

Bài 06

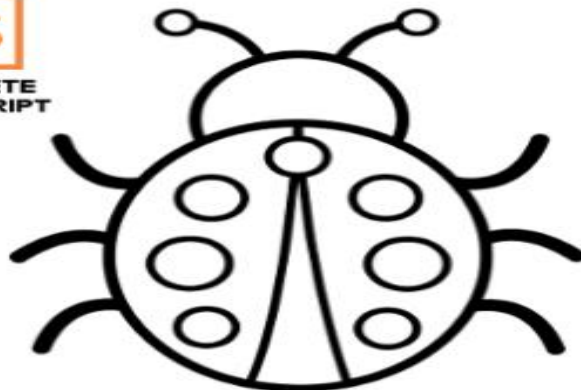
Bẫy lỗi ngoại lệ (Exception)

Nội dung bài học

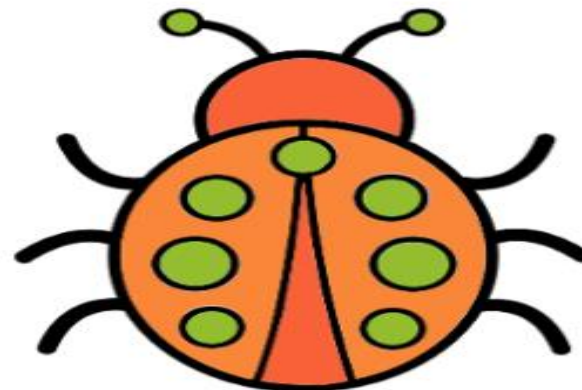
- ❖ Lỗi phần mềm là gì?
- ❖ Các cấp độ lỗi: lỗi biên dịch, lỗi thực thi, lỗi logic
- ❖ Cách xử lý với lỗi: **Checked** vs **Unchecked**
- ❖ Cách thức bắt lỗi: try...catch exception
- ❖ Cách thức gỡ lỗi logic bằng công cụ debug

Lỗi phần mềm

- ❖ Lỗi là sự khác nhau của kết quả thực tế và kết quả mong đợi
- ❖ Lỗi là một loại của lỗi phần mềm
- ❖ Lỗi có thể được giới thiệu như là kết quả của việc không hoàn thành hoặc sai yêu cầu hoặc do vấn đề nhập dữ liệu, thao tác của con người



BUG



FEATURE

Code anh không có bug, tất cả chỉ là tính năng



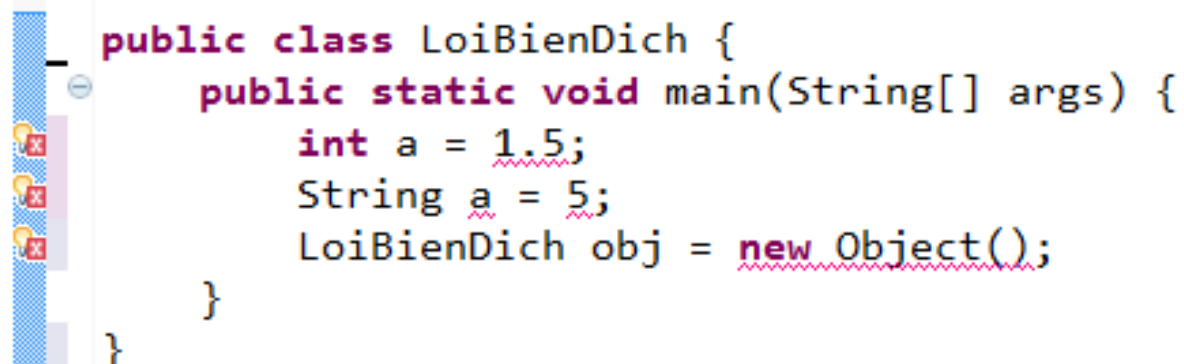
Các cấp độ lỗi

❖ Các loại lỗi chính thường gặp

- Lỗi biên dịch
- Lỗi logic
- Lỗi thực thi

Lỗi biên dịch – Compile error

❖ **Tên gọi:** Lỗi cú pháp, lỗi biên soạn. Thực chất nó đều là compile-time error



```
public class LoiBienDich {  
    public static void main(String[] args) {  
        int a = 1.5;  
        String a = 5;  
        LoiBienDich obj = new Object();  
    }  
}
```

