Exception

Exception

Exception

프로그래밍 오류의 종류

1. 컴파일 에러(compile-time error) : 소스상의 문법 에러

2. 런타임 에러(runtime error) : 입력 값이 틀렸거나, 배열의 인덱스범위를 벗어났거나, 계산식의 오류 등으로 인해 발생하는 에러

3. 논리 에러(logical error) : 문법상 문제가 없고, 런타임 에러도 발생하지 않지만, 개발자의 의도대로 작동하지 않는 경우

4. 시스템 에러(system error) : 컴퓨터 오작동으로 인한 에러
   → 소스코드로 해결이 불가능

Exception

Exception

- 예외 라는 뜻을 가지고 있으며, 예외는 예기치 못한 상황
- 프로그래밍을 하다 보면 수 많은 오류상황을 직면
- 자바에서 예외(Exception)란 프로그램을 만든 개발자가 예상한 정상적인 처리에서 벗어나는 경우에 이를 처리하기 위한 방법
- 예측 가능한 에러를 처리하는 것

예외처리의 목적

- 프로그램의 비정상적인 종료를 막고, 정상적인 실행상태를 유지하기 위함
- 어떻게 → 예외 상황이 발생된 경우 어떻게 처리할 지에 대한 로직을 구현

Exception

Exception 처리 구문

try ~ catch 구문
```
try{
    에러가 발생 할 가능성이 있는 소스코드 작성
}catch(Exception e){
    try블록의 소스코드를 실행하다 에러가 발생하면 즉시 중단하고 catch구문이 실행
}
```

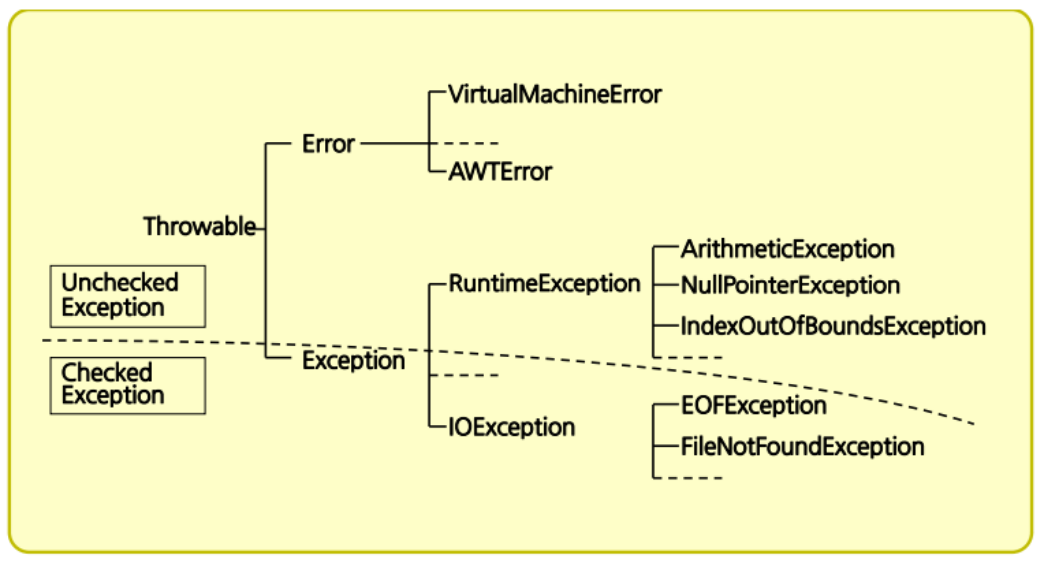
try ~ catch ~ finally 구문
```
try{
    에러가 발생 할 가능성이 있는 소스코드 작성
}catch(Exception e){
    try블록의 소스코드를 실행하다 에러가 발생하면 즉시 중단하고 catch구문이 실행
}finally{
    정상 실행이 되던지, 예외가 발생하던지 무조건 실행해야 하는 코드
}
```

※ 일반적으로 코드에서 사용한 자원을 반납하는 코드를 finally에 작성

Exception

예외의 종류

1. Throwable : 예외의 가장 큰 조상으로 실제 직접 구현하지 않음
2. Error : 개발자가 소스코드로 해결할 수 없는 에러(시스템 에러)
3. Exception : 개발자가 소스코드로 해결할 수 있는 에러
4. Runtime Exception : 어플리케이션이 동작하면서 발생하는 에러



