JAVA 12강 예외처리

강사 박주병

1. 예외의 종류

프로그램을 실행하면 예상치 못한 상황이 발생하며 에러가 날수 있다. 일반적으로 프로그램에서 에러가 발생하면 운영체제는 해당 프로그램을 종료시킨다. 하지만 어떤 문제인지에 따라서 미리 대비 할 수 있는 코드를 작성해 놓는다면 종료되지 않고 프로그램 실행이 유지 될 수 있다.

예를 들어 계산기를 만든다면 사용자가 입력값을 계산 불가능한 너무 큰 수를 입력한다면 프로그램이 강제 종료될 필요 없이 좀더 작은 수를 넣으라는 안내만 하고 다시 입력 대기를 하면 되는 것이다. 이렇게 미리 특정 문제에 대해 대비할수 있게 코드를 작성해놓는 것을 예외처리 라고 한다.

예외처리에 대해 배우기 전에 우선 프로그래밍에서 에러는 어떤 종류들이 있는지 살펴보자.

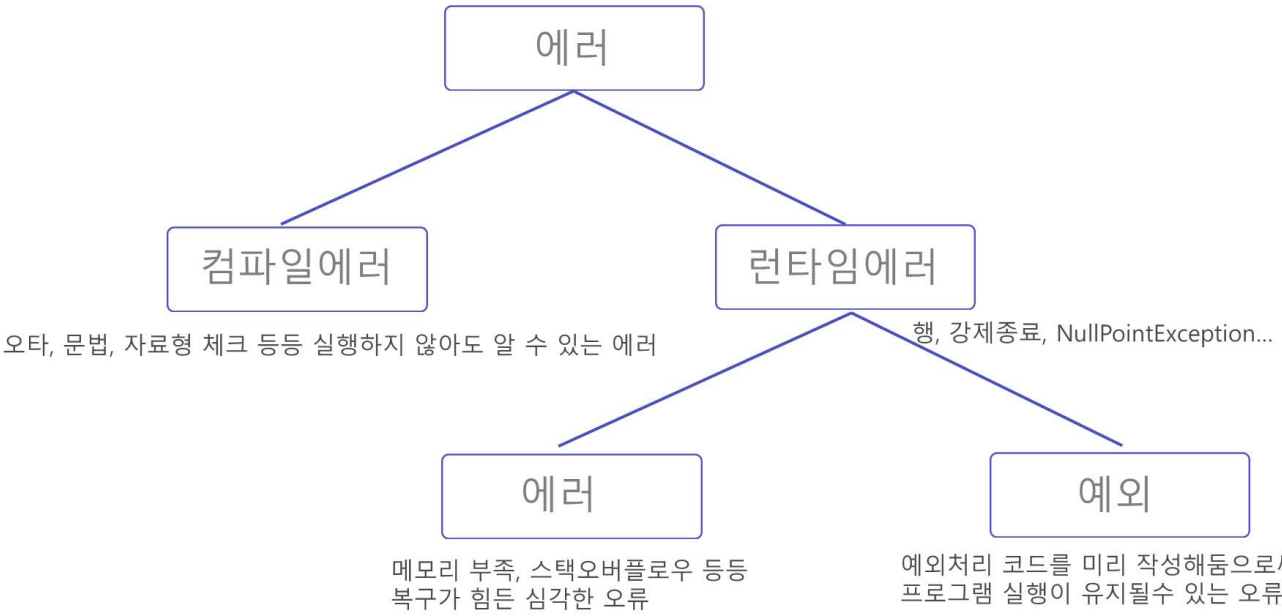


그림 1

에러는 크게 발생하는 시점에 따라 컴파일에러, 런타임에러로 나뉜다.

1.1 컴파일 에러

컴파일 에러는 이름 그대로 소스코드를 실행 가능한 형태로 컴파일 하는 과정에서 발생하는 에러 이다. 컴파일러가 구문체크 등을 하며 오타나 문법 오류 같은 것들을 검증하며 틀렸을 경우 발생하는 에러이다.

