Java Fundamental

Bài 09

Exception

Company: DEVPRO VIỆT NAM

Website: devpro.edu.vn

Design by Minh An

Exception?

- Ngoại lệ (exception) là một sự kiện xảy ra trong quá trình thực thi của chương trình và làm gián đoạn luồng chạy bình thường của nó.
- Một ngoại lệ có thể xảy ra với nhiều lý do khác nhau, nó nằm ngoài dự tính của chương trình.
- Ngoại lệ xảy ra có thể người dùng, lập trình viên hoặc lỗi của nguồn dữ liệu vật lý.
  - Nhập dữ liệu không hợp lệ
  - Truy cập phần tử mảng không hợp lệ
  - Mở file không tồn tại …

Design by Minh An

Ví dụ

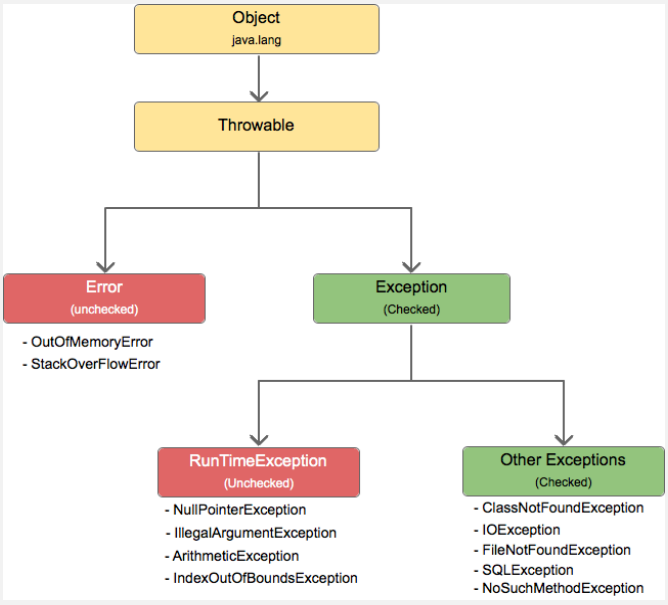
- Chương trình có câu lệnh thực hiện phép chia số a cho số b. Khi thực hiện chương trình giá trị số b nhận được là 0, phép chia a / b xảy ra lỗi, được gọi là ngoại lệ, chương trình bị gián đoạn.

```
3  public class Exam01_DivideByZero {
4      public static void main(String[] args) {
5          int a = 10, b = 0;
6          int c = a / b; //Xảy ra ngoại lệ
7          System.out.println("a = " + a);
8
9          String numString = "323A2";
10         int num = Integer.parseInt(numString); //Xảy ra ngoại lệ
11         System.out.println("Num = " + num);
12     }
13 }
```


Exception & Error

- Lỗi (Error): là những vấn đề nghiêm trọng xảy ra trong quá trình chạy, mà chương trình không nên bắt và xử lý, khi lỗi xảy ra chương trình sẽ ngừng hoạt động. Hầu hết các lỗi là biểu hiện của những điều kiện bất thường mà hệ thống gặp phải.
- Ngoại lệ (Exception) là vấn đề ít nghiêm trọng hơn và chương trình vẫn có thể chạy được nếu ngoại lệ được bắt và xử lý.
- Ví dụ:
  - OutOfMemoryError là một lỗi (hết bộ nhớ) và chúng ta không thể làm gì nhiều với trường hợp này, chương trình sẽ ngừng hoạt động. Hay lỗi tràn số.
  - ArrayIndexOutOfBoundsException là một ngoại lệ (chỉ số ở ngoài phạm vi chỉ số của mảng) trong trường hợp này hàm được gọi sẽ không chạy được, nhưng chương trình vẫn có thể chạy tiếp nếu ngoại lệ sinh ra được bắt và xử lý.

Exception Hierarchy



Checked & Unchecked Exception Handling

- **Checked Exception:**
  - Ngoại lệ mà trình biên dịch có thể nhận biết được, nghĩa là có thể phát hiện ra được tại thời điểm biên dịch code (compile time).
  - Khi một đoạn code phát sinh ra một checked Exception thì nó buộc phải được xử lý thì chương trình mới có thể biên dịch thành công (try-catch or throw).
- **Unchecked Exception:** là những ngoại lệ mà trình biên dịch không thể nhận biết được, unchecked exception thường phát sinh trong quá trình chạy (trỏ null, tham số không hợp lệ, ...)

