9주차 과제

목표

자바의 예외 처리에 대해 학습하세요.

학습할 것 (필수)

- 자바에서 예외 처리 방법 (try, catch, throw, throws, finally)
- 자바가 제공하는 예외 계층 구조
- Exception과 Error의 차이는?
- RuntimeException과 RE가 아닌 것의 차이는?
- 커스텀한 예외 만드는 방법

Error에 대한 대처

https://youtu.be/srQR0Qb7Joo

참고 : catch-me-java.tistory.com/46

영상에서 말하다 시피 에러 내용을 확인하고 해결하는 것으로 굉장히 좋은 학습법이 될 수 있다.

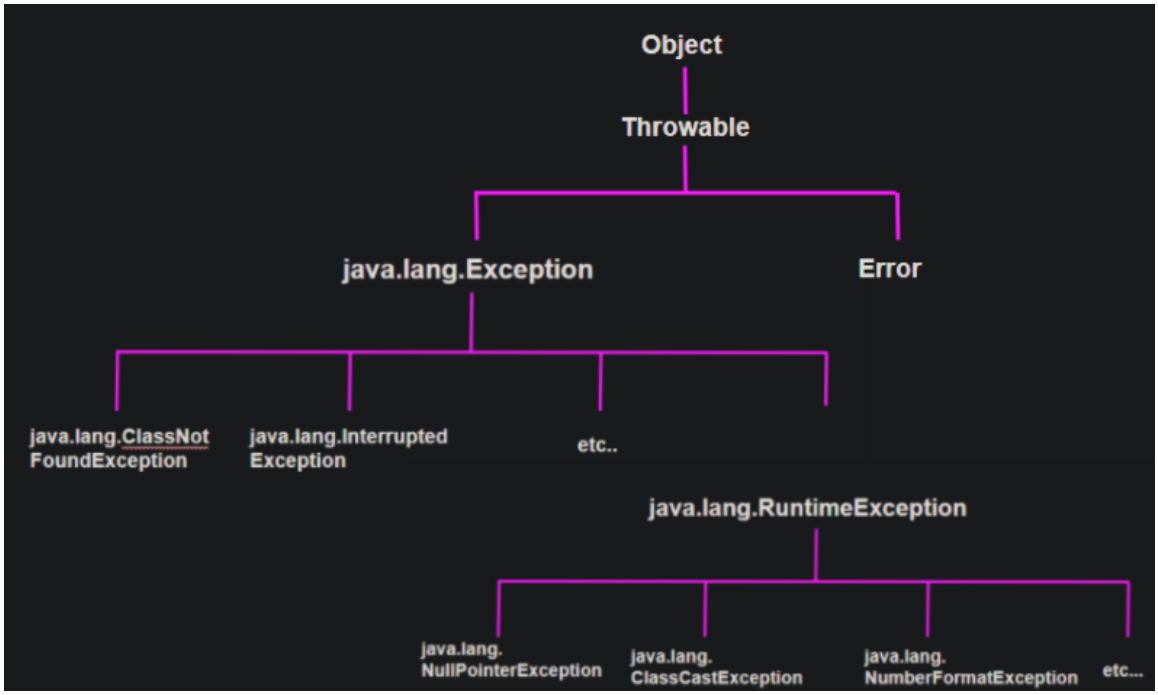
그렇다면 에러는 어떠한 종류가 있을까?

자바에서 코드를 작성하면서 발생하는 문제는 크게 2가지로 분류된다.

1. Error : JVM실행에 문제가 생김
2. Exception : 사용자의 잘못된 조작

간단히 말해서 에러는 보통 시스템 충돌이다. 예를 들면 메모리부족 등이 있고, 예외는 컴파일, 런타임시 발생하는게 대부분이다.

그렇다면 예외 계층 구조도 확인해자.



Exception (일반예외) 은 컴파일 할때,

RuntimeException (실행예외)은 컴파일 후 런타임 할때 발생하는 오류임을 알 수 있다.