1.2 런타임 에러

런타임은 프로그램이 실행되고 있는 시간대를 의미한다. 즉 컴파일이 완료되고 프로그램이 실행중일 때 발생하는 에러이다. 예를들어 의도하지 않는 무한루프, 객체를 생성하지 않아 Null인 상태의 참조변수를 사용하는 경우, 외부 파일을 여는 과정에 존재하지 않는 파일일 경우 등등이 있다.

이러한 런타임에러는 크게 에러와 예외로 나누어 질수 있다. 에러는 프로그램이 더 이상 실행이 불가능한 치명적인 오류이다. 반면에 예외의 경우 프로그램이 강제 종료될 필요까진 없으며 개발자가 미리 조치를 취 할 수 있는 코드를 작성해두면 프로그램이 계속 실행 될 수 있을만한 오류들이다.
따라서 예외처리를 한다는 것은 런타임에러 중에 어느정도 회복이 가능한 예외들에 대해서 미리 조치를 취해놓는 것이다.

에러의 클래스 관계도

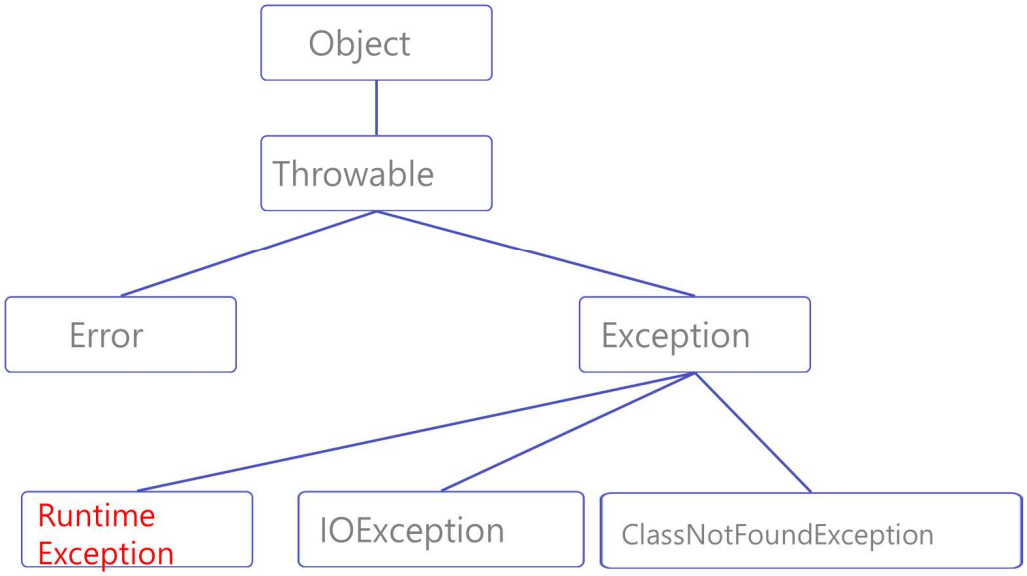


그림 2

그림2처럼 JAVA에서는 에러 역시 클래스로 정의되어 관리된다.여기서 우리는 예외처리가 가능한 Exception과 그의 자식 클래스들을 살펴볼 것이다.
우선 크게 Exception의 자식은 RuntimeException 과 그 외 기타등등 으로 분류하여 기억해두자.

1.2.1 RuntimeException

RuntimeException은 예외 중에서도 코드 내부적인 원인이나 개발자의 실수에 의해서 주로 발생한다.
아래의 예시들을 보자

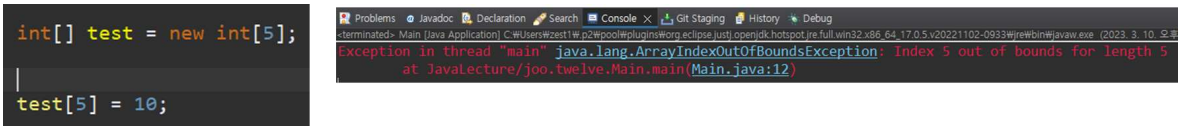


그림 3

그림3은 길이5의 배열을 생성 후 index5에 접근하려다가 index의 범위를 넘어서서 발생한 예외이다.