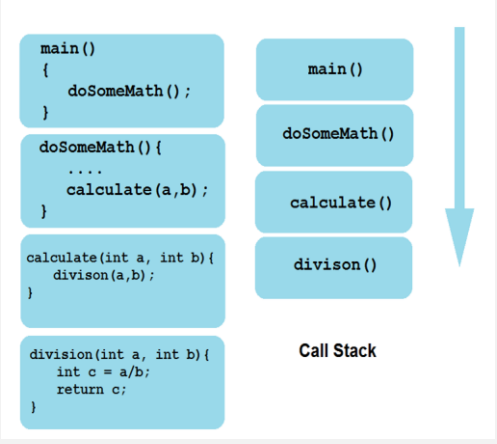
Xử lý ngoại lệ - Một số khái niệm liên quan

1. Call Stack

- Call stack là một danh sách tuần tự các methods đã được gọi để tới được một method nhất định nào đó. Trong trường hợp này là danh sách các method đã được gọi để tới được method mà tại đó xảy ra ngoại lệ.

2. Exception handler

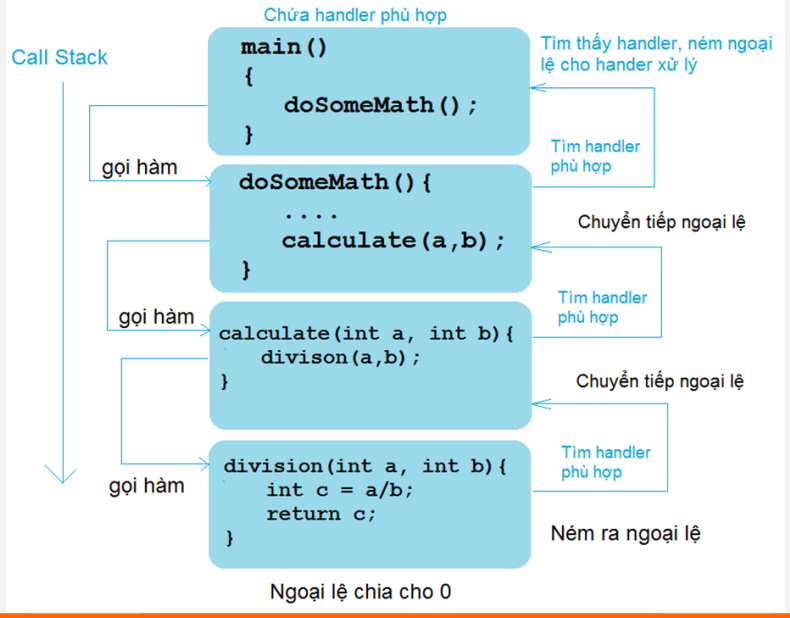
- Là khối code mà tại đó ngoại lệ được xử lý, một Exception handler có thể xác định rõ loại ngoại lệ nào mà nó có thể xử lý.



Máy ảo JVM xử lý ngoại lệ

- Khi xảy ra ngoại lệ trong một method, method đó sẽ tạo ra một đối tượng Exception và chuyển nó ra cho hệ thống (JVM).
- Đối tượng Exception chứa tên và mô tả của ngoại lệ đã xảy ra.
- Việc tạo đối tượng Exception và chuyển nó ra cho hệ thống được gọi là ném ra ngoại lệ (throwing an Exception).
- Có thể có một danh sách các method được gọi trước khi tới được method nơi mà ngoại lệ xảy ra. Danh sách này được gọi là Call Stack. Và lúc này các thủ tục sẽ diễn ra như sau:

Máy ảo JVM xử lý ngoại lệ



Máy ảo JVM xử lý ngoại lệ

- Hệ thống sẽ rà soát trong call stack để tìm method chứa exception handler phù hợp có thể xử lý được ngoại lệ.
- Hệ thống bắt đầu tìm kiếm từ method nơi mà ngoại lệ xảy ra, và truy ngược call stack về những method mà đã gọi method sinh ngoại lệ.
- Nếu tìm thấy một handler phù hợp hệ thống sẽ ném ngoại lệ cho handler đó xử lý. Một handler phù hợp là một handler mà kiểu của ngoại lệ nó có thể xử lý khớp với kiểu của ngoại lệ xảy ra.
- Nếu sau khi đã tìm trong tất cả các method trong call stack và không tìm thấy handler phù hợp, hệ thống sẽ ném ngoại lệ cho handler mặc định xử lý (default exception handler - được tích hợp sẵn trong máy ảo). Handler này sẽ in các thông tin của ngoại lệ ra console và hủy chạy chương trình.