❖ **Lỗi cú pháp:** Một số editor sẽ hiển thị thông báo lỗi

❖ **Kết luận:** "Compiler Error là loại lỗi xảy ra trong quá trình biên dịch code (compile time) thành ứng dụng (build application) và bạn không thể nào bỏ qua lỗi này nếu muốn build application thành công"



Lỗi logic – luận lý



```
public class LoilLogic {  
    public static void main(String[] args) {  
        double toan=10;  
        double ly=8;  
        double hoa=9;  
        // SAI VỚI THỰC TẾ  
        double dtbf = (toan+ly+hoa)/5;  
        // THÔNG THƯỜNG LÀ VẬY  
        double dtb=(toan+ly+hoa)/3;  
        System.out.println(dtb + dtbf);  
    }  
}
```

Là loại lỗi khó tìm thấy và sửa chữa nhất vì dấu hiệu lỗi **không thể hiện rõ**. Thông thường các chương trình chạy

thành công, nhưng nó không trả về kết quả như mong đợi. Trình biên dịch không thể chuẩn đoán lỗi luận lý, do đó lập trình viên phải là người **kiểm tra toàn bộ từng dòng code** của mình và đảm bảo chương trình chạy đúng như mong đợi.



Lỗi logic – luận lý

❖ NGUYÊN NHÂN:

- Bạn chưa thật sự hiểu yêu cầu của chương trình
- Bạn chưa hiểu rõ các hoạt động của từng dòng code trong chương trình mình viết
- Bạn đã bất cẩn trong khi lập trình
- Phát hiện càng muộn thì càng gây thiệt hại và chi phí càng cao
- VD: Calculate in supermarket

❖ CÁC TRÁNH LỖI LUẬN LÝ:

- Tuân thủ các bước trong thiết kế phần mềm: Xác định yêu cầu, phân tích yêu cầu và thiết kế giải thuật
- Hãy thực hiện mỗi dòng code cẩn thận, code phải có tâm và đúng logic của chương trình
- Debug chương trình để phát hiện lỗi



Lỗi thực thi

- ❖ Xảy ra trong quá trình thực thi chương trình
- ❖ Chương trình dừng lại ngay lập tức, toàn bộ phần mã chương trình phía sau không được thực hiện (crash)

- ❖ Nguyên nhân

Khi thực hiện xử lý tính toán, một **biểu thức, phương thức** có vấn đề sau khi thực hiện => Sinh ra một ngoại lệ (exception) trên console và dừng chương trình.

- Nhập vào số nguyên
- Phép toán chia cho 0
- Vượt quá kích thước mảng
- Khởi tạo giá trị cho đối tượng chưa tồn tại
- Tạo mới file
- Mở hidden files trong hệ thống

Ngoại lệ - Exception

❖ Exception là gì ?

- Là events xảy ra trong quá trình thực thi chương trình, exception disrupt the normal flow of instructions
- Trong Java, Exception là một object tạo ra các phương thức bao gồm
 - Thông tin lỗi, hình thức lỗi
 - Trạng thái của chương trình khi xảy ra lỗi ...
- Exception objects can be thrown and caught