Exception은 사실 Error와 비슷하게 문제가 발생하면 어플리케이션이 종료되는 특징을 가지고 있지만 자바에서는 Exception Handling(예외처리)를 통해서 프로그램이 정상 작동할 수 있게 도와준다.

예외처리를 공부하기 전에 알아야 할 Exception의 특성이 있다.

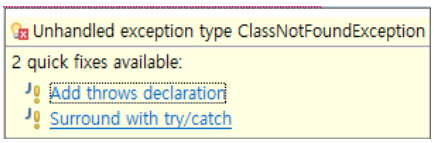
바로 Checked/Unchecked Exception 두가지 예외이다.

체크 가능/불가능 예외라는 것이다.

Checked Exception

우선 체크란, Compiler 단에서 체크가 가능한 에러를 말하는 것이다.

```
public static void main(String[] args) {
        Class clazz = Class.forName("java.lang.String2");
      //String2라는 Class 미존재
      // Output : Unhandled exception: java.lang.ClassNotFoundException , Add Exception to method signature
   }
```



위와 같이 코드를 작성해 주었을때 IDE에서는 자동으로 컴파일이 가능한 방향을 제시해준다. 이렇게 체크된 에러들, 즉 에러가 발생할 수 있을 때에는 예외처리를 진행해줘야 하는데 자바에서 는 두가지 방식으로 Checked Exception에 대한 예외처리가 가능하다.

1. throws
2. try/catch

이렇게 일반 예외는 컴파일러 측에서 개발자가 직접 처리를 작성하도록 시킨다.

```
public static void main(String[] args) {
       try {
      Class clazz = Class.forName("java.lang.String2");
    } catch (ClassNotFoundException e) {
      e.printStackTrace();
    }
  }
```

Unchecked Exception

```
public static void main(String args[])
    {
        int arr[] ={1,2,3,4,5};
        System.out.println(arr[7]);
        /*
        OutPut : Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 7 out of bounds for length 5
    at Example.main(Example.java:9)
         */
    }
```

위의 코드를 보면 컴파일러는 Checked로 예외를 잡아낼 수 없다. arr이라는 배열이 성공적으로 만들어졌고 arr[7]은 실행, Runtime시에 arr이라는 배열에 7번째 인덱스가 없다는 것을 알 수 있기 때문이다. 이렇게 컴파일러가 알 방도가 없을 때는 개발자에게 예외처리를 강제하지 않는다.

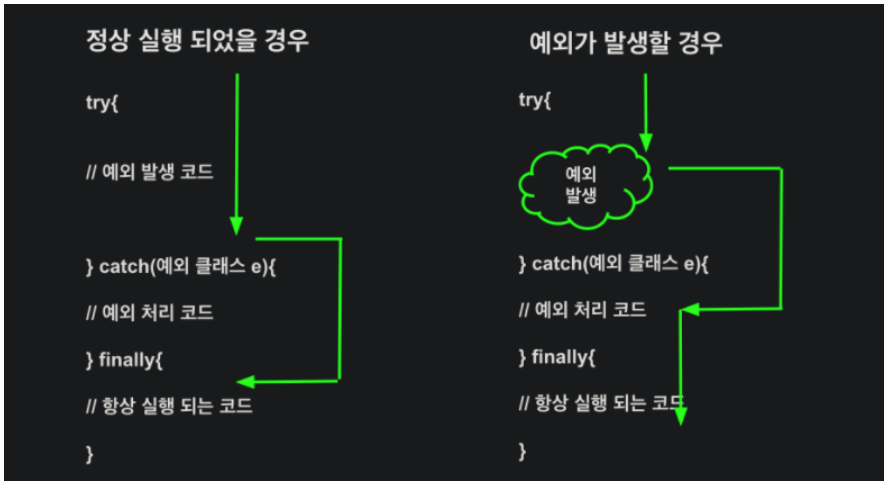
---

try ~ catch

프로그램에서 오류가 발생한다면 어플리케이션은 강제적으로 종료가 된다. 이 점에서 개발자들은 강제로 종료를 시킬 것인가, 아니면 다른 작업을 진행할 수 있게 하는가? 라는 질문에 보통 Exception 처리가 되어도 정상적인 작동을 하게 하길 원할 것이다.