Xử lý ngoại lệ: khối try - catch

- Để tránh việc chương trình bị buộc dừng bởi hệ thống, lập trình viên cần phải xử lý ngoại lệ bằng cách viết các exception handler phù hợp. Trong java khối lệnh xử lý ngoại lệ được điều khiển bởi 5 từ khóa: try, catch, throw, throws và finally.
- Đầu tiên, các đoạn code mà có thể phát sinh ngoại lệ phải được đưa vào trong khối try. Theo sau khối try là (các) khối catch với các **Exception** được chỉ định trong cặp ngoặc (…).
- **Cú pháp:**

```
try{
      // code có thể gây ra exception
} catch (Exception e) {
      // code xử lý ngoại lệ
} catch (AnotherException e) {
      // code xử lý ngoại lệ
}
```


Xử lý ngoại lệ: khối try - catch

- Khi ngoại lệ xảy ra trong khối try{…}, nó sẽ được ném ra và có thể được bắt bởi một trong số các khối catch(…){…} theo sau nếu kiểu của **exception** ném ra khớp với kiểu của **exception** chỉ định cho khối catch đó.
- Khi ngoại lệ được ném ra, chương trình dừng thực thi các lệnh phía sau lệnh gây ra ngoại lệ, và chuyển xuống thực thi các lệnh trong khối catch tương ứng mà có thể bắt được ngoại lệ hiện tại. Các khối catch chính là các **exception handler.**
- Các ngoại lệ được sinh bởi hệ thống sẽ tự động được ném ra.

Xử lý ngoại lệ: khối try - catch

```
public class Exam01_DivideByZero {
    public static void main(String[] args) {
        int a = 10, b = 0;
        try {
            int c = a / b; //Xảy ra ngoại lệ
            System.out.println("a = " + a);
        }catch(ArithmeticException exc) {
            System.out.println("Loi: Chia cho khong");
        }

        String numString = "323A2";
        try {
            int num = Integer.parseInt(numString); //Xảy ra ngoại lệ
            System.out.println("Num = " + num);
        }catch(NumberFormatException exc) {
            System.out.println("Loi: Chuoi so khong hop le");
        }
    }
}
```


Xử lý ngoại lệ: ném ngoại lệ

- Lập trình viên cũng có thể chủ động ném ra các ngoại lệ bằng cách dùng từ khóa throw.

```
try{
   // chủ động ném ngoại lệ
   throw exception_object;
} catch (Exception e){
   // code xử lý ngoại lệ
} catch (AnotherException e){
   // code xử lý ngoại lệ
}
```

Xử lý ngoại lệ: ném ngoại lệ

```java
package vn.devpro.dividebayzero;


public class DivideByZeroException extends Exception{
    private String messageError;
    public DivideByZeroException (String messageError) {
        this.messageError = messageError;
    }
    public String getMessage() {
        return messageError;
    }
}
```


Xử lý ngoại lệ: ném ngoại lệ

```java
package vn.devpro.dividebayzero;

public class DivideByZeroUtils {
    public static void dividing(double a, double b)
                throws DivideByZeroException{
        if (b == 0) {
            throw new DivideByZeroException("Loi: So chia bang 0!");
        }
        double d = a / b;
        System.out.println(a + " / " + b + " = " + d);
    }
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        double a = 4, b = 0;
        try {
            dividing(a,b);
        }
        catch(DivideByZeroException exc) {
            System.out.println(exc.getMessage());
        }
    }
}
```

Xử lý ngoại lệ: ném ngoại lệ

```
1  package vn.devpro.markexception;
2
3  public class LowerMarkException extends Exception{
4      private String errorMessage;
5
6      public LowerMarkException(String errorMessage) {
7          super();
8          this.errorMessage = errorMessage;
9      }
10     @Override
11     public String getMessage() {
12         return errorMessage;
13     }
14
15 }
16
```


Xử lý ngoại lệ: ném ngoại lệ

```
1  package vn.devpro.markexception;
2
3  public class HigherMarkException extends Exception{
4      private String errorMessage;
5
6      public HigherMarkException(String errorMessage) {
7          super();
8          this.errorMessage = errorMessage;
9      }
10     @Override
11     public String getMessage() {
12         return errorMessage;
13     }
14 }
15
```

