{  
    }  
class FloppyDisk extends Disk {  
    // ERROR!  
    void readFile() throws IOException {  
    }  
}
```

# Sửa lại

```
class Disk {  
    void readFile() throws IOException  
        {}  
}  
  
class FloppyDisk extends Disk {  
    void readFile() throws EOFException  
        {} //OK  
}
```

# Exception tự định nghĩa

- Chúng ta có thể tự định nghĩa ra những lớp ngoại lệ riêng nếu chúng ta thấy java cung cấp chưa đủ thông tin ngoại lệ như chúng ta mong muốn.
  - Kế thừa từ một lớp Exception hoặc lớp con của nó
  - Có tất cả các phương thức của lớp Throwable

- Ví dụ

```
class MyException extends Exception {  
    MyException() {.... }  
    MyException(String s) {  
        super(s);  
        .... }  
}  
public class ExceptionDemo {  
    public static void executeHasException() throws  
    MyException {  
        throw new MyException();  
    }  
}
```

# Ví dụ (nganhang)

- Ví dụ: Khi rút tiền ngân hàng, người dùng rút tiền mà vượt quá số số dư có trong ngân hàng, sẽ không có Exception hệ thống nào giúp mình làm điều đó, vì thế chúng ta phải định nghĩa Exception phù hợp với nhu cầu thực tế
- Tạo class **TaiKhoanNganHang** để thực hiện các tác vụ gửi tiền, rút tiền và một class **LoiChuyenTien** để xử lý nếu có ngoại lệ xảy ra khi ta rút số tiền vượt quá trong tài khoản.

```
public class TaiKhoanNganHang {  
    double sodu;  
    public void guiTien(double tienGui) {  
        sodu += tienGui;    }  
    public double getSodu() {  
        return sodu;    }  
    public void setSodu(double sodu) {  
        this.sodu = sodu;    }  
    public void rutTien(double tienRut)  
throws LoiChuyenTien{  
        if(sodu>tienRut)  
            sodu = sodu - tienRut;  
        else  
            throw new LoiChuyenTien(tienRut) ;  
    }  
}
```

```
class LoiChuyenTien extends  
    Exception{  
    double tienRut;  
    public LoiChuyenTien(double  
tienRut) {  
        this.tienRut = tienRut;    }  
    public String toString() {  
        return ("So tien "+tienRut + "  
can rut lon hon so tien co!!");  
    }  
}
```

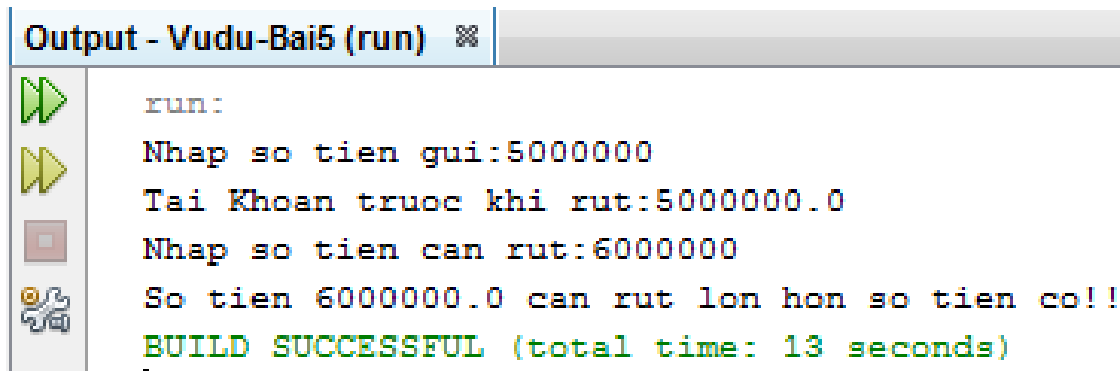
```
public class RutTienNH {  
    public static void main(String[] args) {  
        double tienGui,tienRut;  
        Scanner in = new Scanner(System.in);  
        System.out.print("Nhap so tien gui:");  
        tienGui = in.nextDouble();  
        TaiKhoanNganHang tk = new  
TaiKhoanNganHang();  
        tk.guiTien(tienGui);  
        System.out.println("Tai Khoan truoc  
khi rut:"+tk.getSodu());  
        System.out.print("Nhap so tien can  
rut:");  
        tienRut = in.nextDouble();
```



```
try{
    tk.rutTien(tienRut);
    System.out.println("So du su rut:
    "+tk.getSodu());
}catch (LoiChuyenTien ex) {
    System.out.println(ex.getMessage());
}

in.close();    }}
```

■ Output:



The screenshot shows an IDE output window titled "Output - Vudu-Bai5 (run)". The output text is as follows:

```
run:
Nhap so tien gui:5000000
Tai Khoan truoc khi rut:5000000.0
Nhap so tien can rut:6000000
So tien 6000000.0 can rut lon hon so tien co!!
BUILD SUCCESSFUL (total time: 13 seconds)
```

# Ví dụ (person)

- AgeException.java

```
public class AgeException extends Exception {  
    public AgeException(String message) {  
        super(message);  
    }  
}
```

- TooYoungException.java

```
public class TooYoungException extends  
    AgeException {  
    public TooYoungException(String message) {  
        super(message);  
    }  
}
```

- **TooOldException.java**

```
public class TooOldException extends AgeException {  
    public TooOldException(String message) {  
        super(message);  
    }  
}
```

- **AgeUtils.java**

