T3H
IT-Institute

VIỆN CÔNG NGHỆ THÔNG TIN T3H

Đào tạo chuyên sâu - Trải nghiệm thực tế

Lập trình Java

Exceptio

GV Nguyễn Đắc Kiên

Nội dung

I. Định nghĩa
II. Hệ thống Exception Handling trong Java
III. Các kiểu ngoại lệ
IV. Cách xử lý ngoại lệ
V. Cách tự tạo ra Exception

1. Định nghĩa

- Trong Java, Exception (ngoại lệ) là một sự kiện xảy ra trong quá trình thực thi chương trình, làm gián đoạn luồng thực thi bình thường của chương trình.
- Exception thường xuất hiện khi có lỗi xảy ra, chẳng hạn như chia một số cho 0, truy cập vào một phần tử ngoài giới hạn của mảng, hoặc khi tệp cần đọc không tồn tại.
- Xử lý ngoại lệ là quá trình quản lý và phản ứng lại với một số điều kiện có thể xảy ra trong quá trình chương trình chạy. Trong Java ngoại lệ là một đối tượng đại diện cho những điều kiện làm chương trình bị gián đoạn. Khi một trường hợp ngoại lệ xảy ra thì một đối tượng sẽ được tạo ra.

1.2 Ví dụ về Exception

- Ví dụ 1:

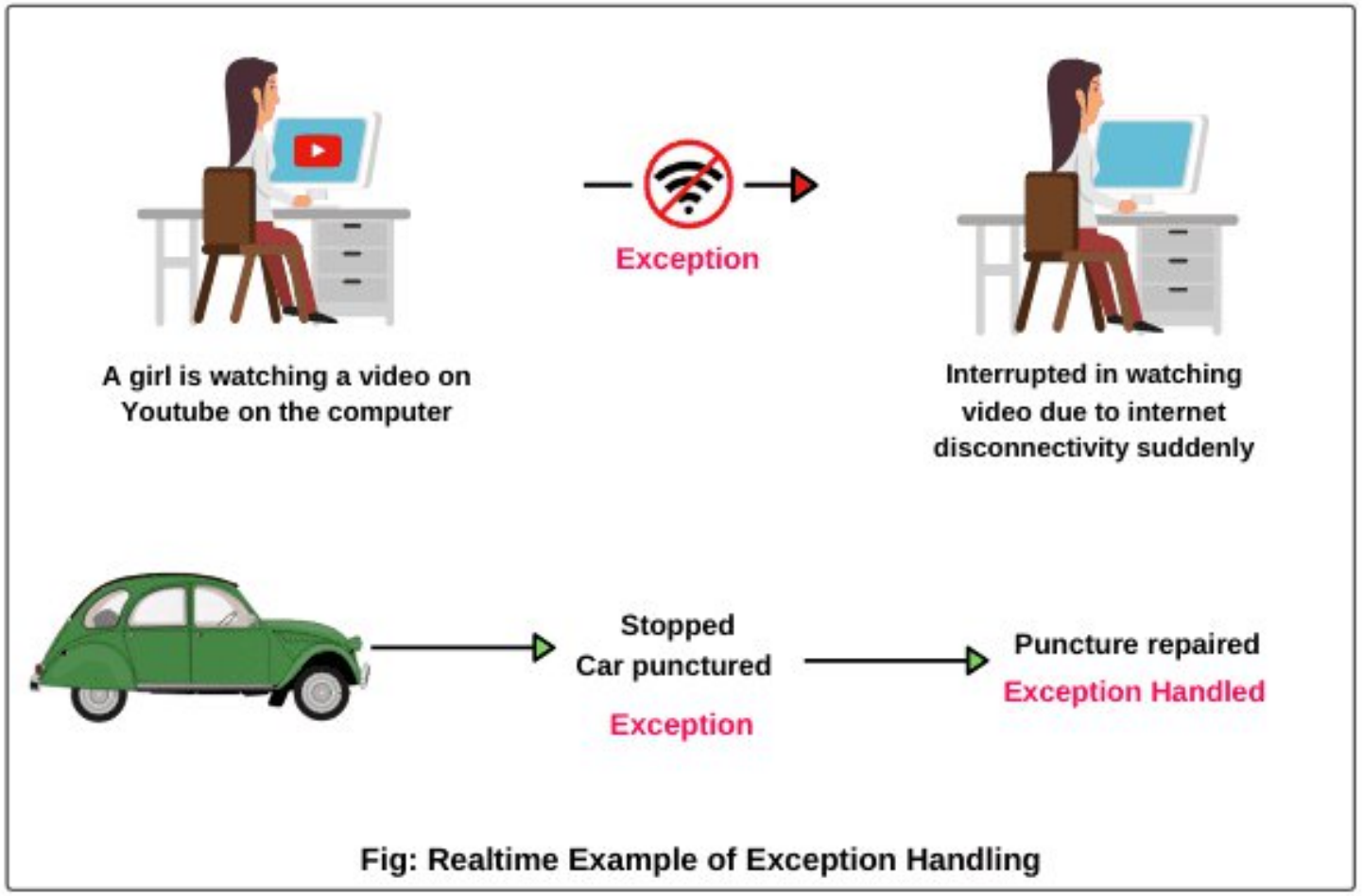


Fig: Realtime Example of Exception Handling

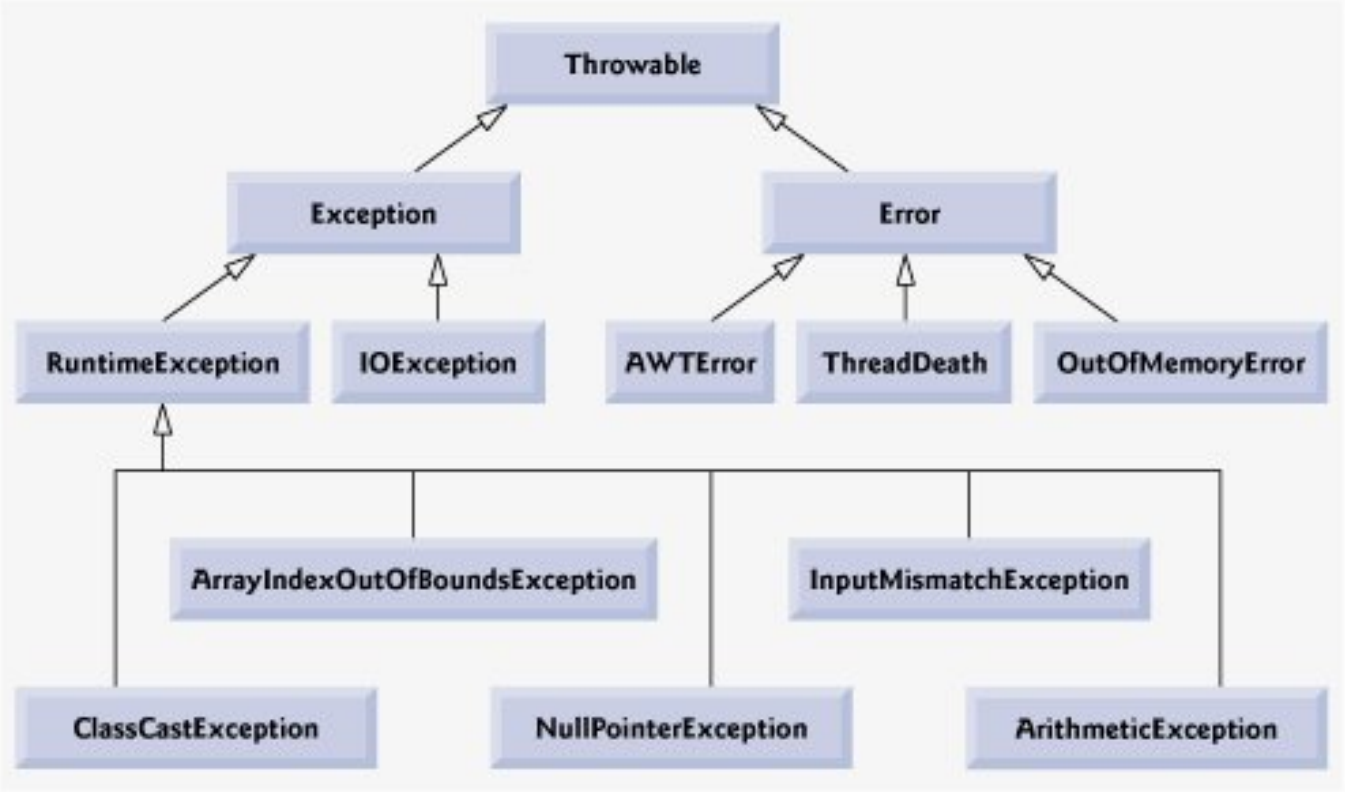
1.2 Ví dụ về Exception

Giải thích ví dụ 1:

- **Khi không có xử lý ngoại lệ:**
  - Giả sử rằng bạn đang xem một video trên Youtube. Đột nhiên tín hiệu internet của bạn mất kết nối hoặc model wifi không hoạt động. Trong trường hợp này bạn không thể tiếp tục xem video trên Youtube nữa. **Sự gián đoạn này chính là exception.**
- **Khi có xử lý ngoại lệ:**
  - Giả sử tại hình phía dưới một người đang đi du lịch bằng ô tô từ Hà Nội đến Đà Nẵng. Sau khi đi được nửa chặng đường, thì lốp xe bị thủng. **Đây là một "ngoại lệ" không mong muốn.**
  - Người chủ xe đã tính đến tường hợp này và đã dự phòng một chiếc lốp để thay thế. Anh ta thay chiếc lốp thủng bằng chiếc lốp mới. Sau khi thay lốp, anh này lại tiếp tục cuộc hành trình còn lại. **Việc thay lốp này còn gọi là xử lí ngoại lệ (exception handling).**

2. Hệ thống Exception Handling trong Java

- Sơ đồ minh họa cấu trúc phân cấp của các loại ngoại lệ và lỗi trong Java



2.1 Throwable

- Đây là lớp cha của tất cả các đối tượng ngoại lệ và lỗi trong Java.
- Throwable có hai lớp con chính:

1. Exception: Các ngoại lệ có thể phục hồi và thường được xử lý trong mã nguồn của bạn.