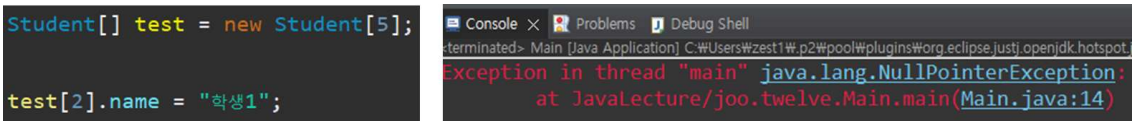


그림 4

그림4는 객체배열 사용 시 흔히들 하는 실수이다. 바로 배열만 생성 후 객체생성 없이 배열 요소의 객체를 사용할 때 발생하는 예외이다.

이처럼 RuntimeException은 프로그램 실행도중 개발자의 실수 등으로 발생하는 경우가 대부분이다.

1.2.2 그 외 Exception

RuntimeException을 제외한 나머지 많은 예외들이 있는데 이들은 주로 프로그램 외부적인 요인에 의해 발생하는 경우가 많다. 예외에 대해서 Runtime과 그 외 기타 등등으로 분류하는 이유는 조금 뒤에 배울 check, uncheck를 구분 할 때도 사용 되므로 기억 하도록 하자.

외부적 요인에 의해 주로 발생한다.

1. FileNotFoundException : 외부파일 을 찾지 못함.
2. ClassNotFoundException : 외부에서 클래스파일을 참조시 이름이 잘못
3. DataFormatException : 사용자가 잘못된 데이터를 입력

2. 예외처리

2.1 try catch문

앞서 배운 이러한 예외들은 별도의 처리를 하지 않은 채로 발생하면 발생한 line에서 프로그램은 종료된다. 하지만 예외가 발생했다고 해서 무조건 프로그램이 종료되어야 하는건 아니다. 가령 계산기를 만든다고 했을 때 사용자가 0으로 나누도록 숫자를 입력했다고 해서 프로그램이 종료될 필요가 있는가? 그냥 0으로 나누지 말라는 안내 문구를 띄우고 다시 입력을 받으면 되는 것이다. 이렇게 예상되는 예외들을 미리 케이스별로 처리를 해두는 것을 예외처리 라고 한다.

```
try
{
    int a =10/0;

}catch(NullPointerException ex)
{
    System.out.println("객체를 생성하고 쓰세요");
}
catch(ArithmeticException ex)
{
    System.out.println("0으로 나누지 마세요 ㅡㅡ");
}
```

1줄 이더라도 중괄호 생략 불가

try 영역 내부의 코드에서 실행도중 예외가 발생한다면 해당 라인에서 뒷 라인을 수행하지 않고 바로 catch 구문으로 넘어간다. catch 구문에서는 발생한 예외 타입 과 일치하는지 비교 후 일치한다면 catch구문 내부를 수행하고 그 뒤의 catch구문은 실행하지 않고 try catch구문을 벗어나 그 뒤에 코드들을 실행한다.

중첩 사용 가능

변수명 중복

```
try
{
    int a =10/0;

    try
    {
        System.out.println("try catch 중첩 사용가능");
    }catch(Exception ex)
    {

    }

}catch(NullPointerException ex)
{
    try
    {
        System.out.println("try catch 중첩 사용가능");
    }catch(Exception ex)
    {

    }
    System.out.println("객체를 생성하고 쓰세요");
}
catch(ArithmeticException ex)
{
    System.out.println("0으로 나누지 마세요 ㅡㅡ");
}
```

그림 7

잘은 쓰이지 않지만 그림7처럼 try catch문 역시 무한중첩이 가능하다.

2.2 예외의 상속관계

예외는 최상위에 Exception 클래스가 있으며 그 하위에 Runtime과 그 외 기타 등등의 클래스들이 있다.
즉 예외들도 클래스로 관리되며 부모, 자식의 관계를 가지며 다형성 역시 적용된다.

