Handle Exception

1. Phân biệt **throw** và **throws**

Đặc điểm	throw	throws
Mục đích	Dùng để ném một ngoại lệ (exception) cụ thể	Dùng để khai báo rằng một phương thức có thể ném ra một ngoại lệ
Vị trí	Nằm bên trong phần thân của phương thức hoặc khối lệnh (try – catch)	Nằm sau tên phương thức, trong phần khai báo
Cú pháp	Theo sau là một đối tượng ngoại lệ (vd: throw new IOException();).	Theo sau là tên của một hoặc nhiều lớp ngoại lệ (vd: throws IOException, SQLException).
Sử dụng	Dùng để tạo và ném một ngoại lệ một cách tường minh	Dùng để bắt buộc người gọi phương thức phải xử lý ngoại lệ này (bằng try- catch hoặc throws tiếp).

2. Thế nào là **checked** và **unchecked** exception

- Checked exceptions: là những ngoại lệ mà trình biên dịch bắt buộc phải xử lý tại thời điểm biên dịch. Nếu không, chương trình sẽ không thể biên dịch được. Chúng thường đại diện cho các lỗi bên ngoài, nằm ngoài tầm kiểm soát của lập trình viên.

Đặc điểm:

- **Kiểm tra lúc biên dịch:** Trình biên dịch sẽ kiểm tra xem đã xử lý ngoại lệ này chưa.
- **Bắt buộc xử lý:** Phải sử dụng try-catch để bắt ngoại lệ hoặc dùng từ khóa throws để khai báo trong chữ ký của phương thức.
- **Thường có thể phục hồi:** Chúng thường chỉ ra một vấn đề mà chương trình có thể khôi phục được.

Ví dụ:

- IOException: Xảy ra khi có lỗi trong quá trình thao tác I/O (ví dụ: đọc/ghi file).
- FileNotFoundException: Xảy ra khi cố gắng truy cập một file không tồn tại.
- SQLException: Xảy ra khi có lỗi trong quá trình kết nối cơ sở dữ liệu.

- Unchecked exceptions: là những ngoại lệ mà trình biên dịch không bắt buộc phải xử lý. Chúng thường xảy ra do các lỗi logic trong code và có thể tránh được nếu viết code cẩn thận hơn.

Đặc điểm:

- **Kiểm tra lúc chạy:** Chúng chỉ được phát hiện khi chương trình đang chạy.
- **Không bắt buộc xử lý:** Không cần phải sử dụng try-catch hoặc throws (mặc dù vẫn có thể làm vậy).
- **Thường không thể phục hồi:** Chúng thường chỉ ra một lỗi nghiêm trọng trong chương trình mà không thể dễ dàng khôi phục.

Ví dụ:

- NullPointerException: Xảy ra khi cố gắng truy cập một đối tượng null.
- ArrayIndexOutOfBoundsException: Xảy ra khi truy cập một chỉ mục không hợp lệ của mảng.
- ArithmeticException: Xảy ra khi có lỗi toán học, ví dụ: chia cho 0.

3. **try catch** , **try with resource** khác nhau như thế nào ?

- **try-catch:** là cách truyền thống để bắt và xử lý ngoại lệ. Nó yêu phải tự quản lý các tài nguyên (resources) như luồng dữ liệu (streams), kết nối cơ sở dữ liệu (database connections),... Điều này có nghĩa là phải đảm bảo rằng các tài nguyên này được đóng lại một cách tường minh, ngay cả khi có lỗi xảy ra.

Cú pháp:

```
try {
        // Khối lệnh có thể ném ngoại lệ
```

```
} catch (ExceptionType e) {
        // Khối lệnh xử lý ngoại lệ
} finally {
        // Khối lệnh luôn được thực thi, dùng để đóng tài nguyên
 }
```

- try-with-resources: được giới thiệu từ Java 7, là một cải tiến nhằm tự động đóng các tài nguyên. Nó giúp mã nguồn gọn gàng hơn và an toàn hơn, vì không cần phải viết khối finally để đóng tài nguyên một cách thủ công. Để sử dụng try-with-resources, tài nguyên phải implement **java.lang.AutoCloseable**.

Cú pháp:

```
try (ResourceType resource = new ResourceType()) {
   // Khối lệnh sử dụng tài nguyên
} catch (ExceptionType e) {
   // Khối lệnh xử lý ngoại lệ
}
```

Ưu điểm:

- **Tự động đóng:** Các tài nguyên sẽ tự động được đóng lại khi khối try kết thúc, bất kể có ngoại lệ xảy ra hay không.
- **Mã nguồn sạch hơn:** Không cần khối finally phức tạp, giúp code dễ đọc và dễ bảo trì hơn.
- **An toàn hơn:** Giảm thiểu nguy cơ rò rỉ tài nguyên.

4. Làm thế nào để tạo được 1 custom exception ?

Các bước tạo Custom Exception

B1. Chọn loại ngoại lệ

Cần quyết định xem custom exception của mình sẽ là **checked** hay **unchecked**.

- **Nếu muốn là checked exception:** Kế thừa từ lớp Exception. Sẽ bị bắt buộc phải xử lý ngoại lệ này (bằng try-catch hoặc throws). Thường dùng cho các trường hợp có thể phục hồi, như lỗi I/O.
- **Nếu muốn là unchecked exception:** Kế thừa từ lớp RuntimeException. Không bị bắt buộc phải xử lý ngoại lệ này. Thường dùng cho các lỗi logic, không thể phục hồi, như lỗi đầu vào không hợp lệ.

B2. Xây dựng lớp ngoại lệ

Tạo một lớp mới và thêm ít nhất một constructor để có thể truyền thông điệp lỗi.

B3. Sử dụng custom exception

Sử dụng từ khóa throw để ném ngoại lệ trong phương thức khi điều kiện lỗi xảy ra.

Ví dụ:

```
public class InvalidNumberException extends Exception {
    public InvalidNumberException(String message) {
        super(message);
    }
}

public class NumberProcessor {
    public void processPositiveNumber(int number) throws InvalidNumberException {
        if (number < 0) {
            throw new InvalidNumberException("Number cannot be negative: " + number);
        }
```

```
    }

    public static void main(String[] args) {
        NumberProcessor processor = new NumberProcessor();
        try {
            processor.processPositiveNumber(5);
            processor.processPositiveNumber(-3);
        } catch (InvalidNumberException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```