Exception

Checked Exception, Unchecked Exception

- Checked Exception은 예외처리를 강제화 해야 하는 것이고, Unchecked Exception은 예외처리를 강제화 하지 않음
- Checked Exception은 소스코드에서 개발자가 반드시 처리 해야 하며, 처리하지 않는 경우 Eclipse로 컴파일 시 에러가 발생
- Unchecked Exception은 개발자가 소스코드에서 처리하지 않아도 컴파일에 아무 문제가 없음

Exception

RuntimeException 클래스

- RuntimeException은 Exception의 자식이며, Unchecked Exception임

- 주로 개발자의 부주의로 인한 bug가 많기 때문에 Exception처리 보다는 코드를 수정해야 하는 경우가 많음
  → 예외처리를 강제화 하지 않음

Exception

RuntimeException 클래스

1. ArithmethicException
   → 나누기 연산 시 분모가 0인 경우 발생
2. NullPointerException
   → Null인 레퍼런스의 변수나 메소드 참조 시도 시 발생
3. NegativeArraySizeException
   → 배열의 크기를 음수 또는 0으로 지정한 경우 발생
4. ArrayIndexOutOfBountException
   → 배열의 index범위를 넘어서 참조하는 경우 발생
5. ClasscastException
   → cast 연산자 사용시 타입 오류 일때 발생

Exception

Exception 확인

API Document에서 해당 클래스에 대한 생성자나 메소드를 검색하면, 그 메소드가 어떠한 Exception을 발생시킬 가능성이 있는지 확인이 가능하며, 해당 메소드를 사용하려면 반드시 명시된 예외 클래스를 처리해야함

Constructor Detail

FileInputStream

```
public FileInputStream(String name)
                throws FileNotFoundException
```

Creates a `FileInputStream` by opening a connection to an actual file, the file named by the path `name` in the file system. A new `FileDescriptor` object is created to represent this file connection.

First, if there is a security manager, its `checkRead` method is called with the `name` argument as its argument.

If the named file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading then a `FileNotFoundException` is thrown.

Parameters:
`name` - the system-dependent file name.

Throws:
`FileNotFoundException` - if the file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading.

`SecurityException` - if a security manager exists and its `checkRead` method denies read access to the file.

See Also:
`SecurityManager.checkRead(java.lang.String)`

FileInputStream

```
public FileInputStream(File file)
                throws FileNotFoundException
```

Creates a `FileInputStream` by opening a connection to an actual file, the file named by the `File` object `file` in the file system. A new `FileDescriptor` object is created to represent this file connection.

First, if there is a security manager, its `checkRead` method is called with the path represented by the `file` argument as its argument.

If the named file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading then a `FileNotFoundException` is thrown.

Parameters:
`file` - the file to be opened for reading.

Throws:

Exception

예외의 처리

1. try ~ catch 문을 사용하여 발생한 곳에서 직접 처리

2. throws를 사용하여 예외를 던질 수 있음
   → 예외가 발생하는 곳에서 직접 처리하는 것이 아니라 해당 메소드를 호출한 곳에 예외에 대한 처리를 위임하는 방식
   → throws는 여러번 사용이 가능하지만, 결국 한곳에서는 예외에 대한 처리를 해야함

```
public void test() throws FileNotFoundException{
    new FileReader("test.txt");
}
```

→ FileReader 생성자는 반드시 예외를 처리해야하는 해당 소스코드가 있는 곳에 예외를 처리하지 않고 test() 메소드를 호출하는 곳에다가 예외에 대한 처리를 위임함

Exception

사용자 정의 예외

- 표준 예외 클래스로 많은 예외상황을 표현할 수 있으나 그렇지 않은 경우도 발생 가능하며 그 경우 사용자 정의 예외를 만들어서 사용이 가능
- 사용자 정의 예외를 만들 때는 Checked Exception인지 Unchecked Exception인지 결정해야함
- Checked Exception은 Exception 클래스를 상속하여 작성
- Unchecked Exception은 RuntimeException 클래스를 상속하여 작성