❖ Ngoại lệ - Exception

❖ Exception được sử dụng biểu thị các hình thức lỗi khác nhau của điều kiện lỗi

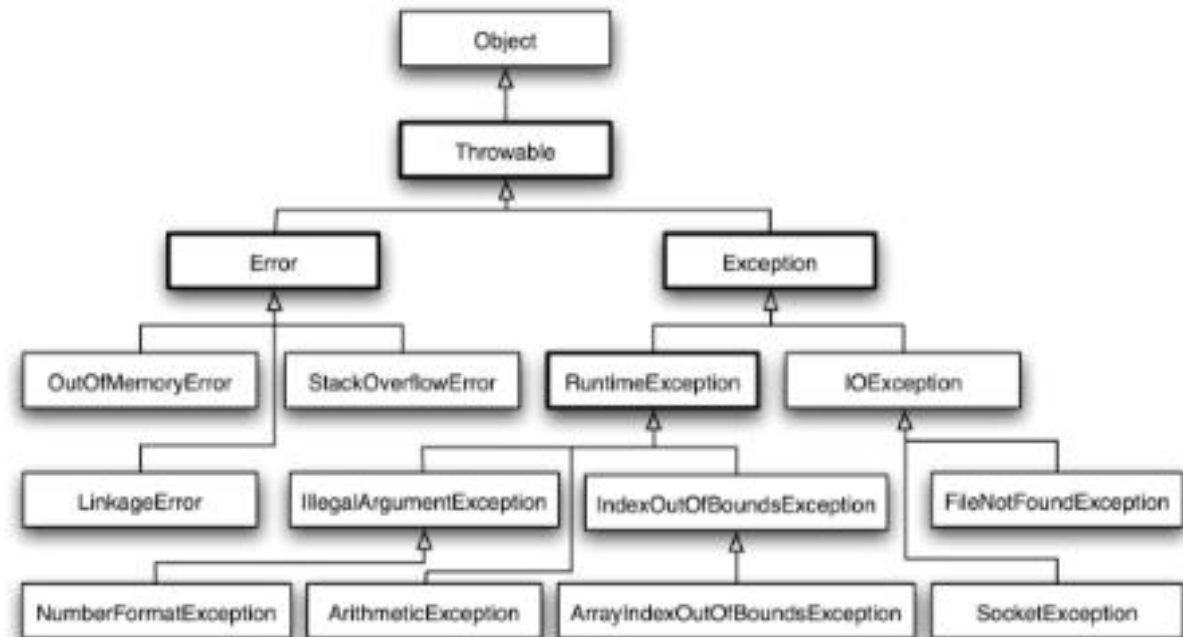
- JVM Errors:

- OutOfMemoryError
- StackOverflowError
- LinkageError

System errors:

- FileNotFoundException
- IOException
- SocketTimeoutException
- Programming errors:
- NullPointerException
- ArrayIndexOutOfBoundsException
- ArithmeticException

❖ Exception type



TẠI SAO PHẢI DÙNG EXCPTION ?

Ngoại lệ - Exception

- ❖ Tất cả Exception và Error được kế thừa từ java.lang.Throwable class
- ❖ Throwable object có thể được thrown và caught
- ❖ java.lang.Error (JVM Error),
- ❖ java.lang.Exception (System Error)
- ❖ java.lang.RuntimeException (Programming Error)

```
public class java.lang.Throwable extends Object
    implements java.io.Serializable {
    public Throwable();
    public Throwable(String msg);
    public Throwable(String msg, Throwable cause);
    public Throwable(Throwable cause);
    public String getMessage();
    public String getLocalizedMessage();
    public Throwable getCause();
    public Throwable initCause(Throwable cause);
    public String toString();
    public void printStackTrace();
    public void printStackTrace(java.io.PrintStream);
    public void printStackTrace(java.io.PrintWriter);
    public Throwable fillInStackTrace();
    public StackTraceElement[] getStackTrace();
    public void setStackTrace(StackTraceElement[] stackTrace);
}
```



Ngoại lệ - Exception

- ❖ Ví dụ: NullPointerException xảy ra khi đối tượng chưa được khởi tạo và gán cho tham chiếu

```
package controller;

public class ExceptionDemo {

    public static void main (String[] args) {
        ExceptionDemo item = null;
        args = new String[2];
        args[0] = "4";
        args[1] = "0";
        System.out.println(item.divideArray(args));
    }

    private int divideArray(String[] array) {
        String s1 = array[0];
        String s2 = array[1];
        return divideStrings(s1, s2);
    }
}
```

Problems @ Javadoc Declaration Console X Diagrams

```
<terminated> ExceptionDemo [Java Application] C:\DevPrograms\Java\jdk1.8.0_111\bin\javaw.exe (Jul 5, 2017, 1
Exception in thread "main" java.lang.NullPointerException
    at controller.ExceptionDemo.main(ExceptionDemo.java:10)
```



Checked vs Unchecked Exception

- ❖ There is a lot of controversy around checked vs. unchecked exceptions.
- ❖ RuntimeExceptions are unchecked — that is, the compiler does not enforce (check) that you handle them explicitly.
- ❖ All other Exceptions, Errors are checked — that is, the compiler enforces that you handle them explicitly.

Difference between checked and unchecked exceptions

1) Checked Exception

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.



