Bài 5
I. Ngoại lệ và xử lý ngoại lệ

NỘI DUNG:

- Ngoại lệ là gì
- Các kiiểu và lớp ngoại lệ
- Xử lý ngoại lệ

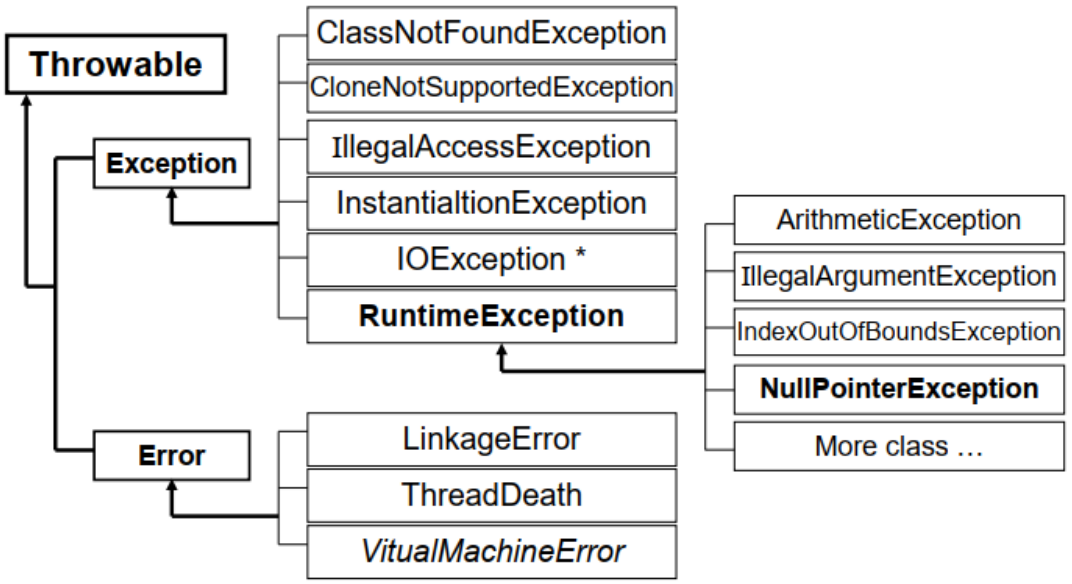
1. Ngoại lệ là gì?

- Có những lỗi chỉ khi chạy chương mới xuất hiện và chương trình đang chạy lập tức ngừng lại và xuất hiện thông báo lỗi – đó chính là ngoại lệ (exception).
- Ví dụ: Chương trình chia 2 số. Nếu ta cho mẫu số =0 thì phát sinh lỗi và đó được coi là 1 ngoại lệ.

5/0 ⇨ error !

1. Ngoại lệ là gì ? (tiếp)

- Mọi lỗi xẩy ra khi chương trình đang chạy trong Java đều được mô tả bởi các lớp tượng ứng (hướng đối tượng)
- Tất cả các lớp ngoại lệ và lỗi đều kế thừa trực tiếp hoặc gián tiếp từ lớp Throwable
- Ngoại trừ lớp ngoại lệ IOException thuộc gói java.io, còn mọi lớp mô tả lỗi và ngoại lệ đều thuộc gói java.lang