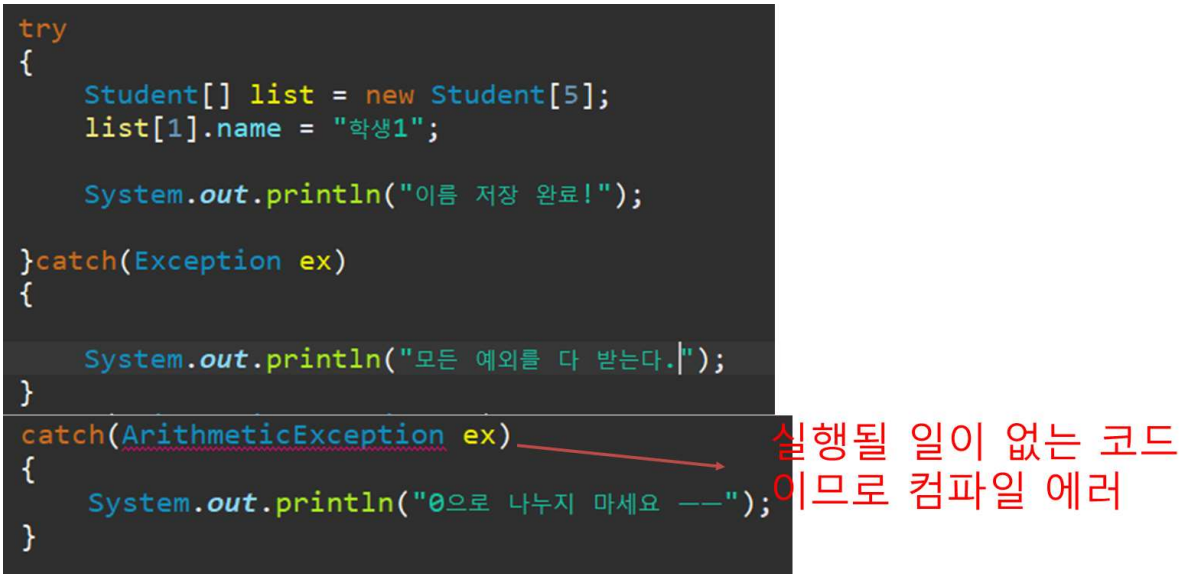


그림 8

그림8처럼 Exception은 모든 예외의 부모이기 때문에 try구문에서 발생한 모든 예외들은 catch(Exceptin ex)에서 모두 걸리게 되어 있다. 그러므로 그 밑에 catch문이 더 있더라도 절대 실행될 일이 없기에 컴파일 에러가 발생한다. 당연히 ArithmeticException 과 순서를 바꾸면 컴파일 가능하다.

