7주차: 예외 처리

https://github.com/whiteship/live-study/issues/9

예외... 처음에 예외라고 들으면 `Process finished with exit code` 가 먼저 떠오르면서 강제 종료 되는 거라고 생각이 든다. 예외는 에러를 예외 처리 해줘서 제대로 실행될 수 있게 하면 걔가 예외가 되는 거 아닌가? 라는 생각을 했는데 막상 찾아보니 잘못 알고 있었다.

- **Exception과 Error의 차이는?**

에러 : 컴퓨터 하드웨어의 오작동 혹은 고장으로 인해 응용프로그램 실행 오류가 발생하는 것.

JVM 실행에 문제가 있다는 것이므로 JVM 위에서 실행되는 프로그램을 아무리 잘 만들어도 결국 에러가 발생하면 실행 불능 상태가 됨.

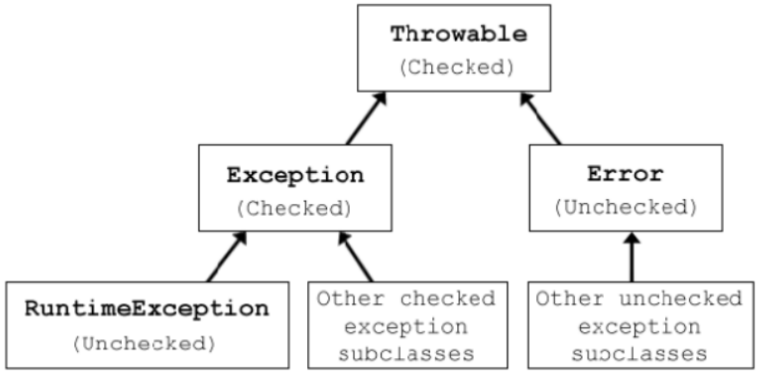
예외 : 사용자의 잘못된 조작 또는 개발자의 잘못된 코딩으로 인해 발생하는 프로그램 오류.

단, 예외는 에러와 달리 예외처리(Exception Handling)을 통해 프로그램 종료하지 않고 정상 실행 상태가 유지되도록 할 수 있다.

즉, 개발자가 프로그램 내에서 처리할 수 있는지 여부에 따라 에러와 예외로 나뉜다.

여기까지만 보면 그럼 이제까지 만난 강제 프로그램 종료는 모두 예외였던 건가? 라는 생각이 들어 예외에 대해 좀 더 알아보았다.

- **자바가 제공하는 예외 계층 구조**



이미지 출처: https://jithub.tistory.com/278

에러와 예외의 계층 구조를 쭉 올라가다보면 **Throwable 클래스**를 확인할 수 있다.

Throwable 클래스는 예외 처리를 할 수 있는 최상위 클래스이다.

java.lang

Class Throwable

java.lang.Object
    java.lang.Throwable

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

Error, Exception

오라클 공식 문서에서

보통 우리가 Exception 으로 예외처리를 하는 이 java.lang.Exception 클래스가 java.lang.Throwable 클래스를 상속하고 있음을 확인할 수 있다.

문서에서 해당 클래스의 메소드들도 볼 수 있는데, 그 중 익숙한 친구들이 보인다.

String	getMessage() Returns the detail message string of this throwable.
void	printStackTrace() Prints this throwable and its backtrace to the standard error stream.
String	toString() Returns a short description of this throwable.

Throwable 클래스의 메소드에서 우리는 getMessage(), toString(), printStackTrace() 등을 통해 예외 내용을 출력할 수 있다.

- getMessage() : 에러의 원인을 담고 있는 문자열 반환.
- toString() : 에러의 Exception 내용과 원인 출력.
- printStackTrace() : 에러의 발생한 위치와 호출된 메소드의 정보를 출력.