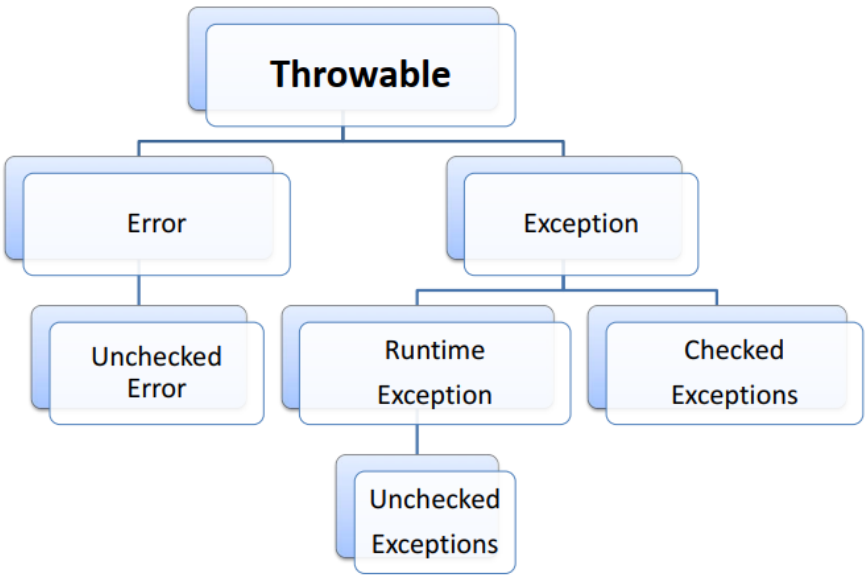


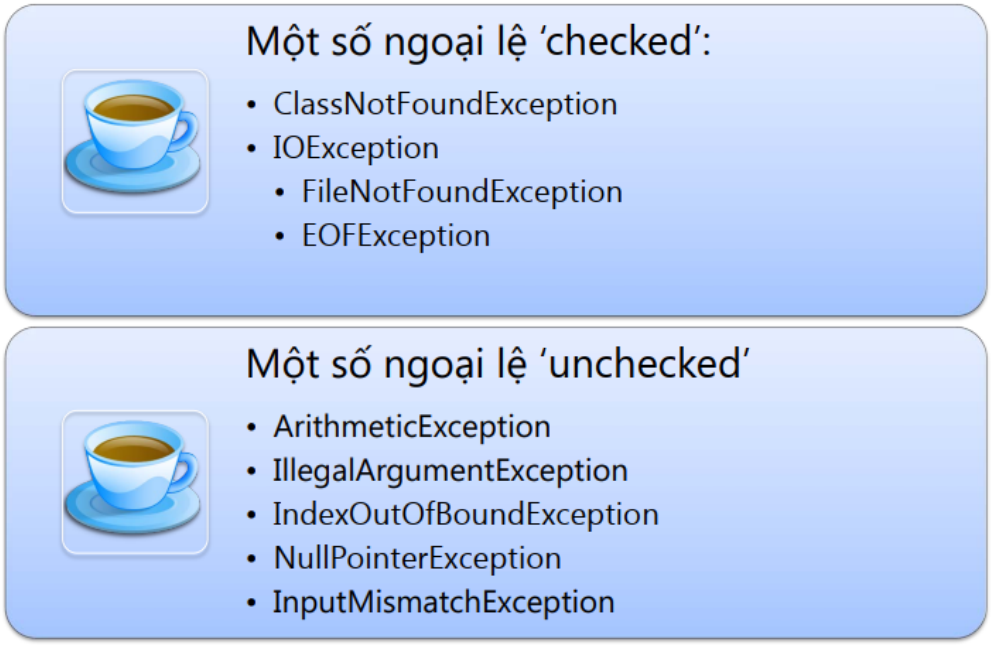
2. Các kiểu ngoại lệ

- 3 kiểu:
  - Ngoại lệ Checked
  - Ngoại lệ Unchecked
  - Lỗi (Error)

2. Các kiểu ngoại lệ

- Ngoại lệ Checked:
  - Là các lớp ngoại lệ kế thừa trực tiếp từ lớp Throwable trừ các lớp IOException, SQLException,…
  - Các ngoại lệ này được kiểm tra tại thời điểm biên dịch.
- Ngoại lệ Unchecked:
  - Là các lớp ngoại lệ kế thừa từ lớp RuntimeException như ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException…
  - Các ngoại lệ Uncheked không được kiểm tra tại thời điểm biên dịch mà đuwọc kiểm tra trong khi chạy chương trình.
- Error:
  - Là các lỗi không xẩy ra trong chương trình mà chủ yếu do môi trường chạy chương trình.
  - Ví dụ như OutOfMemoryError, VirtualMachineError, AssertionError…..





3. Xử lý ngoại lệ:

- Để xử lý ngoại lệ sử dụng 5 từ khóa:

- try: Để giám sát một khối lệnh, khi có lỗi nó ném trả ngoại lệ về chương trình. Cú pháp: try{……}

- catch: Là bẫy để bắt ngoại lệ xử lý mà chủ yếu là hiển thị trạng thái ngoại lệ, từ khóa catch thường đi với try để trở thành cấu trúc xử lý ngoại lệ. Cú pháp:

catch(<Kiểu ngoại lệ> <biến>) { <các lệnh xử lý>}

- finally: Cấu trúc finally bắt buộc một chương trình phải thực hiện khối lệnh của nó trước khi kết thúc cho dù có xảy ra lỗi hay không. Từ khóa finally thường đi với từ khóa try hoặc try…catch để trở thành cấu trúc xử lý ngoại lệ.