그래서 갑작스러운 종료를 막고, 정상 실행을 유지할 수 있도록 처리하는 코드를 예외처리 코드라고 한다.

그렇다면 예외처리 코드의 Flow를 확인해보자




```
public static void main(String[] args) {
        try {
            Class clazz = Class.forName("java.lang.String2");
            //Exception 발생.
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
```

```
            // Error 스택 트레이스 찍음.
        } finally {
            System.out.println("HelloWorld");
            // HelloWorld 출력.
        }
   }

output: java.lang.ClassNotFoundException: java.lang.String2
    at java.base/jdk.internal.loader.BuiltinClassLoader.loadClass(BuiltinClassLoader.java:602)
    at java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(ClassLoaders.java:178)
    at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:522)
    at java.base/java.lang.Class.forName0(Native Method)
    at java.base/java.lang.Class.forName(Class.java:340)
    at Example.main(Example.java:8)
HelloWorld
```

이렇게 예외가 발생하더라도 catch 또는 fianlly로 이동해 강제종료 시키지 않고 실행이 가능하다.

지금 위에 코드는 한 가지 오류만 발생했지만 만약 2개 이상이 발생하게 된다면 어떻게 될까?

다중 catch

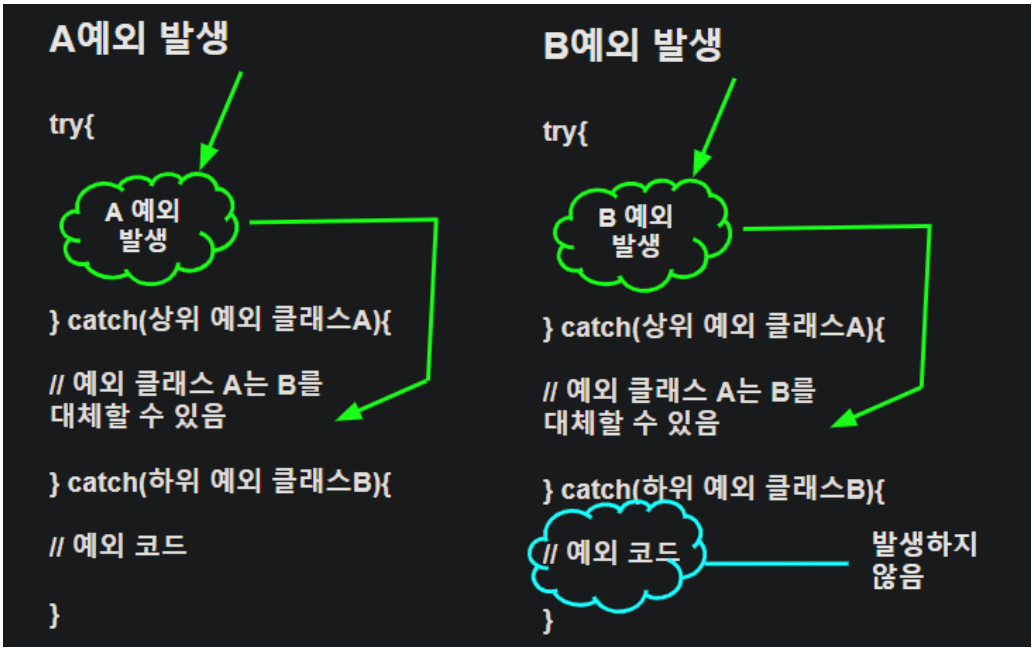
```
public class Main {
    Integer x; //int 로하면 예외 발생 X default 0때문
    public static void main(String[] args) {
        try {
            Class clazz = Class.forName("java.lang.String2");
            //ClassNotFoundException 발생.

            // 다중 catch 발생하지 않음
            Main main = new Main();
            System.out.println(main.x + 1);
            // NullPointerException 발생
        } catch (ClassNotFoundException e) {
            System.out.println("ClassNotFoundException 발생");
            e.printStackTrace();
            // Error 스택 트레이스 찍음.
        } catch (NullPointerException e){
            e.printStackTrace();
            System.out.println("NullPointerException 발생");
        } finally {
            System.out.println("HelloWorld");
            // HelloWorld 출력.
        }
    }
}
```