2.3 예외 객체

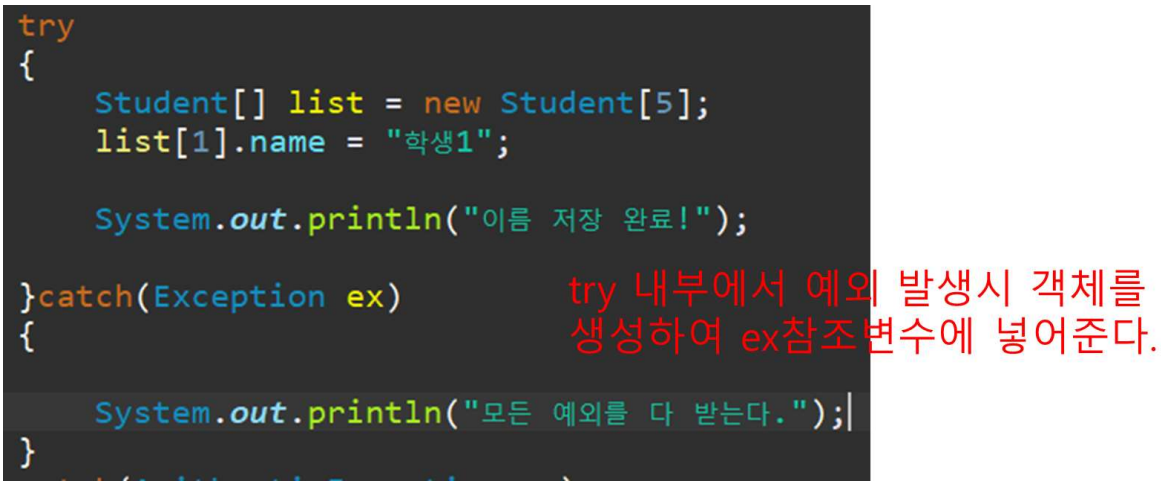


그림 9

그림9를 보면 catch문의 매개변수 부분에 Exception ex 이 선언된 것을 볼 수 있다. 예외발생시 ex변수에 해당 예외 객체를 담아 넘겨준다. ex는 예외에 대한 정보들을 가지고 있다. 이를 활용하여 예외처리를 해도 되며 필요 없다면 사용하지 않아도 된다. 다만 매개변수 선언은 반드시 해줘야 한다.

예외 객체에는 여러 가지 정보들이 있지만 대표적으로 아래와 같이 메서드 2개를 살펴보자.

printStackTrace()

예외가 발생된 메서드의 정보 및 예외 메시지를 화면에 출력

getMessage()

예외클래스의 인스턴스에 저장된 메시지를 String으로 반환

```
try
{
    System.out.println(1);
    Student[] list = new Student[5];

    System.out.println(2);
    list[1].name = "학생1";

    System.out.println("이름 저장 완료!");

}catch(Exception ex)
{
    ex.printStackTrace();

    System.out.println(ex.getMessage());
}
```

예외 객체에서 제공하는 위의 메서드들을 사용해보면 발생한 예외에 대한 정보를 볼수 있다.

```
1
2
java.lang.NullPointerException: Cannot assign field "name" because "list[1]" is null
        at JavaLecture/joo.twelve.Main.main(Main.java:16)
Cannot assign field "name" because "list[1]" is null
```

2.4 예외 던지기

이러한 예외는 코드가 문제가 되어 발생 할 수도 있지만 개발자가 임의로 발생 시킬 수도 있다. 여기서 예외를 왜 일부러 발생시켜야 하는지 의문점이 들 수 있다. 이는 뒷부분에서 자세히 다루도록하며 지금은 예외를 임의로도 발생 시킬 수 있다는 사실과 문법만 보도록 하자.

```
try
{


    int a = 10/1;

    //예외 생성
    ArithmeticException e = new ArithmeticException("1로 나누면 에러 발생!");
    //예외 던지기
    throw e;

}catch(ArithmeticException  ex )
{
    ex.printStackTrace();

    System.out.println(ex.getMessage());
}
```

그림 12

위의 그림처럼 예외역시 클래스이기에 발생시키고 싶은 적절한 클래스의 객체를 생성후 throw 키워드와 해당 객체를 사용하면 throw e; 라인에서 해당 예외가 발생하여 프로그램은 종료된다. 위의 그림에서는 물론 예외처리를 하였기에 catch문으로 넘어갈 것이다.

2.5 checked VS unchecked

예외에는 발생 시 반드시 예외처리를 해야만 하는 것이 있고 하지 않아도 되는 것으로 분류된다.

특정 메서드를 사용하다보면 해당 메서드를 사용하려면 반드시 Try 구분 내부에서 써야 된다는 것들이 있다. 이것들이 바로 checked 예외를 발생시키는 메서드 들이다. unchecked는 예외처리를 강제하지 않는 것들이다.

지금까지 JAVA 코드를 작성하면서 Try Catch를 사용했는가? 아마 하지 않았을 것이다. 즉 지금까지 사용했던 메서드나 클래스들은 모두 unchecked 예외를 발생시키기 때문에 try Catch를 하지 않아도 컴파일이 가능했던 것이다. 다만 Try Catch를 안 해도 된다는 것이다 예외가 발생하면 발생한 지점에서 강제 종료되는건 마찬가지이다.

