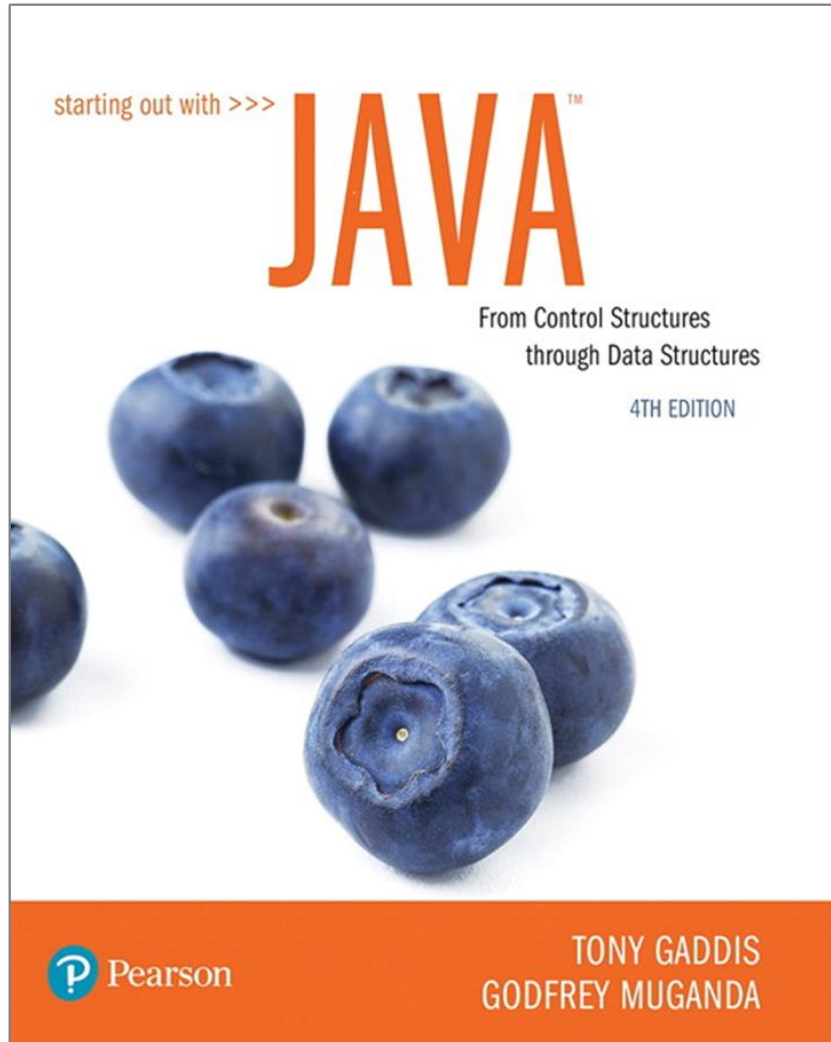


# STARTING OUT WITH JAVA™

4<sup>th</sup> Edition



## Chương 11

Ngoại lệ và xử lý tập tin nâng cao

# Chủ đề

Chương 11 sẽ xoay quanh các chủ đề chính như sau:

- Xử lý ngoại lệ
- Ném ngoại lệ
- Tìm hiểu thêm về Input/Output Streams
- Các chủ đề mở rộng:
  - Tập tin nhị phân,
  - Random Access Files, và
  - Đối tượng Serialization

# Xử lý ngoại lệ (Handling Exceptions) (1 of 2)

- Một exception là một đối tượng được tạo ra do lỗi hoặc một sự kiện không mong muốn phát sinh.
- Khi exception phát sinh thì chương trình sẽ ném thông báo – còn gọi là thrown.
- Khi phát triển chương trình, lập trình viên phải viết mã để phát hiện và xử lý các exceptions nếu chúng phát sinh.
- Nếu không xử lý ngoại lệ thì ứng dụng/phần mềm sẽ bị crash.
- Ví dụ: [BadArray.java](#)
- Java cho phép tạo các xử lý ngoại lệ - **Exception handler**.

## Xử lý ngoại lệ (2 of 2)

- Một exception handler là đoạn mã có chức năng phản hồi exception phát sinh.
- Quá trình chặn và phản hồi exception được gọi là **exception handling**.
- Trình xử lý exception mặc định được dành cho các exception không được xử lý. Nó sẽ thực hiện in ra thông báo lỗi và làm hỏng chương trình. (Crashes the program).