2. Error: Các lỗi nghiêm trọng mà không thể phục hồi và thường không thể xử lý trong mã nguồn.

2.2 Exception (Ngoại lệ)

Exception là lớp cha của các ngoại lệ có thể xử lý. Các lớp con của Exception bao gồm:

1. RuntimeException: Đây là loại ngoại lệ không kiểm tra (unchecked exceptions). Các lỗi thuộc loại này có thể xảy ra trong quá trình thực thi và không bắt buộc phải xử lý.
   - Các ví dụ trong RuntimeException:
     - ArrayIndexOutOfBoundsException: Lỗi khi truy cập phần tử ngoài phạm vi mảng.
     - ClassCastException: Lỗi khi ép kiểu một đối tượng không hợp lệ.
     - NullPointerException: Lỗi khi cố gắng truy cập vào phương thức của đối tượng null.
     - ArithmeticException: Lỗi toán học, ví dụ như chia cho 0.
2. IOException: Là lớp cha của các ngoại lệ liên quan đến các vấn đề trong việc nhập/xuất (I/O), chẳng hạn như khi đọc hoặc ghi tệp không thành công.

2.3 Error (Lỗi)

Error là các lỗi nghiêm trọng mà bạn không thể khắc phục hoặc xử lý trong mã nguồn của mình. Các lớp con của Error bao gồm:

1. **AWTError:** Liên quan đến lỗi trong AWT (Abstract Window Toolkit), một phần của Java dùng để tạo giao diện người dùng.
2. **ThreadDeath:** Lỗi xảy ra khi một thread trong Java bị dừng đột ngột.
3. **OutOfMemoryError:** Lỗi xảy ra khi JVM không còn đủ bộ nhớ để tiếp tục chương trình.