위의 코드에서는 ClassNotFoundException이 발생하고 그 뒤에는 Main Class의 variable X가 정의되어 있지 않아 NullPointException이 발생하게 된다. 이렇게 예외가 두 개 발생하지만 첫 예외 ClassNotFound가 발생했을 때 catch문으로 넘어오게 된다. 따라서 catch문은 첫 예외만 실행이 되게 된다.

그리고 주의해야할 점이있다.

예외의 계층도를 보면 상위예외 클래스와 하위 예외 클래스가 있는데 상위 예외 클래스는 하위 예외 클래스를 포함하기 때문에 다중 catch에서 상위 예외 클래스가 가장 위에 있다면 하위 예외 클래스는 사용할 수 없다.

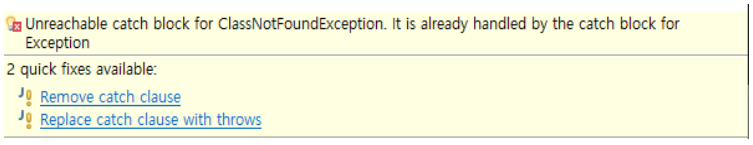


하위 예외 발생,

상위 예외클래스가 하위 예외 클래스 위에있는 경우

```
public class Main {
    Integer x;
    public static void main(String[] args) {
        try {
            Class clazz = Class.forName("java.lang.String2");
            //ClassNotFoundException 발생.

            // 다중 catch 발생하지 않음
            Main main = new Main();
            System.out.println(main.x + 1);
            // NullPointerException 발생
        } catch (Exception e) {
            System.out.println("ClassNotFoundException 발생");
            e.printStackTrace();
            // Error 스택 트레이스 찍음.
        } catch (NullPointerException e){ //하위예외 오류
            e.printStackTrace();
            System.out.println("NullPointerException 발생");
        } finally {
            System.out.println("HelloWorld");
            // HelloWorld 출력.
        }
    }
}
outPut : Error -> Exception 'java.lang.NullPointerException' has already been caught
```



하위 예외 발생,

하위 예외클래스가 상위 예외 클래스 위에있는 경우

```
public class Main {
    Integer x;
    public static void main(String[] args) {
        try {
            Class clazz = Class.forName("java.lang.String2");
            //ClassNotFoundException 발생.
```

```
            // 다중 catch 발생하지 않음
            Main main = new Main();
            System.out.println(main.x + 1);
            // NullPointerException 발생
        } catch (ClassNotFoundException e) {
            System.out.println("ClassNotFoundException 발생");
            e.printStackTrace();
            // Error 스택 트레이스 찍음.
        } catch (Exception e){
            e.printStackTrace();
            System.out.println("NullPointerException 발생");
        } finally {
            System.out.println("HelloWorld");
            // HelloWorld 출력.
        }
    }
}
```