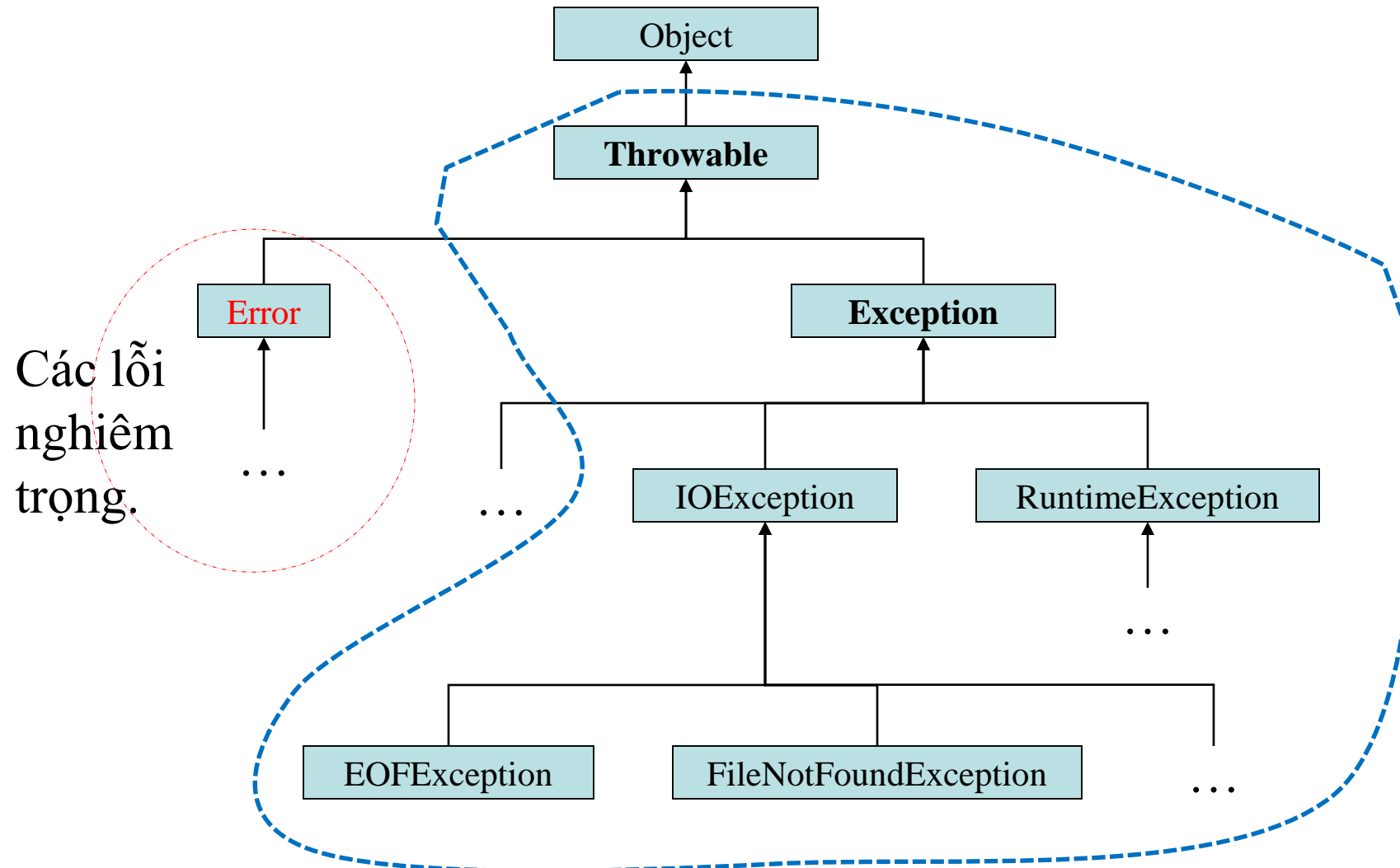
# Lớp Exception(1 of 3)

- Một **exception** là một đối tượng.
- Java API cung cấp lớp **Exception**, lớp này cho phép tạo các đối tượng exception và có thể được kế thừa bởi các lớp khác.
- Tất cả các lớp con của lớp **Exception** trong Java đều thừa kế từ lớp `Throwable`.
- Lớp `Error` và `Exception` thừa kế từ lớp `Throwable`.

## Exception Classes (2 of 3)

- Các lớp thừa kế từ lớp `Error`:
  - dành cho các exceptions được ném ra khi chương trình phát sinh lỗi nghiêm trọng...
    - lỗi nội bộ của JVM, hoặc
    - hết bộ nhớ.
- Các chương trình không nên xử lý các lỗi này vì chúng là những lỗi nghiêm trọng và đáng báo động.
- Các chương trình chỉ nên xử lý các exceptions có nguồn gốc từ lớp `Exception`.