```
public class AgeUtils {  
    public static void checkAge(int age) throws  
        TooYoungException,  
        TooOldException {  
        if (age < 18) {  
            throw new TooYoungException("Age " + age  
+ " too young");  
        } else if (age > 40) {  
            throw new TooOldException("Age " + age +  
" too old");  
        }  
        System.out.println("Age " + age + " OK!");  
    }  
}
```

## ■ TryCatchDemo1.java

```
public class TryCatchDemo1 {  
    public static void main(String[] args) {  
        System.out.println("Start Recruiting ...");  
        System.out.println("Check your Age");  
        int age = 50;  
        try {  
            AgeUtils.checkAge(age);  
            System.out.println("You pass!");  
        } catch (TooYoungException e) {  
            // Làm gì đó tại đây ..  
            System.out.println("You are too young,  
not pass!");  
            System.out.println(e.getMessage());  
        } catch (TooOldException e) {  
            // Làm gì đó tại đây  
            System.out.println("You are too old, not  
pass!");  
            System.out.println(e.getMessage());  
        }  
    }  
}
```

## ■ TryCatchDemo2.java

```
public class TryCatchDemo2 {  
    public static void main(String[] args) {  
        System.out.println("Start Recruiting  
        ...");  
        System.out.println("Check your Age");  
        int age = 15;  
        try {  
            AgeUtils.checkAge(age);  
            System.out.println("You pass!");  
        } catch (AgeException e) {  
            System.out.println("Your age  
invalid, you not pass");  
  
            System.out.println(e.getMessage());  
        }  
    }  
}
```

## ■ TryCatchDemo3.java

```
public class TryCatchDemo3 {  
    public static void main(String[] args) {  
        System.out.println("Start Recruiting ...");  
        System.out.println("Check your Age");  
        int age = 15;  
        try {  
            AgeUtils.checkAge(age);  
            System.out.println("You pass!");  
        } catch (TooYoungException | TooOldException e) {  
            // Gộp 2 ngoại lệ trong cùng một khối  
            catch  
                System.out.println("Your age invalid,  
you not pass");  
            System.out.println(e.getMessage());  
        }  
    }  
}
```

## ■ Person.java

```
public class Person {  
    public static final String MALE = "male";  
    public static final String FEMALE =  
        "female";  
    private String name;  
    private String gender;  
    private int age;  
    public Person(String name, String gender,  
        int age) {  
        this.name = name;  
        this.gender = gender;  
        this.age = age;  
    }  
    // getter and setter  
}
```

## ■ GenderException

```
public class GenderException extends  
    Exception {  
    public GenderException(String  
        message) {  
        super(message) ;  
    }  
}
```

## ■ ValidateException

```
public class ValidateException extends  
    Exception {  
    public ValidateException(Exception e) {  
        super(e) ;  
    }  
  
}
```



## ■ ValidateUtils.java

```
public class ValidateUtils{  
    public static void checkPerson(Person  
person) throws ValidateException {  
        try {  
            AgeUtils.checkAge(person.getAge());  
        } catch (Exception e) {  
            throw new ValidateException(e);  
        }  
        if  
(person.getGender().equals(Person.FEMALE))  
{  
            GenderException e = new  
GenderException("Do not accept women");  
            throw new ValidateException(e);  
        }  
    }  
}
```

## ■ WrapperExceptionDemo

```
public class WrapperExceptionDemo {  
    public static void main(String[] args) {  
        Person person = new Person("Marry",  
        Person.FEMALE, 20);  
        try {  
            ValidateUtils.checkPerson(person);  
        } catch (ValidateException wrap) {  
            Exception cause = (Exception)  
wrap.getCause();  
            if (cause != null) {  
                System.out.println("Not pass,  
cause: " + cause.getMessage());  
            } else {  
  
                System.out.println(wrap.getMessage());  
            }  
        }  
    }  
}
```

# Ví dụ (validate)

```
public class Student {  
    private String id;  
    private String ho;  
    private String ten;  
    private String ngaysinh;  
    public Student() {  
    }  
    public Student(String id, String ho,  
                    String ten, String ngaysinh) {  
        this.id = id;  
        this.ho = ho;  
        this.ten = ten;  
        this.ngaysinh = ngaysinh;  
    }  
}
```

```
//getter and setter  
public String toString() {  
    return id+"\t"+ho+"  
    "+ten+"\t"+ngaysinh;  
}
```

```
public class ValidateException
    extends Exception {
        public ValidateException(String
message) {
            super (message) ;
        }
    }
```

```
public class Dsach {  
    private ArrayList<Student> list;  
    public Dsach() {  
        list=new ArrayList<>();  
    }  
    private boolean isStudent(String  
id) {  
        for(Student s:list) {  
if(s.getId().equalsIgnoreCase(id)) {  
            return true;  
        }  
    }  
    return false;  
}
```

```
private void validateMa(String id)  
    throws ValidateException{  
    if(!id.matches("^[Bb]{1}\\d{2}[A-Za-  
z]{4}\\d{3}$"))  
        throw new ValidateException("Ma  
\""+id+"\"khong dung dinh dang");  
    }  
  
private void validateNgaySinh(String ngaysinh)  
    throws ValidateException{  
    if(!ngaysinh.matches("\\d{2}-\\d{2}-  
\\d{4}"))  
        throw new ValidateException("Ngay sinh  
\""+ngaysinh+"\"khong dung dinh dang");  
    }
```