아래의 그림은 조금전 배운 throw를 이용해 임의로 checked 예외와 unchecked 예외를 발생시키는 모습이다.

```java
public static void main(String[] args) {
    // TODO Auto-generated method stub

    int a = 10/1;

    //예외 생성
    Exception e = new Exception("1로 나누면 에러 발생!");
    //예외 던지기
    throw e;

}
```

```java
public static void main(String[] args) {
    // TODO Auto-generated method stub

    int a = 10/1;

    //예외 생성
    RuntimeException e = new RuntimeException("1로 나누면 에러 발생!");
    //예외 던지기
    throw e;
```

checked 예외 : 반드시 예외처리 해야 한다.

unchecked 예외 : 컴파일은 통과된다.

예외는 Runtime과 그 외 기타 등등으로 분류하라고 했던 것을 기억 하는가? 바로 이 분류 기준이 checked 와 unchecked 이다.

RuntimeException은 예외처리를 강제하지 않는 unchecked 이며 그 외 나머지 예외들은 발생할 가능성이 있는 메서드를 사용시 반드시 Try Catch문을 사용해야 한다.

2.6 예외 넘기기

예외 발생시 try catch문을 통해 예외처리를 할 수 있지만 그 예외를 해당 메서드가 아니라 메서드를 호출한 상위 코드로 넘길 수도 있다.

```java
void method() throws ArithmeticException,NullPointerException
{
    int a = 10/0;
    System.out.println("1");
}
```

메서드 내부에서 해당 예외 발생시 메서드를 호출한쪽으로 예외를 넘긴다.

그림 14

그림14를 보면 0으로 나누면서 예외가 발생할 것이다. try catch문이 없어 마치 해당 예외지점에서 프로그램이 종료될 것처럼 보이지만 method() 오른쪽을 보면 throws 라는 구문을 통해 특정 예외가 발생하면 method()를 호출한 쪽으로 예외를 넘길 수 있다.

즉 method메서드 내부에서 ArithmeticException,NullOpinterException 이 발생한다면 바로 아래 1은 출력되지 않고 method()를 호출한 라인으로 예외가 넘어간다. 물론 그 외에 나머지 예외가 발생한다면 예외처리를 하지 않은것과 마찬가지로 해당 라인에서 프로그램이 종료 된다.
그리고 method() 내부를 보면 NullPointerException은 발생할 가능성이 없다. throws는 그런 것을 체크하지 않는다. 그저 내부에서 발생하는 예외가 어떤 건지 체크하고 throws에 명시된 예외라면 상위코드로 넘기는역할을 한다.

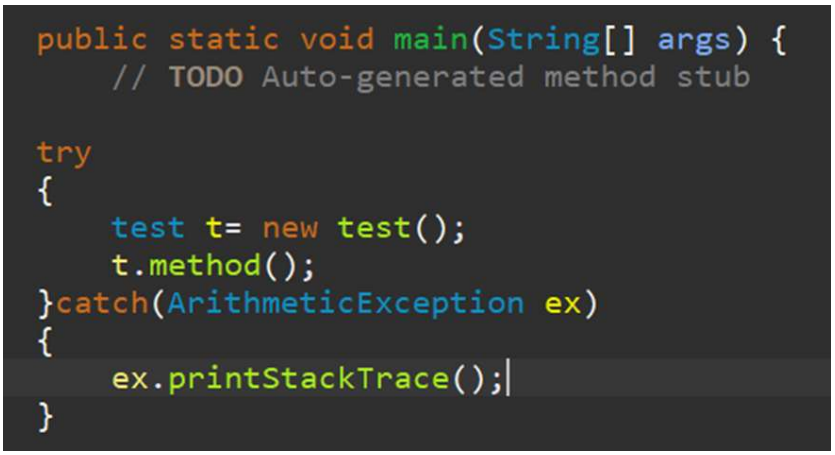


그림 15

그림15처럼 method()를 호출한 라인에서 예외가 발생하며 예외처리하고 있다.
즉 예외를 상위코드로 넘긴다 하여도 호출하는 쪽에서 예외처리를 하지 않으면 호출하는 라인에서 프로그램이 종료된다.