# Lớp Exception (3 of 3)



## Xử lý ngoại lệ (1 of 6)

- Lệnh **try** được sử dụng để xử lý một exception

```
try
```

```
{
```

*Dấu ngoặc nhọn là bắt buộc.*      **(try block statements...)**

```
}
```

```
catch (ExceptionType ParameterName)
```

```
{
```

```
    (catch block statements...)
```

```
}
```

- Khối mã phía sau từ khóa try (phần trong cặp dấu ngoặc nhọn) có thể hiểu là: Khối mã này rất có thể sẽ phát sinh exception.
- Khối mã này được gọi là khối thử nghiệm – **try-block**.



## Xử lý ngoại lệ (2 of 6)

- Một *try-block* là:
  - Một hoặc nhiều câu lệnh được thực thi và các lệnh này có khả năng ném ra exception.
- Ứng dụng sẽ không tạm dừng nếu try-block ném ra exception.
- Sau try-block, lệnh **catch** bắt buộc phải có.

## Xử lý ngoại lệ (3 of 6)

- Lệnh catch khai báo tham số là các lớp con của lớp Exception hoặc lớp Exception:

**catch** (*ExceptionType ParameterName*)

– *ExceptionType* – tên lớp của đối tượng exception

– *ParameterName* - là tên biến tham chiếu đến đối tượng exception mà try-block ném ra.

- Đoạn mã sau lệnh catch được gọi là – **catch-block** (catch-block phải đặt trong dấu ngoặc nhọn).
- Các mã lệnh trong **catch-block** sẽ được thực thi nếu **try-block** ném ra exception.

## Xử lý ngoại lệ (4 of 6)

- Mã dưới đây được viết để xử lý `FileNotFoundException` khi nó được ném ra.

```
try
{
    File file = new File ("MyFile.txt");
    Scanner inputFile = new Scanner(file);
}
catch (FileNotFoundException e)
{
    System.out.println("File not found.");
}
```

*Tập tin không tồn tại nên một exception sẽ được ném ra*

*Ngoại lệ được xử lý bằng cách in ra màn hình thông báo “File not found.”*

- JVM sẽ tìm kiếm catch-block để xử lý ngoại lệ.
- Example: [OpenFile.java](#)

## Xử lý ngoại lệ (5 of 6)

- Tham số trong lệnh catch phải tương ứng với exception được ném ra.

```
try
{
    File file = new File ("MyFile.txt");
    Scanner inputFile = new Scanner(file);
}
catch (FileNotFoundException e)
{
    System.out.println("File not found.");
}
```

- Sau khi xử lý ngoại lệ, chương trình vẫn sẽ tiếp tục thực hiện các câu lệnh phía sau **catch-block**.

## Xử lý ngoại lệ (6 of 6)

- Mỗi đối tượng exception có một phương thức **getMessage** có thể được sử dụng để truy xuất thông báo lỗi.
- Ví dụ:
  - [ExceptionMessage.java](#)
  - [ParseException.java](#)

# Tính đa hình trong tham chiếu ngoại lệ

## (1 of 2)

- Xử lý ngoại lệ có thể áp dụng tính đa hình khi khai báo tham số trong lệnh catch.
- Hầu hết các ngoại lệ đều thừa kế từ lớp Exception.
- Vì thế, khi khai báo tham số trong lệnh catch là Exception cũng đồng nghĩa với việc có thể bắt hầu hết các ngoại lệ.
- Xem ví dụ:

# Polymorphic References To Exceptions

## (2 of 2)

```
try
{
    number = Integer.parseInt(str);
}
catch (Exception e)
{
    System.out.println("The following error occurred: "
        + e.getMessage());
}
```

- Phương thức `parseInt` của lớp `Integer` sẽ ném ra đối tượng của lớp `NumberFormatException` khi ngoại lệ phát sinh.
- Lớp `NumberFormatException` thừa kế từ lớp `Exception`.