Exception – Xử lý ngoại lệ

- ❖ Cách I : Sử dụng khối try catch
- ❖ Cách II: Sử dụng throw, throws để ném ngoại lệ

Exception – Xử lý ngoại lệ

❖ KHỐI TRY CATCH

```
package controller;

public class ExceptionDemo {

    public static void main (String[] args) {
        ExceptionDemo item = new ExceptionDemo();
        args = new String[2];
        args[0] = "4";
        args[1] = "0";
        try{
            System.out.println(item.divideArray(args));
        } catch (Exception ex){
            System.out.println(ex.toString());
        }
        System.out.println("Trở lại chương trình chính !!!");
    }
}
```

Problems @ Javadoc Declaration Console Diagrams

<terminated> ExceptionDemo [Java Application] C:\DevPrograms\Java\jdk1.8.0_111\bin\javaw.exe (Jul 5, 2017, 10
java.lang.ArithmeticException: / by zero
Trở lại chương trình chính !!!

- Đặt đoạn mã có khả năng xảy ra ngoại lệ trong khối **try**
- Đặt đoạn mã xử lý ngoại lệ trong khối **catch**
- Khi xảy ra ngoại các câu lệnh trong khối catch sẽ được thực hiện tùy vào kiểu của ngoại lệ.
- Sau khi thực hiện xong khối catch, điều khiển sẽ được trả lại cho chương trình



Exception – Xử lý ngoại lệ

❖ Ném ngoại lệ với throw, throws

```
    }*/
    System.out.println(item.divideArray(args));
    System.out.println("Welcome back to main pro !!!");
}

private int divideArray(String[] array) {
    String s1 = array[0];
    String s2 = array[1];
    int result = 1;
    try {
        result = divideStrings(s1, s2);
    } catch (NumberFormatException ex){
        System.out.println("Haizz, appear a exception :((( ");
    }
    return result;
}

private static int divideStrings(String s1, String s2) throws NumberFormatException{
    int i1 = 0 ;
    int i2 = 0;
    try{
        i1 = Integer.parseInt(s1);
        i2 = Integer.parseInt(s2);
    } catch (NumberFormatException ex){
        // throw new NumberFormatException();
        throw ex;
    }
    return divideInts(i1, i2);
}
```




Exception – Xử lý ngoại lệ

❖ NÉM NGOẠI LỆ THROW THROWS

All exceptions and errors extend from a common `java.lang.Throwable` parent class. Only Throwable objects can be thrown and caught.

- Thay vì try catch (caught) bắt ngoại lệ đó ta có thể ném (thrown) nó đi cho một nơi khác xử lý
- Thực hiện :
 - Trong khối catch thay vì xử lý hoặc thông báo lỗi
 - Ta thực hiện throw e ; để ném ngoại lệ



Tạo một lớp Exception Class

- ❖ Nếu bạn muốn tạo một Exception để sở hữu.
 - Đặt tên *Exception hợp lý.
 - Quyết định Exception là checked hay unchecked
 - Check: extends Exception
 - Unchecked: extends RuntimeException
 - Định nghĩa constructor để gọi đến parent's constructor, gọi messages, printStackTrace để hiện thông báo.



Tạo một lớp Exception Class

```
package exception;

public class InvalidAgeException extends Exception {
    /**
     *
     */
    private static final long serialVersionUID = 1L;

    public InvalidAgeException(String msg) {
        super(msg);
    }
}
```

Khi nào nên dùng try/catch ?
Khi nào dùng throw/throws?

Khi nào nên tạo một Exception Class riêng để quản lý ?

```
package controller;

import exception.InvalidAgeException;

public class AgeController {
    static void validate(int age) throws InvalidAgeException {
        if (age < 18)
            throw new InvalidAgeException("Age is not valid");
        else
            System.out.println("welcome to vote");
    }

    public static void main(String args[]) {
        try {
            validate(13);
        } catch (Exception ex) {
            System.out.println("Exception occurred: " + ex);
        }

        System.out.println("rest of the code...");
    }
}
```