2. Hệ thống Exception Handling trong Java

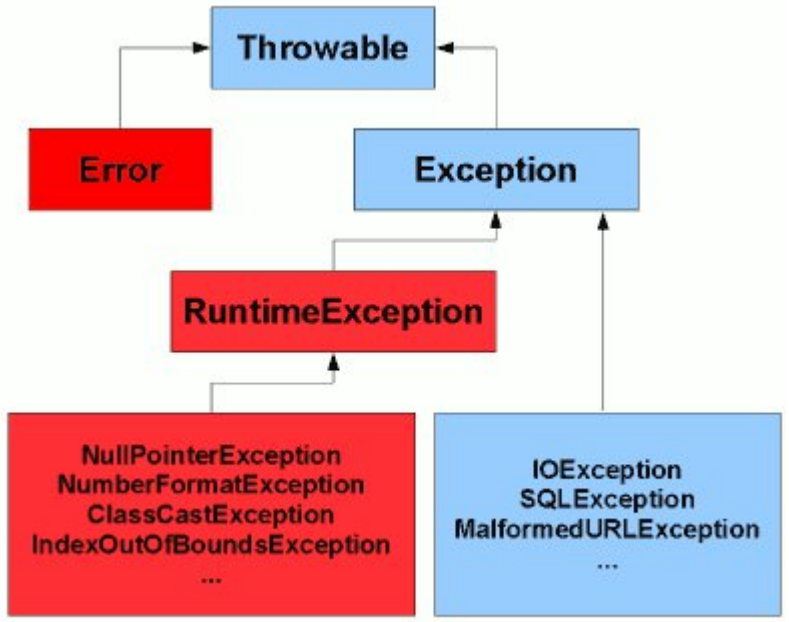
- Tóm tắt các lớp trong sơ đồ:

Lớp	Mô tả
`Throwable`	Lớp cha của tất cả các ngoại lệ và lỗi trong Java.
`Exception`	Các ngoại lệ có thể xử lý, thường gặp trong các tình huống mà lập trình viên có thể xử lý.
`RuntimeException`	Ngoại lệ không bắt buộc phải xử lý. Thường xảy ra do lỗi lập trình.
`IOException`	Ngoại lệ liên quan đến việc nhập/xuất (I/O), ví dụ như đọc và ghi tệp.
`Error`	Lỗi nghiêm trọng không thể phục hồi, thường liên quan đến tài nguyên hệ thống hoặc vấn đề môi trường chạy JVM.
`OutOfMemoryError`	Lỗi khi hệ thống hết bộ nhớ để tiếp tục thực thi chương trình.

3. Phân loại Exception trong Java

Trong Java, Exception được phân thành hai loại chính:

- Checked Exception (Ngoại lệ kiểm tra được)
- Unchecked Exception (Ngoại lệ không kiểm tra)
- Ngoài ra, còn có Error, tuy không phải là Exception nhưng cũng ảnh hưởng đến chương trình.



3.1 Checked Exception (Ngoại lệ kiểm tra được)

- Là những ngoại lệ mà trình biên dịch bắt buộc bạn phải xử lý (dùng try-catch hoặc throws).
- Chủ yếu xảy ra do các yếu tố bên ngoài như lỗi file, lỗi kết nối mạng, lỗi nhập/xuất,...Nếu không xử lý, chương trình sẽ không thể biên dịch được.
- **Checked Exception sảy ra tại thời điểm compile time. Nếu không xử lý sẽ biên dịch code lỗi**

3.1 Checked Exception (Ngoại lệ kiểm tra được)

- Ví du về các trường hợp có thể sảy ra CheckedException

1. IOException: Lỗi trong quá trình nhập/xuất dữ liệu (ví dụ: đọc/ghi file).
2. SQLException: Lỗi khi thao tác với cơ sở dữ liệu (SQL).
3. FileNotFoundException: Lỗi khi cố gắng truy cập vào một tệp không tồn tại.