# Xử lý nhiều ngoại lệ

- Các mã lệnh trong try-block có thể phát sinh nhiều hơn một ngoại lệ.
- Lệnh catch cần được viết cho từng ngoại lệ có thể được ném ra bởi try-block.
- JVM sẽ chạy catch-block của lệnh catch có tham số tương thích đầu tiên được tìm thấy.
- Các lệnh catch phải được liệt kê theo thứ tự: ***cụ thể nhất đến tổng quát nhất***.
- Ví dụ: [SalesReport.java](#), [SalesReport2.java](#)



# Bộ xử lý các ngoại lệ (1 of 3)

- Để xử lý khi ngoại lệ phát sinh có thể sử dụng nhiều lệnh **catch**.
- Tuy nhiên, một lệnh **try** chỉ có thể có một lệnh **catch** cho từng loại ngoại lệ cụ thể

```
try
{
    number = Integer.parseInt(str);
}
catch (NumberFormatException e)
{
    System.out.println("Bad number format.");
}
catch (NumberFormatException e) // ERROR!!!
{
    System.out.println(str + " is not a number.");
}
```

## Bộ xử lý các ngoại lệ (2 of 3)

- Các lệnh catch phải được liệt kê theo thứ tự: cụ thể nhất đến tổng quát nhất.
- Lớp `NumberFormatException` thừa kế từ lớp `IllegalArgumentException`.

*Tổng quát*  
↓  
**ERROR**  
↓  
*Cụ thể*

```
try
{
    number = Integer.parseInt(str);
}
catch (IllegalArgumentException e)
{
    System.out.println("Bad number format.");
}
catch (NumberFormatException e) // ERROR!!! →
{
    System.out.println(str + " is not a number.");
}
```

Lớp `NumberFormatException` là trường hợp cụ thể của lớp `IllegalArgumentException`.

## Bộ xử lý các ngoại lệ (3 of 3)

- Đoạn mã ở slide trước có thể được sửa bằng cách triển khai các lệnh catch đúng thứ tự.

```
try
{
    number = Integer.parseInt(str);
}
catch (NumberFormatException e)
{
    System.out.println(str + " is not a number.");
}
catch (IllegalArgumentException e) //OK
{
    System.out.println("Bad number format.");
}
```

# Lệnh `finally` (1 of 2)

- Lệnh `try` ngoại trừ việc đi kèm với lệnh `catch` thì còn có lệnh **`finally`** (tùy chọn).
- Nếu sử dụng thì lệnh `finally` phải được đặt ở cuối cùng – sau `catch`-block.

```
try
{
    (try block statements...)
}
catch (ExceptionType ParameterName)
{
    (catch block statements...)
}
finally
{
    (finally block statements...)
}
```

## Lệnh **finally** (2 of 2)

- Tương tự try-block và catch-block, **finally-block** có thể chứa một hoặc nhiều câu lệnh.
  - Các lệnh trong khối này luôn luôn được thực thi sau khi try-block được thực thi, và sau khi khối catch-block thực thi nếu có ngoại lệ ném ra.
- Các lệnh trong finally-block luôn được thực thi dù có ngoại lệ phát sinh hay không.


# The Stack Trace

- Call stack hay còn gọi là Ngăn xếp cuộc gọi là danh sách các phương thức đã được gọi để có được một phương thức cụ thể.
- Một *stack trace* là danh sách tất cả phương thức trong call stack.
- Nó chỉ ra:
  - phương thức đang thực thi khi một ngoại lệ xảy ra và tất cả các phương thức đã được gọi để thực thi phương thức đó.
- Example: [StackTrace.java](#)

## Multi-Catch (Java 7)

- Bắt đầu từ Java 7, lệnh catch có thể chỉ định nhiều hơn một tham số ngoại lệ hay nói cách khác là: Có thể bắt nhiều hơn một ngoại lệ trong một l

```
try
{
}
catch (NumberFormatException | InputMismatchException ex)
{
}
```



Separate the exceptions with  
the | character.

## Uncaught Exceptions (1 of 2)

- Khi một ngoại lệ được ném ra, nó không thể bị bỏ qua.
- Nó phải được xử lý bởi chương trình hoặc bởi trình xử lý ngoại lệ mặc định.
- Khi mà trong một phương thức ném ra một ngoại lệ :
  - quá trình thực thi bình thường của phương thức dừng lại, và
  - JVM tìm kiếm một trình xử lý ngoại lệ tương thích (catch).



## Uncaught Exceptions (2 of 2)

- Nếu không có trình xử lý ngoại lệ bên trong phương thức:
  - quyền kiểm soát của chương trình được chuyển cho phương thức trước đó trong ngăn xếp cuộc gọi. Điều này lặp lại đến khi JVM tìm được trình xử lý ngoại lệ hoặc quyền kiểm soát quay lại điểm đầu vào (phương thức main).
- Nếu quyền kiểm soát quay về phương thức main:
  - phương thức chính phải xử lý ngoại lệ hoặc
  - chương trình bị tạm dừng và trình xử lý ngoại lệ mặc định xử lý ngoại lệ.