다음은 Throwable 클래스의 하위 클래스인 **Error** 클래스와 **Exception** 클래스를 살펴보자.

java.lang

Class Error

java.lang.Object
    java.lang.Throwable
        java.lang.Error

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

AnnotationFormatError, AssertionError, AWTError, CoderMalfunctionError, FactoryConfigurationError, FactoryConfigurationError, IOError, LinkageError, ServiceConfigurationError, ThreadDeath, TransformerFactoryConfigurationError, VirtualMachineError

Error 클래스의 서브클래스를 확인해보면 에러의 의미를 좀 더 확실하게 알 수 있다.

서브클래스 중, VirtualMachineError 는 **JVM에 심각한 오류**가 발생한 것이고 IOError는 입출력 관련해서 **코드 수준 복구가 불가능한 오류**가 발생한 상황에서 발생한다.

이를 보면 알 수 있듯 사용자 혹은 개발자가 해결할 수 있는 문제가 아닌 것들을 에러로 분류한다.

그에 비해 예외 클래스의 서브클래스를 확인해보면,

java.lang

Class Exception

java.lang.Object
    java.lang.Throwable
        java.lang.Exception

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

AclNotFoundException, ActivationException, AlreadyBoundException, ApplicationException, AWTException, BackingStoreException, BadAttributeValueExpException, BadBinaryOpValueExpException, BadLocationException, BadStringOperationException, BrokenBarrierException, CertificateException, CloneNotSupportedException, DataFormatException, DatatypeConfigurationException, DestroyFailedException, ExecutionException, ExpandVetoException, FontFormatException, GeneralSecurityException, GSSException, IllegalClassFormatException, InterruptedException, IntrospectionException, InvalidApplicationException, InvalidMidiDataException, InvalidPreferencesFormatException, InvalidTargetObjectTypeException, IOException, JAXBException, JMException, KeySelectorException, LastOwnerException, LineUnavailableException, MarshalException, MidiUnavailableException, MimeTypeParseException, MimeTypeParseException, NamingException, NoninvertibleTransformException, NotBoundException, NotOwnerException, ParseException, ParserConfigurationException, PrinterException, PrintException, PrivilegedActionException, PropertyVetoException, ReflectiveOperationException, RefreshFailedException, RemarshalException, RuntimeException, SAXException, ScriptException, ServerNotActiveException, SOAPException, SQLException, TimeoutException, TooManyListenersException, TransformerException, TransformException, UnmodifiableClassException, UnsupportedAudioFileException, UnsupportedCallbackException, UnsupportedFlavorException, UnsupportedLookAndFeelException, URIReferenceException, URISyntaxException, UserException, XAException, XMLParseException, XMLSignatureException, XMLStreamException, XPathException

우선 예외 클래스가 굉장히 많다. 사용자와 개발자가 잘못 사용했을 때, 작동하는 오류들이기 때문이다. 왜 그런말이 있지 않은가...

'행복한 가정의 형태는 행복한 가정은 모두 비슷한 이유로 행복하지만 불행한 가정은 저마다의 이유로 불행하다' 라고... 정답 결과값은 하나지만, 잘못될 수 있는 방법은 다양하기 때문에 이렇게 많은게 아닐까... 이 친구들은 다시 잘 실행한다면 정상적으로 작동하기 때문에 그때그때 처리해주어야한다.