- throw: để kích hoạt ném về một ngoại lệ (đối tượng)

- throws: cho phép một phương thức ném trả về ngoại lệ thay vì xử lý trong phương thức. Chương trình nào sử dụng phương thức này phải xử lý ngoại lệ nó ném trả về.

Các cấu trúc xử lý ngoại lệ:

Sử dụng từ khóa try và catch

```
try{
    //Khối lệnh
}catch(…){
    //Khối lệnh xử lý ngoại lệ
}
```

Ví dụ: Nếu không dùng try... catch, xét ví dụ sau:

```
c=a/b;
System.out.println("Sau phep chia !");(*)
```

Câu lệnh (*) sẽ không được thực hiện nếu mẫu số b=0, chương trình lập tức ngừng lại và xuất hiện thông báo lỗi của hệ thống

Ví dụ:

```
try{
    c=a/b;
}catch(Exception e){
          System.out.println("Có lỗi "+e);
}
System.out.println("Sau phép chia !"); (*)
```

Câu lệnh (*) sẽ luôn được thực hiện dù mẫu số b=0 hay b!=0.

Dùng try có nhiều catch

- Trong một đoạn code có thể có nhiều ngoại lệ xảy ra nên ta sẽ dùng nhiều catch để xử lý các ngoại lệ đó.
- Các lệnh catch thường được viết theo thứ tự xuất hiện của ngoại lệ.
- Chú ý: Tất cả các ngoại lệ sẽ là lớp con của class Exception nên catch cuối cùng sẽ là Exception.

Dùng try có nhiều catch

```
public static void main(String[] args) {
    {
        try {
            int b = 0;
            int a = 10 / b;
            int c[] = {2};
            c[5] = 3;
            System.out.println("a = "+a);
        } catch (ArithmeticException e) {
            System.out.println("Phep chia cho 0 : " +e);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Vuot ngoai chi so mang : "+e);
        }
    }
}
```

```
public static void main(String[] args) {
    {
        try {
            int b = 0;
            int a = 10 / b;
            int c[] = {2};
            c[5] = 3;
            System.out.println("a = "+a);
        } catch (ArithmeticException e) {    // (1)
            System.out.println("Phep chia cho 0 : " +e);
        } catch (Exception e) {              //(2)
            System.out.println("Co loi : "+e);
        }
    }
}
```

Nếu xuất hiện ngoại lệ phép chia cho 0 thì lệnh (1) sẽ xử lý, còn các ngoại lệ khác sẽ được xử lý bởi lệnh (2). Nếu đặt (2) đổi chỗ cho (1) thì (2) sẽ xử lý luôn ngoại lệ chia cho 0 vì như thế không cần (1) nữa. Vì thế không thể thay đổi vị trí giữa lệnh (1) và lệnh (2)

Cấu trúc try...catch lồng nhau:

```
try {
    int a = 2;  //cho a=0 hoặc a=1 hoặc a=2 để test
    int b = 42 / a;
    System.out.println("Ket qua phep chia b=" + b);
    try {
        if (a == 1) {
            a = a / (a - a);
        }
        if (a == 2) {
            int c[] = {2};
            c[5] = 3;
        }
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Có lỗi vượt quá chỉ số mảng");
    }
} catch (ArithmeticException e) {
    System.out.println("Có lỗi: mẫu số = 0");
```

Cấu trúc sử dụng finally:

Trong khối finally sẽ chứa một khối mã sẽ thực hiện sau khối try/catch. Khối finally sẽ được thực hiện dù ngoại lệ có xuất hiện hay không. Tuy nhiên, mỗi try sẽ yêu cầu có ít nhất 1 catch hoặc 1 finally.