# Checked and Unchecked Exceptions (1 of 5)

- Exceptions sẽ được chia thành loại như sau:
  - unchecked
  - checked.
- *Các loại ngoại lệ Unchecked thừa kế từ lớp `Error` hoặc `RuntimeException`.*
- Nhắc lại: Các ngoại lệ bắt nguồn từ lớp `Error` được ném ra khi một lỗi nghiêm trọng xảy ra và không nên được xử lý.
- `RuntimeException` như là một lớp cha (lớp nguồn gốc) cho tất cả các `Exception` được sinh ra bởi lỗi lập trình.

## Checked and Unchecked Exceptions (2 of 5)

- Trong hầu hết các trường hợp thì các exception loại Unchecked không nên được xử lý nhưng có thể tránh chúng thông qua việc viết chương trình đúng cách.
- Các ngoại lệ không thừa kế từ lớp `Error` hoặc lớp `RuntimeException` là những *exceptions thuộc loại checked*.

## Checked and Unchecked Exceptions (3 of 5)

- Nếu đoạn chương trình trong một phương thức gây ra exception, thì phương thức:
  - Phải xử lý ngoại lệ, hoặc
  - Phần header của phương thức phải có sử dụng lệnh throw
- Lệnh throws sẽ thông báo cho trình biên dịch những ngoại lệ nào có thể được ném ra từ một phương thức.

# Checked and Unchecked Exceptions (4 of 5)

```
// Phương thức này sẽ không biên dịch
public void displayFile(String name)
{
    // Open the file.
    File file = new File(name);
    Scanner inputFile = new Scanner(file);
    // Read and display the file's contents.
    while (inputFile.hasNext())
    {
        System.out.println(inputFile.nextLine());
    }
    // Close the file.
    inputFile.close();
}
```

## Checked and Unchecked Exceptions (5 of 5)

- Đoạn chương trình trong phương thức này sẽ có khả năng phát sinh Exception loại checked.
- Từ khóa throws sẽ được khai báo ở cuối phần header của phương thức. Ví dụ:

```
public void displayFile(String name)  
    throws FileNotFoundException
```

# Throwing Exceptions (1 of 2)

- Khi Exception phát sinh, lập trình viên có thể xử lý và ném ra (throws) standard Java exceptions hoặc ném ra ra một Exception đã tùy chỉnh.
- Lệnh `throw` được sử dụng để ném ra các Exception đã tùy chỉnh.

`throw new ExceptionType(MessageString);`

- Câu lệnh `throw` khiến một đối tượng ngoại lệ được tạo và ném.

## Throwing Exceptions (2 of 2)

- Đối số `MessageString` chứa thông báo lỗi tùy chỉnh có thể được truy xuất từ phương thức `getMessage` của đối tượng lớp `Exception`.
- Nếu bạn không truyền đối số (một thông báo) cho phương thức khởi tạo, ngoại lệ sẽ có một thông báo rỗng.

```
throw new Exception("Out of fuel");
```

–*Lưu ý: Tránh nhầm lẫn giữa `throw` với `throws`.*

- Ví dụ: [DieExceptionDemo.java](#)



# Creating Exception Classes (1 of 2)

- Lập trình viên có thể tạo ra các ngoại lệ của riêng mình bằng cách kế thừa lớp Exception hoặc kế thừa lớp con của lớp Exception.
- Example:
  - [BankAccount.java](#)
  - [NegativeStartingBalance.java](#)
  - [AccountTest.java](#)

## Creating Exception Classes (2 of 2)

- Ví dụ: Một số ví dụ về các trường hợp ngoại lệ có thể ảnh hưởng đến tài khoản ngân hàng:
  - Số dư ban đầu là số 00 được truyền vào phương thức khởi tạo.
  - Lãi suất âm được truyền vào phương thức khởi tạo
  - Một số âm được truyền vào phương thức gửi tiền (deposit).
  - Số tiền được truyền vào phương thức rút tiền(withdrawal) vượt quá số dư tài khoản.
- Lập trình viên sẽ tạo ra các Exception đại diện cho từng điều kiện có thể phát sinh ngoại lệ.

# @exception Tag in Documentation Comments

- Định dạng chung

`@exception` *Mô tả ngoại lệ*

- Các quy tắc

- Thẻ `@exception` trong Java của một phương thức phải xuất hiện sau phần mô tả chung của phương thức.
- Mô tả có thể kéo dài nhiều dòng. Nó kết thúc ở cuối chú thích JavaDoc (ký hiệu `* /`) hoặc ở đầu một thẻ khác.

*Sinh viên tìm hiểu thêm về JavaDoc.*