이는 Exception 클래스와 다른 클래스와의 차이점으로도 넘어가는데,

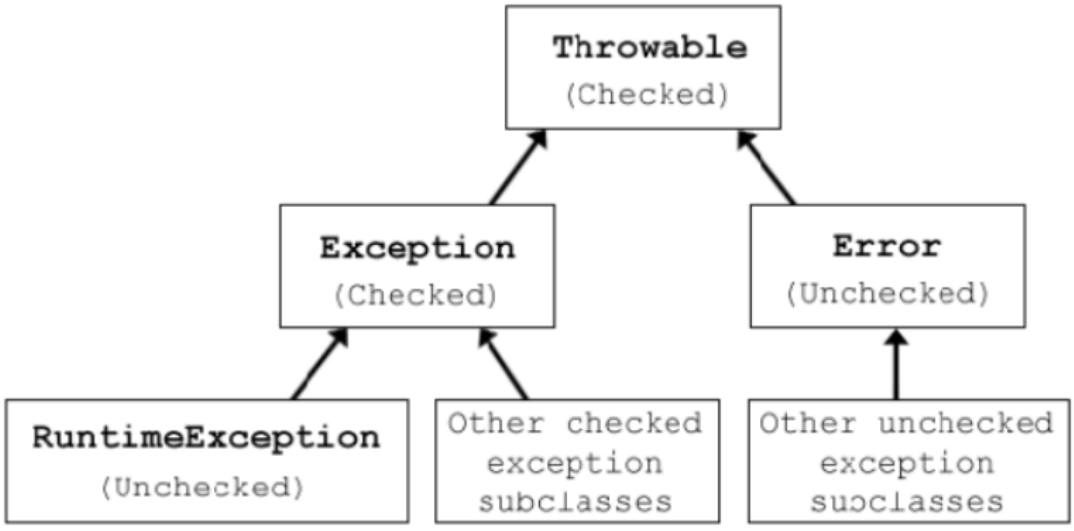
Error 를 상속하거나 RuntimeException을 상속하는 예외의 발생은 코드 작성 과정에서 특별히 무언가를 하지 않아도 되지만, Exception을 상속하는 예외의 발생에 대해서는 try catch 문을 통해 예외를 처리하거나 예외의 처리를 다른 영역으로 넘겨야한다. 그렇지 않으면 **컴파일이 불가능**하다.

다음은 RuntimeException 클래스이다. Exception 클래스를 상속한다. 가장 익숙한 이름들이 많이 나온다.

대표적인(자주 보는) 서브클래스들은 아래와 같다.

1. java.lang.ArithmeticException : 0으로 나누기 등의 산술 오류
2. java.lang.InputMismatchException : 잘못된 입력을 한 경우
3. java.lang.ArrayIndexOutOfBoundsException : 배열의 인덱스를 잘못 참조한 경우
4. java.lang.NegativeArraySizeException : 배열 생성 시 길이를 음수로 지정하는 경우
5. java.lang.NullPointerException : null 객체를 잘못 참조한 경우
6. java.lang.ArrayStoreException : 배열에 적절치 않은 인스턴스를 저장하는 경우

그럼 처음에 나온 구조도는 이해가 되었는데..



다시 이 그림을 살펴보았을 때 드는 의문이 두 가지 있다.

1. 보통 예외를 나눌 때 Error, Exception, RuntimeException 으로 나누었는데 RuntimeException 은 Exception 에 속해있는데 왜 파트를 나눌까?
2. Checked 와 Unchecked 는 무엇을 의미하는거지

이 두 질문에 대한 답은 하나로 해결된다. RuntimeException 과 이를 제외한 Exception 클래스의 차이가 바로 checked 와 unchecked 이기 때문이다.

- RuntimeException과 RE가 아닌 것의 차이는?

- checked 예외 : 반드시 예외 처리를 해야하는 것
- unchecked 예외 : 해도 되고 안 해도 되는 예외.

	Exception - Checked Exception	**RuntimeException** - Unchecked Exception
처리 여부	반드시 예외 처리 해야함.	예외 처리 하지 않아도 됨.
트랜잭션 롤백	롤백이 되지 않음.	롤백 진행.
예외 처리 확인	컴파일 중	런타임 중
발생	외부 영향으로 발생할 수 있는 것들 (사용자 동작, 운영체제 등)	개발자 실수에 의해 발생

처리 여부와 예외 처리 확인은 앞에서 언급했고, 트랜잭션 롤백이 조금 생소해서 살펴보았다.

만약 데이터베이스의 데이터를 수정하는 도중에 예외가 발생된다면 어떻게 해야 할까? DB의 데이터들은 수정이 되기 전의 상태로 다시 되돌아가져야 하고, 다시 수정 작업이 진행되어야 할 것이다.