- try ⇨ catch ⇨ finally
- try ⇨ catch
- try ⇨ finally

Ví dụ:

```
static void proA(){
        try{
                System.out.println("Trong phương thức proA");
                throw new RuntimeException("Demo");      (1)
        }
        finally{
                System.out.println("Trong khối finally của proA");
        }

}
```

```
static void proB(){
        try{
                System.out.println("Trong phương thức proB");
                return;
        }
        finally{
                System.out.println("Trong khối finally của proB");
        }

}
```

```
static void proC(){
      try{
            System.out.println("Trong phương thức proC");

      }
      finally{
            System.out.println("Trong khối finally của proC");
      }
}
```

Nhận xét:

- ở procA() có tạo ra ngoại lệ mà vẫn chạy khối finally
- ở procB() có return mà vẫn chạy khối finally.

Từ khóa throws

Từ khóa throws được sử dụng trong method dùng để đề xuất các ngoại lệ có thể xảy ra trong method đó. Có những method sử dụng một số lệnh mà các lệnh đó có thể xảy ra ngoại lệ 'checked' nên chúng ta bắt buộc phải xử lý ngoại lệ đó. Ví dụ khi xử lý các lệnh thao tác với file, phải xử lý ngoại lệ 'checked' FileNotFoundException. Tất cả các ngoại lệ được khai báo bởi throws đều phải được xử lý, nếu không có đủ sẽ bị thông báo lỗi.

Ví dụ 1:

```
public void ghifile() throws IOException{
        FileWriter file = new FileWriter("data.txt");
        file.write("Xu ly ngoai le trong java");
        file.write(100);
        System.out.println("Da ghi xong !");
        file.close();
    }
```

Ví dụ 1 (tiếp):

```
public static void main(String[] args)    {
    try {
        throwsexampel obj = new throwsexampel();
        obj.ghifile();
        System.out.println("Su dung tu khoa throws");
    } catch (IOException ex) {
        System.out.println("Co loi: "+ex);
    }
}
```

Ví dụ 2: Dùng cách throws trong phương thức main

```
public void ghifile() throws IOException{
        FileWriter file = new FileWriter("data.txt");
        file.write("Xu ly ngoai le trong java");
        file.write(100);
        System.out.println("Da ghi xong !");
        file.close();
    }
    public static void main(String[] args) throws IOException {
            throwsexampel obj = new throwsexampel();
            obj.ghifile();
        System.out.println("Su dung tu khoa throws");
    }
}
```

Sử dụng từ khóa throw:

- Thông thường các exception sẽ được 'ném' ra bởi hệ thống Java runtime. Tuy vậy ta vẫn có thể lập trình để 'ném' ra các ngoại lệ khi gặp một tình huống nào đó trong khi lập trình.
- Trong một phương thức có thể throw nhiều ngoại lệ.
- Có 2 cách để 'ném' (throw) ra các ngoại lệ:
  - Dùng toán tử new
  - Đưa 1 tham số vào mệnh đề catch.
- Ví dụ:

```
if (check==0)
    throw new NullPointerException();
```

```
public class throwDemo {
    static void demoProc() {
        try {
            throw new NullPointerException("demo");
        } catch (NullPointerException e) {
            System.out.println("Ben trong xu ly ngoai lệ demoPro");
            throw e;
        }
    }
    public static void main(String args[]) {
        try {
            demoProc();
        } catch (NullPointerException e) {
            System.out.println("Trong main, tiep tuc xu ly ngoai le");
        }
    }
}
```

Chúng ta có thể tự viết class xử lý ngoại lệ của riêng mình bằng cách kế thừa class Exception của Java:

```
public class myexception extends Exception {

    private int message;

    myexception(int a) {
        message = a;
    }

    @Override
    public String toString() {
        return "My exception " + message;
    }
```

```
    static void tinhtoan(int a) throws myexception {
        if (a > 10) {
            throw new myexception(a);
        }
        System.out.println("Normal exit");
    }

    public static void main(String args[]) {
        try {
            tinhtoan(1);          //khong tao ra exception
            tinhtoan(20);         //tao ra exception
        } catch (myexception e) {
            System.out.println("Caugh " + e);
        }
    }
}
```

Tổng kết:

- Ngoại lệ là các lỗi chỉ xảy ra khi chạy chương trình
- Khi gặp ngoại lệ thì chương trình lập tức dừng lại
- Dùng try... catch để xử lý ngoại lệ theo ý đồ của người lập trình.
- Dùng try có nhiều catch
- Dùng try lồng nhau
- Sử dụng try-catch-finally
- Sử dụng từ khóa throws
- Sử dụng từ khóa throw