- **Code ví dụ về 3:**

```
public class CheckedExceptionExample {  new *
    public static void main(String[] args) {  new *
        try {
            File file = new File( pathname: "example.txt");
            /*
            Có thể gây lỗi FileNotFoundException
            khi không tìm thấy file example.txt
             */
            Scanner sc = new Scanner(file);
            while (sc.hasNextLine()) {
                System.out.println(sc.nextLine());
            }
        } catch (FileNotFoundException e) {
            System.out.println("Lỗi: Tệp không tồn tại.");
        }
    }
}
```

3.1 Checked Exception (Ngoại lệ kiểm tra được)

- Ví dụ về ParseException trong Java

```
public class ParseExceptionExample {  new *
    public static void main(String[] args) {  new *
        String dateStr = "2025-03-19"; // Định dạng ngày sai: Năm-tháng-ngày
        SimpleDateFormat sdf = new SimpleDateFormat( pattern: "dd/MM/yyyy"); // Định dạng ngày đúng: ngày/tháng/năm
        try {
            // Cố gắng phân tích chuỗi thành đối tượng Date theo định dạng sai
            Date date = sdf.parse(dateStr); // Lỗi xảy ra tại đây nếu chuỗi không khớp định dạng
            System.out.println("Ngày hợp lệ: " + date);
        } catch (ParseException e) {
            System.out.println("Lỗi phân tích chuỗi ngày: " + e.getMessage());
        }
    }
}
```

3.2 Unchecked Exception (Ngoại lệ không kiểm tra)

- Unchecked Exception (Ngoại lệ không kiểm tra) là các ngoại lệ không yêu cầu lập trình viên phải xử lý hoặc khai báo trong mã nguồn.
- Điều này có nghĩa là bạn không cần phải sử dụng try-catch hoặc throws để xử lý những ngoại lệ này.
- **Unchecked Exception thường liên quan đến lỗi logic hoặc các vấn đề phát sinh trong quá trình thực thi chương trình(Runtime)** mà lập trình viên có thể tránh được bằng cách cải thiện mã nguồn.
- Unchecked Exception kế thừa từ RuntimeException. Tất cả các Unchecked Exception đều kế thừa từ lớp RuntimeException hoặc các lớp con của nó.

3.2 Unchecked Exception (Ngoại lệ không kiểm tra)

- Ví dụ về Unchecked Exception: Lỗi này xảy ra khi cố gắng chia một số cho 0.

```
2
3   public class UncheckedExceptionExample {  new *
4       public static void main(String[] args) {  new *
5           int a = 10;
6           int b = 0;
7
8           // Cố gắng chia cho 0
9           int result = a / b; // Sẽ gây ra ArithmeticException
0           System.out.println("Kết quả: " + result);
1       }
2   }
```

3.2 Unchecked Exception (Ngoại lệ không kiểm tra)

- Ví dụ 2: NullPointerException (Truy cập vào đối tượng null)

```
3  public class NullPointerExample {  new *
4      public static void main(String[] args) {  new *
5          String str = null;
6
7          // Cố gắng gọi phương thức trên đối tượng null
8          System.out.println(str.length()); // Sẽ gây ra NullPointerException
9      }
10 }
```

- Lỗi này xảy ra khi bạn cố gắng gọi phương thức hoặc truy cập trường của đối tượng null.

3.3 Error

- **Không thể xử lý:** Các Error thường xảy ra trong môi trường JVM và không thể được phục hồi hoặc xử lý. Chúng không nên được xử lý bằng try-catch vì chúng thường là những sự cố hệ thống nghiêm trọng.
- **Không nên can thiệp:** Nếu một Error xảy ra, chương trình thường không thể tiếp tục thực thi bình thường, và việc xử lý sẽ không mang lại kết quả tốt.
- **Không phải là ngoại lệ:** Mặc dù cả Exception và Error đều kế thừa từ lớp Throwable, nhưng Error không phải là một ngoại lệ mà người lập trình có thể kiểm soát hoặc sửa chữa.