이렇듯 여러 작업을 진행하다가 문제가 생겼을 경우 이전 상태로 롤백하기 위해 사용되는 것이 **트랜잭션(Transaction)** 이다.

트랜잭션
: 더 이상 쪼갤 수 없는 최소 작업 단위

그래서 트랜잭션은 commit으로 성공 하거나 rollback으로 실패 이후 취소되어야 한다.
하지만 모든 트랜잭션이 동일한 것은 아니고 속성에 따라 동작 방식을 다르게 해줄 수 있다.
예를 들어 1개의 새로운 데이터를 추가하는 와중에 에러가 발생하면 해당 추가 작업은 없었던 것처럼 되돌려진다. 하지만 만약 여러 개의 작업에 대해 롤백을 하려면 어떻게 해야 될까?
여러 개의 작업을 1개의 트랜잭션으로 관리해야 할 것이다.

위에서 설명한 것과 마찬가지로 트랜잭션의 마무리 작업으로는 크게 2가지가 있다.

- 트랜잭션 커밋: 작업이 마무리 됨
- 트랜잭션 롤백: 작업을 취소하고 이전의 상태로 돌림

간단히 예시를 들어 어느 경우에 checked 와 unchecked 를 사용해야하는지 살펴보자.
쿠팡 사용자 회원가입 기능이 있고, 회원가입 시 사용자에게 쿠폰을 발급한다. 사용자가 가입하는 도중에 원인 모를 상황이 발생했다고 했을 때, 비지니스 요구에 따라 Exception 상황을 둘로 나눠볼 수 있다.

1. 사용자 가입은 허용하지만, 쿠폰 발급 보류
2. 사용자 가입과 쿠폰 발급이 동시에 완료되지 못한 경우, 모두 롤백.

1번의 경우는 Checked Exception을, 2번의 경우 Unchecked Exception을 사용해야 한다.

+ 갑자기 궁금해서 추가하는 어떤 분이 Spring으로 개발하다가 Spring이 RuntimeException 과 Error를 롤백하는 상황이 벌어졌다고 작성해주신 글
  : 응? 이게 왜 롤백되는거지? | 우아한형제들 기술블로그

- 자바에서 예외 처리 방법 (try, catch, throw, throws, finally)

그럼 이 예외들을 처리하는 방법에 대해 살펴보자. 크게 둘로 나눌 수 있다.

1. 예외를 잡아서 그 자리에서 처리하는 방법: `try-catch(-finally)`, `try-with-resources`

```
// try-catch
try {
    //something
} catch (Exception 하위 클래스 e) {
    //handle Exception
}

// try-catch-finally
try {
    //something
} catch (Exception 하위 클래스 e) {
    //handle Exception
} finally {
    //Exception 발생 유무에 상관 없이 무조건 동작
}

// try-with-resources
try (Exception 처리가 필요한 객체 선언) {
       //something
} catch (Exception 하위 클래스 e) {
    //handle Exception
}
```

2. 메서드가 예외를 발생시킨다고 기술하는 방법: throws

```
public void doSomething() throws Exception {
       //something
}
```

`throws` 와 `throw`를 헷갈리면 안된다. `throws`는 예외를 처리하는 것이 아닌 **생성**하는 키워드이기 때문이다.

```
throw new (Throwable 객체)
```

예외 발생 시, 자신을 호출한 상위 메소드로 예외를 넘겨준다.

이러한 예외 처리 동작은 **호출 스택(Call Stack)**을 통해 수행된다.
만약, 호출 스택을 계속해서 탐색하면서 예외 처리를 구현한 메서드를 찾는 과정에서 해당 예외 처리를 구현한 메서드가 존재하지 않는다면 **런타임 시스템이 그냥 프로그램을 종료**시킨다.
하지만 자신에게 발생한 예외를 다른 곳에서 처리하도록 책임을 넘기는 방식이기 때문에 반드시 **상위 메서드에서 책임**을 지도록 구현해 줘야 한다.