Xử lý ngoại lệ: ném ngoại lệ

```
1  package vn.devpro.markexception;
2  public class MarkExceptionUtil {
3      public static void checkMark(double mark)
4              throws LowerMarkException, HigherMarkException{
5          if (mark < 0)
6              throw new LowerMarkException("Diem nho hon 0");
7          else if (mark > 10)
8              throw new HigherMarkException("Diem lon hon 10");
9          else
10             System.out.println("Diem hop le");
11     }
12     public static void main(String[] args) {
13         double mark = 10;
14         try {
15             checkMark(mark);
16             System.out.println("Xep loai hoc tap cho sinh vien");
17         }catch(LowerMarkException elow) {
18             System.out.println(elow.getMessage());
19         }catch(HigherMarkException ehigh) {
20             System.out.println(ehigh.getMessage());
21         }
22     }
23 }
```


Xử lý ngoại lệ: khối try – catch – finally

- Một khối try - catch cũng có thể được theo sau bởi khối finally. Các lệnh trong finally sẽ luôn được chạy cho dù ngoại lệ có xảy ra trong khối try hay không.

```
try{
        // code có thể gây ra exception
} catch (Exception e) {
        // code xử lý ngoại lệ
}
finally {
        //code luôn được thực thi
}
```

- Khối finally thường được dùng để thực thi các lệnh mà cần thiết phải được thực hiện cho dù ngoại lệ có xảy ra hay không, như đóng file đang mở, ngắt kết nối với database.

Xử lý ngoại lệ: khối try – catch – finally

Luồng chạy của các khối try-catch-finally

- Nếu có ngoại lệ xảy ra trong khối try các lệnh phía sau lệnh gây ngoại lệ sẽ không được chạy, mà chương trình chuyển xuống chạy các lệnh trong khối catch tương ứng, nếu khối catch đó có thể bắt được ngoại lệ. Sau khi thực thi xong khối catch, khối finally sẽ được thực thi.
- Nếu không có khối catch nào bắt được ngoại lệ khối finally sẽ được thực thi, sau đó ngoại lệ sẽ được chuyển cho **default exception handler** của **JVM** xử lý.
- Nếu không có ngoại lệ xảy ra trong khối try, thì khối try sẽ thực thi hết các lệnh một cách bình thường sau đó khối finally sẽ được thực thi, sau đó chương trình tiếp tục thực thi các lệnh còn lại nằm phía sau khối try-catch-finally.
- Trong bất cứ trường hợp nào các lệnh trong finally vẫn luôn được thực thi.

Khối try – catch – finally

```
3  public class Sach {
4      String name;
5      int amount;
6      public Sach(String name, int amount) {
7          super();
8          this.name = name;
9          this.amount = amount;
10     }
11     public int getAmount() {
12         return amount;
13     }
14     public void setAmount(int amount) {
15         this.amount = amount;
16     }
17 }
18
```

Khối try – catch – finally

```
public class IllegalExceptionJavaUtils {
    public static void buy(Sach s, int buyAmount) {
        if (buyAmount > s.getAmount()) {
            throw new IllegalArgumentException("So luong sach khong du");
        }
        System.out.println("Ban da dat mua " + buyAmount + " quyen sach");
        s.setAmount(s.getAmount() - buyAmount);
    }
    public static void main(String[] args) {
        Sach s = new Sach("Java in application", 3);
        try {
            buy(s, 3);
        }catch(IllegalArgumentException ex) {
            System.out.println(ex.getMessage());
        }
        finally {
            System.out.println("Cam on ban");
        }
    }
}
```

Updates A

Design by Minh An

Bài tập

- Cho một danh sách các cầu thủ của một câu lạc bộ bóng đá nam, mỗi cầu thủ gồm: họ đệm, tên, tuổi, giới tính, vị trí (tiền đạo / tiền vệ / hậu vệ / thủ môn).
- Cài đặt chương trình thực hiện các yêu cầu
  - Nhập danh sách cầu thủ. Xử lý ngoại lệ: tên, vị trí không được để trống, tuổi từ 16 đến 40, giới tính phải là nam.
  - Hiển thị danh sách cầu thủ.
  - Chèn thêm một cầu thủ vào vị trí k trong danh sách, xử lý trường hợp vị trí k không hợp lệ.
  - Sắp xếp danh sách theo tên cầu thủ
  - …

Design by Minh An