3.3 Error

Các loại Error

1. **OutOfMemoryError:** Xảy ra khi JVM không còn đủ bộ nhớ để cấp phát cho đối tượng mới. Đây là một lỗi nghiêm trọng và không thể phục hồi trong hầu hết các trường hợp.
2. **StackOverflowError:** Xảy ra khi ngăn xếp (stack) của chương trình bị tràn, thường do đệ quy vô hạn hoặc quá nhiều cuộc gọi hàm.
3. **VirtualMachineError:** Xảy ra khi có lỗi trong môi trường chạy JVM, như việc JVM gặp sự cố hoặc không thể tiếp tục.
4. **ThreadDeath:** Xảy ra khi một thread bị giết hoặc dừng đột ngột. Đây là lỗi nghiêm trọng trong quản lý các thread.

3.3 Error

- Code ví dụ về StackOverflowError: Lỗi này thường xảy ra do đệ quy vô hạn, khi một phương thức gọi chính nó quá nhiều lần mà không có điều kiện dừng, khiến ngăn xếp (stack) của JVM bị tràn.

```
2
3   public class StackOverflowErrorExample {  new *
4       public static void recursiveMethod() {  2 usages  new *
5           recursiveMethod(); // Gọi đệ quy vô hạn
6       }
7       public static void main(String[] args) {  new *
8           try {
9               recursiveMethod(); // Gây ra StackOverflowError
10          }catch (Exception e){
11              System.out.println("Lỗi");
12          }
13      }
14  }
```

4. Các từ khóa xử lý ngoại lệ

- Trong Java, để xử lý các ngoại lệ (exceptions), có thể sử dụng một số từ khóa quan trọng.
- Dưới đây là các từ khóa chủ yếu trong cơ chế xử lý ngoại lệ:

1. Try – catch
2. Finally
3. Throw
4. Throws

4.1 Try – catch

- **Mục đích:** Dùng để bao bọc các đoạn mã có thể gây ra ngoại lệ. Nếu một ngoại lệ xảy ra trong khối try, chương trình sẽ chuyển đến khối catch để xử lý.
- Cú pháp:

```
try {
    // Code có thể gây ra ngoại lệ
} catch ( Loại Exception) {
        // Code xử lý khi ngoại lệ xảy ra
}
```

```java
public class Demo1 {  new *
    public static void main(String[] args) {  new *
        try {
            int result = 10 / 0; // Lỗi chia cho 0
        } catch (ArithmeticException e) {
            System.out.println("Lỗi: Không thể chia cho 0!");
        }
    }
}
```

4.2 Finally

- **Mục đích:** Khối finally chứa mã cần thực thi dù có xảy ra ngoại lệ hay không. Thường được dùng để dọn dẹp tài nguyên như đóng tệp, đóng kết nối cơ sở dữ liệu, v.v.
- Cú pháp:

```
try {
    // Code có thể gây ra ngoại lệ
} catch (ExceptionType e) {
    // Xử lý ngoại lệ
} finally {
    // Mã này luôn được thực thi
}
```

```java
public class Demo2 {  new *
    public static void main(String[] args) {  new *
        try {
            int result = 10 / 2;
            System.out.println("Kết quả: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Lỗi toán học!");
        } finally {
            System.out.println("Khối finally luôn được thực thi.");
        }
    }
}
```

4.3. Throw

- **Mục đích:** Dùng để ném ngoại lệ trong phương thức hoặc chương trình khi bạn muốn báo hiệu rằng một điều gì đó sai và cần được xử lý.
- Cú pháp:
  - throw new ExceptionType("Message");
  - **ExceptionType:** tên class đại diện cho lỗi ví dụ RuntimeException, NullPointerException
  - **Message:** Thông điệp tự dev custome để bắn ra khi sảy ra lỗi

```
public class Demo3 {  new *
    public static void checkAge(int age) {  1 usage  new *
        if (age < 18) {
            throw new IllegalArgumentException("Tuổi phải lớn hơn hoặc bằng 18");
        } else {
            System.out.println("Tuổi hợp lệ");
        }
    }
    public static void main(String[] args) {  new *
        checkAge(16); // Sẽ ném ra IllegalArgumentException
    }
}
```

