< 예외처리 >

1. 에러

➔ 컴파일 에러 : 소스상의 문법 에러 → 에러가 발생한 소스 구문을 수정하여 해결

➔ 런타임 에러 : 프로그램 실행 시 발생하는 에러 (입력 값이 틀렸거나, 배열의 인덱스 범위를 벗어났거나, 계산식의 오류, 0으로 나누기 하거나 등등..)

→ 소스상에서 에러에 대한 처리를 해서 해결 (if문 주로 사용)

➔ 시스템 에러 : 컴퓨터 오작동으로 인한 에러 → 소스 구문으로는 해결 못함

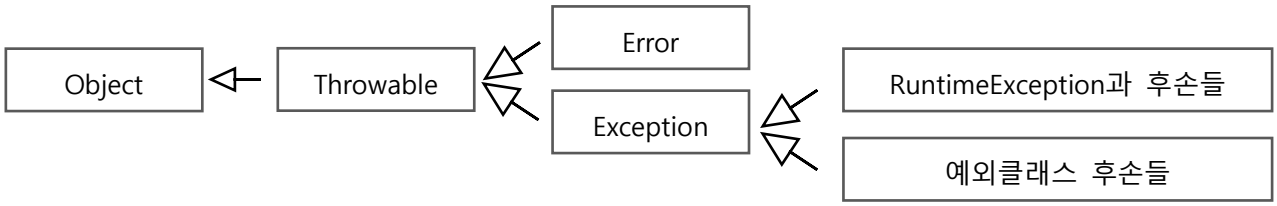
- 소스로 해결 못하는 에러 => 에러(Error)
- 소스로 해결 가능한 에러 => 예외(Exception)

→ 이러한 예외상황(예측 가능한 에러)의 구문을 처리하는 방법 => 예외처리

* 자바가 제공하는 클래스의 메소드 사용 시, 사용할 메소드의 헤드 뒤쪽에 throws XXXException이 명시된 경우, 그냥 사용 시 에러 발생함

→ 해당 메소드 사용 시 반드시 뒤에 명시된 예외 클래스를 처리해야 사용 가능

<예외클래스의 상속관계>



- **자바에서 사용 가능한 예외 클래스의 최상위는 Exception 클래스임**

- **checked Exception : 소스 코드 상에서 반드시 개발자가 처리해야 되는 예외클래스들**

  → RuntimeException이 아닌 예외클래스들

- **unchecked Exception : 개발자가 소스 코드상에서 다룰 필요가 없는 예외클래스들**

  → RuntimeException과 후손 클래스들

2. 예외처리 방법

1) 떠 넘기는 방법 (throws 사용)

접근제어자 반환자료형 메소드명([자료형 매개변수]) throws 처리할예외클래스명 {

// 예외를 처리해야 하는 구문

}

➔ 여러 단계로 떠 넘길 수 있음

ex)

```
public void testA() throws 처리할예외클래스명 {
        testB();
}

public void testB() throws 처리할예외클래스명 {
        testC();
}

public void testC() throws 처리할예외클래스명 {
        // 예외를 처리해야 하는 구문 작성
}
```

2) 예외처리 구문을 사용해서 직접 처리하는 방법 (try ~ catch 구문 사용)

try {

// 예외가 발생할 구문(반드시 예외를 처리해야 하는 구문)

} catch(처리할예외클래스명 레퍼런스) {

// 발생 한 예외에 대한 처리 구문 작성

}

*** **catch 구문을 여러 번 사용할 경우(각각의 예외에 각각의 처리를 적용 시키기 위해)**

상속관계를 고려해서 최하위 후손 예외클래스가 가장 먼저 위에 제시되어야 함

→ 최상위 예외클래스가 가장 밑에(나중에) 제시되어야 함.

→ 최상위 예외클래스가 가장 먼저 제시되면, 부모가 후손의 주소를 다 받게 되므로, 아래에 명시된 후손 예외 클래스가 작동되지 못하기 때문임

*** **finally { }** : 실행 도중 해당 Exception이 발생하든, 안 하든 반드시 실행해야 하는 구문은 finally 블록에 작성하면 됨. (보통 close()를 위해 사용)

→ try ~ catch 블록 맨 마지막에 추가함.

ex)

```
try {
        // 예외가 발생할 구문(반드시 예외를 처리해야 하는 구문)
} catch(처리할예외클래스명 레퍼런스) {
        // 발생 한 예외에 대한 처리 구문 작성
} finally {
        // 예외 발생 하던 안하던 간에 반드시 실행되어야 하는 구문
}
```

*** **try ~ with ~ resource 구문 (자바 7에서부터 추가된 기능임)**

→ finally에 작성되었던 close()처리를 생략함 == 자동으로 close되게 하는 문장임

ex)

```
try(반드시 close 처리 해야 하는 객체에 대한 생성 구문) {
        // 예외가 발생할 구문(반드시 예외를 처리해야 하는 구문)
} catch(처리할예외클래스명 레퍼런스) {
        // 발생 한 예외에 대한 처리 구문 작성
} /* finally {
        // 예외 발생 하던 안 하던 간에 반드시 실행되어야 하는 구문
} */
```

3. 사용자 정의 예외클래스

➔ Exception 클래스를 상속받아야 함

➔ 생성자 작성 시 에러 메세지를 부모 생성자로 넘겨주면 됨.

➔ 개발자가 지정한 상황에서 만든 예외클래스가 작동되게 하려면

```
if(지정한 조건 제시){
        // 예외발생 시킴
}
```

➔ 예외를 직접 발생 시킬 때는, throw 키워드 사용함.

```
throw new MyException();
```

➔ 예외를 발생 시킨 구문을 포함하고 있는 메소드 헤더에

```
throws 예외클래스명을 표기함
```

ex)

```
public 반환자료형 메소드명(....) throws 예외클래스명 {
    if (조건) {
        // throw new 사용자정의예외클래스생성자(...);
    }
}
```