Exception – Xử lý ngoại lệ

❖ KHỐI FINALLY

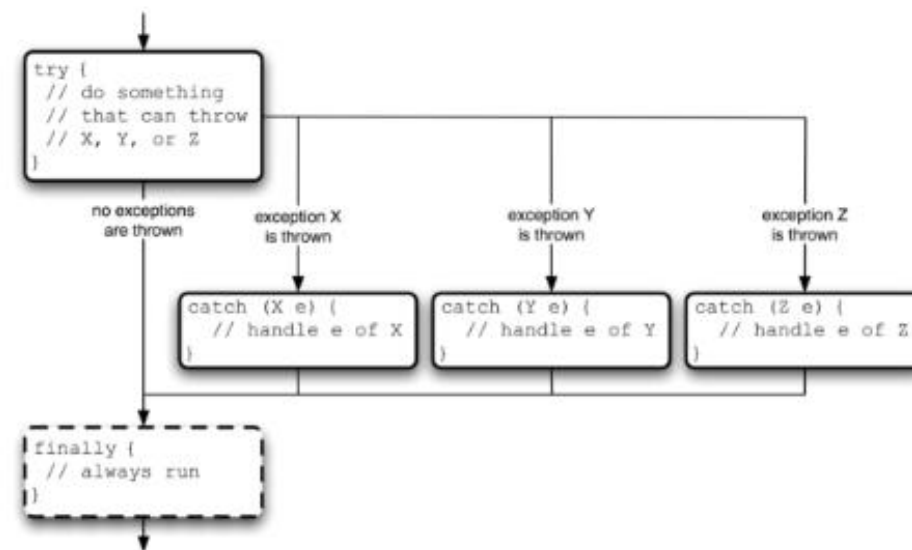
- Là block code để execute những đoạn code quan trọng như close connection, stream, etc ..
- Luôn được thực thi dù exception có handle hoặc không

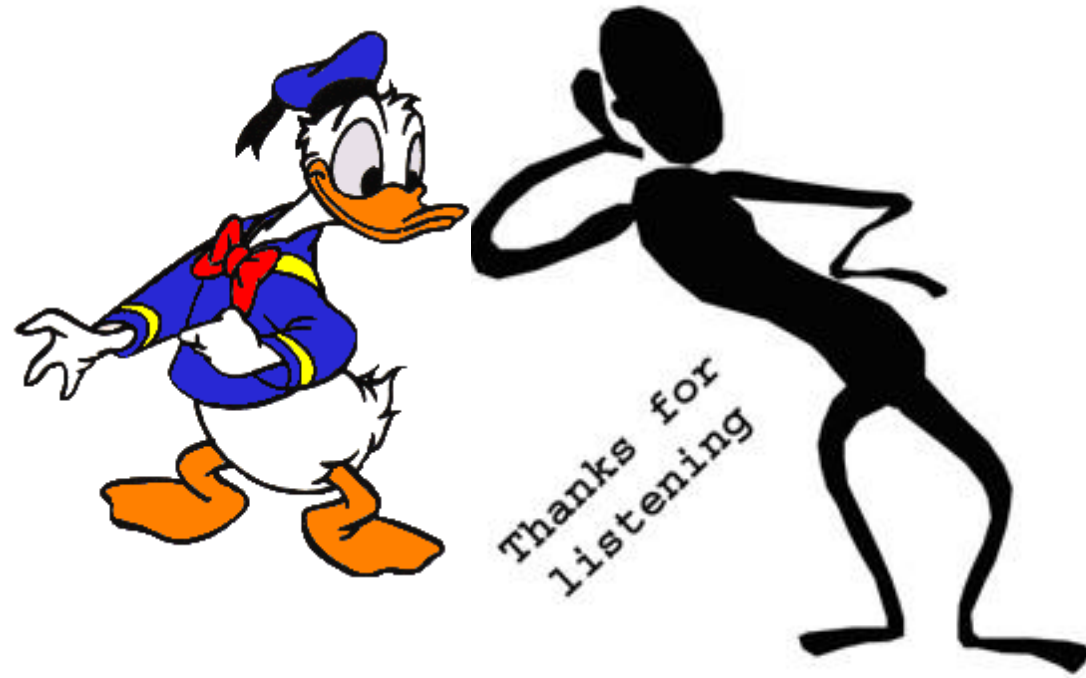


Note: If you don't handle exception, before terminating the program, JVM executes finally block(if any).

- Được sử dụng để “cleanup” code khi xảy ra ngoại lệ hoặc đóng chương trình

❖ The ‘try-catch-finally’ control structure





END