4.4 throws

- **Mục đích:** Dùng trong khai báo phương thức để khai báo các ngoại lệ có thể được ném ra từ phương thức đó. Phương thức này không xử lý ngoại lệ mà để cho phương thức gọi xử lý.
- Cú pháp:

```
public void methodName() throws ExceptionType {
    // Code có thể ném ra ngoại lệ
}
```

ExceptionType: tên class đại diện cho lỗi ví dụ RuntimeException, NullPointerException

4.4 throws

- Code ví dụ:

```java
public class ThrowsExample {  new *
    static void checkAge(int age) throws IllegalArgumentException {  1 usage  new *
        if (age < 18) {
            throw new IllegalArgumentException("Tuổi phải lớn hơn hoặc bằng 18");
        } else {
            System.out.println("Tuổi hợp lệ");
        }
    }
    public static void main(String[] args) {  new *
        try {
            checkAge(16); // Phương thức checkAge ném ra ngoại lệ
        } catch (IllegalArgumentException e) {
            System.out.println("Lỗi: " + e.getMessage());
        }
    }
}
```

```
D:\T3h\LJava2502\core\core\out\produc
Lỗi: Tuổi phải lớn hơn hoặc bằng 18
```

5. Cách tự tạo ra Exception

- Trong Java, để tạo ra một exception tùy chỉnh, có thể định nghĩa một lớp mới kế thừa từ lớp Exception hoặc RuntimeException.
- Dưới đây là cách bạn có thể tạo và sử dụng exception trong Java.

❑ Bước 1: Tạo lớp Exception tùy chỉnh

❑ Bước 2: Sử dụng exception tùy chỉnh trong mã của chương trình

5. Cách tự tạo ra Exception

- Bước 1: Tạo lớp Exception tùy chỉnh
  - **Kế thừa lớp Exception:** Nếu muốn tạo một checked exception (lỗi mà yêu cầu phải xử lý bằng try-catch hoặc khai báo trong phương thức với throws), bạn kế thừa từ lớp Exception.

```
// Tạo lớp exception tùy chỉnh
public class MyCheckedException extends Exception {
    // Constructor
    public MyCheckedException(String message) {
        super(message);
    }
}
```

  - **Kế thừa lớp RuntimeException:** Nếu muốn tạo một unchecked exception (lỗi không yêu cầu phải xử lý), bạn kế thừa từ RuntimeException.

```
// Tạo lớp exception tùy chỉnh
public class MyUncheckedException extends RuntimeException {
    // Constructor
    public MyUncheckedException(String message) {
        super(message);
    }
}
```

5. Cách tự tạo ra Exception

- Bước 2: Sử dụng exception tùy chỉnh trong mã của bạn
- **Với checked exception:** Cần phải xử lý nó bằng try-catch hoặc khai báo phương thức sử dụng throws.

```
public class TestChecked {  new *
    public static void main(String[] args) {  new *
        try {
            // Ném exception tùy chỉnh
            throw new MyCheckedException("Đây là checked exception!");
        } catch (MyCheckedException e) {
            System.out.println("Lỗi: " + e.getMessage());
        }
    }
}
```

5. Cách tự tạo ra Exception

- Bước 2: Sử dụng exception tùy chỉnh trong mã của bạn
- **Với unchecked exception:** Không cần khai báo hoặc xử lý bằng try-catch (mặc dù vẫn có thể nếu muốn).

```
public class TestUnchecked {  new *
    public static void main(String[] args) {  new *
        // Ném exception tùy chỉnh
        throw new MyUncheckedException("Đây là unchecked exception!");
    }
}
```