- 커스텀한 예외 만드는 방법

만드는 방법은 경우에 따라, Exception 클래스와 RuntimeException 클래스를 상속받아 구현한다.

권장 사항은 아래와 같다.

- 클래스 이름은 Exception으로 끝내는 것을 권장.
- 생성자는 매개 변수가 없는 기본 생성자와 예외 발생 원인을 전달하기 위한 String 타입의 매개변수를 갖는 생성자 두 가지를 선언하는 것이 일반적이다.

예시 코드

```
public class CustomException extends Exception 혹은 RuntimeException {
	// 기본 생성자
	CustomException() { }
	// String 타입의 매개변수를 갖는 생성자
	CustomException(String message) {
		super(message); // Exception 혹은 RuntimeException 클래스의 생성자 호출
	}
}
```

+ 참고 링크: https://tecoble.techcourse.co.kr/post/2020-08-17-custom-exception/

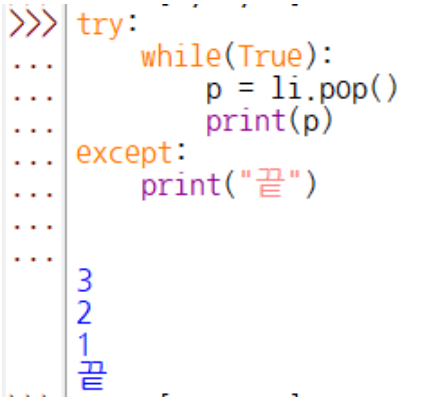
  커스텀 예외도 의견이 분분하다. 표준 예외만으로도 충분하다고 하는 측과 커스텀 예외를 사용하면 좋다고 하는 측의 의견이 잘 정리되어있다.

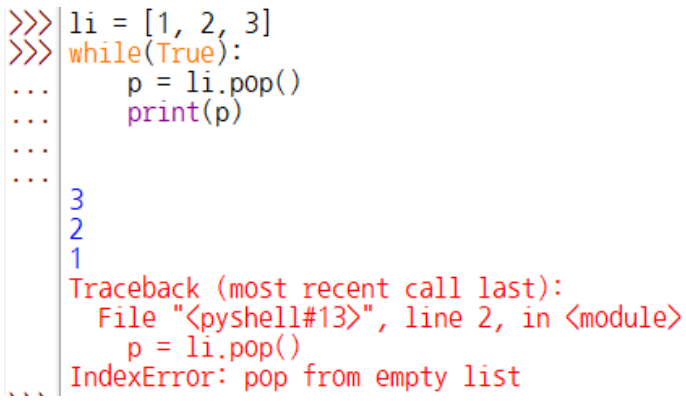
+ 참고 링크: https://parkadd.tistory.com/69

  커스텀 예외 구현 시 참고하면 좋은 4가지 방법 + Exception 예시 / RuntimeException 예시

+ tmi: 파이썬에서의 에러와 예외 차이 : https://m.blog.naver.com/youndok/222036362465

얘네는 출력했을 때 Exception이 아니라 Error라고 되어있어서 궁금해서 찾아봄.





참고 링크

- https://jithub.tistory.com/278
- 예외의 선조 - Throwable - [바로실습] 생활코딩 - 자바(JAVA)
- https://travelbeeee.tistory.com/454
- https://sorjfkrh5078.tistory.com/104
- https://dreamcoding.tistory.com/70
- [JAVA] e.toString(), e.getMessage(), e.printStackTrace() 예외처리
- https://goateedev.tistory.com/148
- [Spring] 트랜잭션에 대한 이해와 Spring이 제공하는 Transaction(트랜잭션) 핵심 기술 - (1/3)
- Spring Transaction Exception 상황에서 Rollback 처리하기
- [Java] 자바 커스텀 예외 만들기(Custom Exception)