Java7 Multi Catch

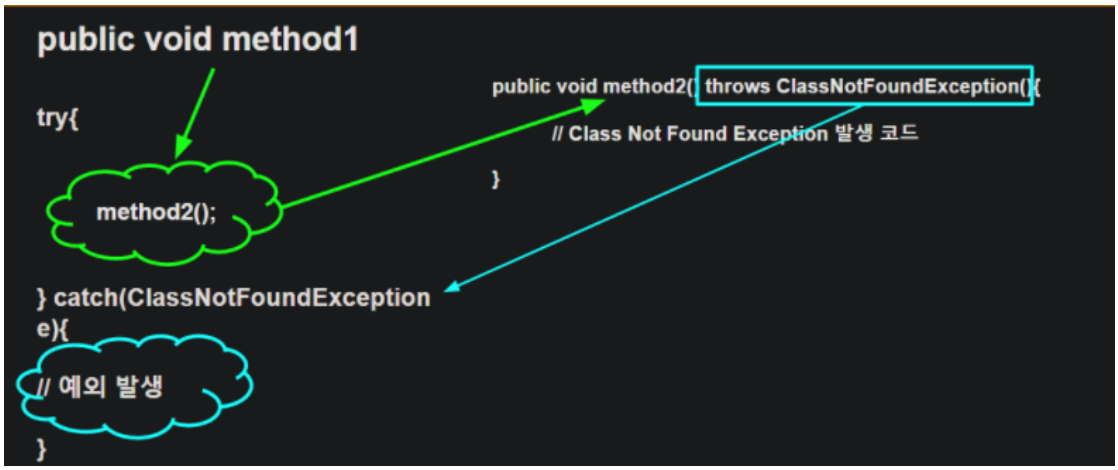
Java7 이상부터는 여러개의 catch문을 효과적으로 다룰수 있는 방법을 제안하였다. 한꺼번에 scope로 잡아주게 되는 것이다.

```
try{
    // Exception 발생 코드
}catch(예외1 | 예외2 e){
    e.printStackTrace();
}finally{
    //finally 코드
}
```

예외 던지기 throws

기본적으로 예외처리는 try catch로 하는게 보통이지만, 자신이 부른 메서드에 Exception 문을 넘겨서 처리할 수 있다.

예를 들어 Method1에서 Method2를 처리하는데 예외가 발생한다면, 이 예외는 Method2에서 발생할 수도 있지만 Method1에 예외를 던져줄 수 있다는 것이다.



모든 클래스의 상위 클래스는 Object이다.

계속 적인 throws의 끝은 결국 Object라는 말이다.

하지만 Object에 도달하기 전 throws를 처리해주는게 좋다.

사용자 정의 예외 및 예외 발생

사용자 정의 예외는 왜 배워야 하는 것일까, 사실 지금 예시로 작성한 모든 예외는 자바 문법에서 발생할 수 있는 예외일 뿐, 실제로 우리가 어플리케이션에서 발생할 수 있는, 예외를 커버하기에는 예외가 적다. 자바 입장에서는 만들어주지 않아도 되고 말

이다. 예를 들어보자

```
public class Student{
    int studentNum;
    //학생 번호
    String name;
    //학생 이름
}
```

- 만약 학생이라는 클래스가 존재한다고 가정하고, studentNum과 name이란 맴버변수를 작성한다.
- 한개의 학급에 studentNum 즉, 학생번호는 1번부터 20번까지만 가능하다고 가정해보자
- 만약 studentNum, 학생번호가 20을 초과하면 어떤 Exception을 작성해야될까?
- 이런 상황을 예방하고자, 커스텀 예외를 작성해보자

```
public class OutOfStuNumException extends Exception{
    public OutOfStuNumException(){} // 기본 생성자
    public OutOfStuNumException(String message){
        super(message); // 에러메세지 생성자
    }
}
```

사용자 정의 예외 클래스는 컴파일러가 체크하는 일반 예외로 선언이 가능하며, 컴파일러가 체크하지 않는 실행 예외로 선언할 수 있다. 클래스를 선언할 경우에는 클래스 네임 끝에 ~Exception을 붙혀주면 좋다.

2가지 생성자를 선언하여, 1개의 생성자는 기본생성자로 작성하고, 또다른 한개는 예외 오류문을 작성할 message를 작성할 수 있도록 해주면 된다.

그럼 이 예외문을 어떻게 발생시킬 수 있을까? throw new 키워드를 사용해보도록 하자.

```
public void addStuNum(Student student) throws OutOfStuNumException{
        if(studentNum > 20){
            throw new OutOfStuNumException("학생번호가 20을 초과할 수 없습니다.");
        }else{
            studentNum++; // 학생 번호 증가
        }
    }
```

다음과 같이 학생 번호를 추가시키는 addStuNum 을 작성하였다. 만약 학생의 수 studentNum이 2을 초과하게 되면 발생할 수 있도록 하였다