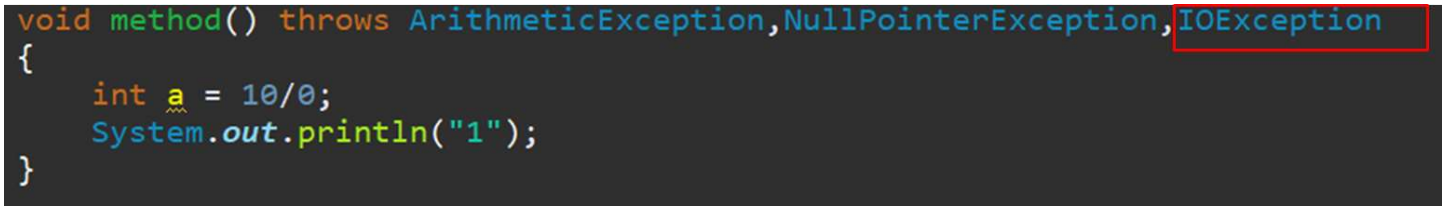
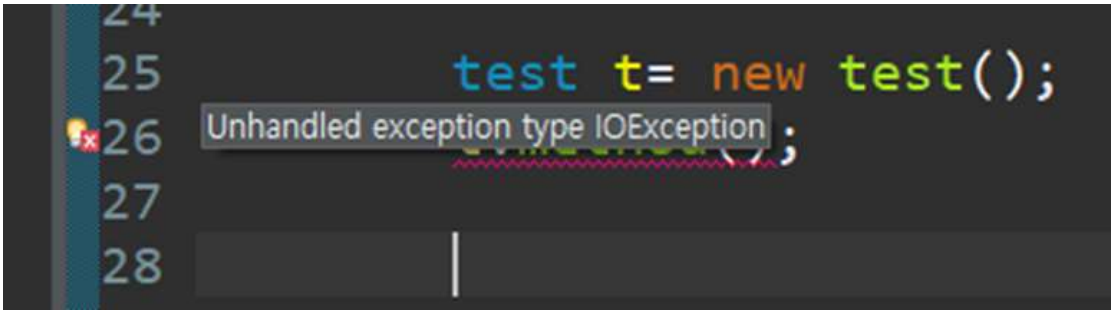


그림 16
만약 위와 같이 IOException처럼 checked예외에 해당 되는 예외를 넘긴다면 method()를 호출하는쪽에서는 반드시 예외처리를 해주어야 한다.



2.7 finally

예외처리시 예외가 발생하든 안하든 무조건 실행해 주어야 하는 코드들이 있다. 예를들어 파일입출력 관련해서 스트림을 열고 작업을 다 하면 반드시 close()를 하여 스트림을 닫아주어야 한다. 하지만 작업 도중 예외가 발생한다면 해당 라인에서 바로 catch구문으로 넘어 가기 땜에 실행이 안된다.

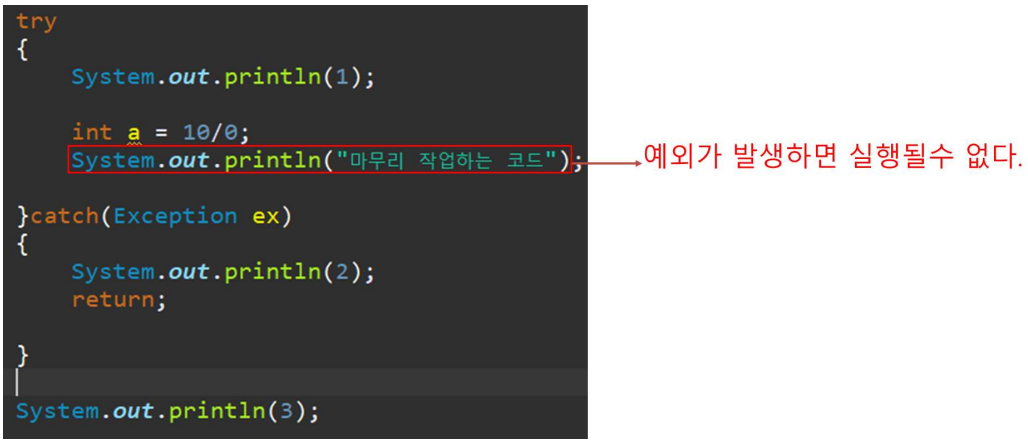
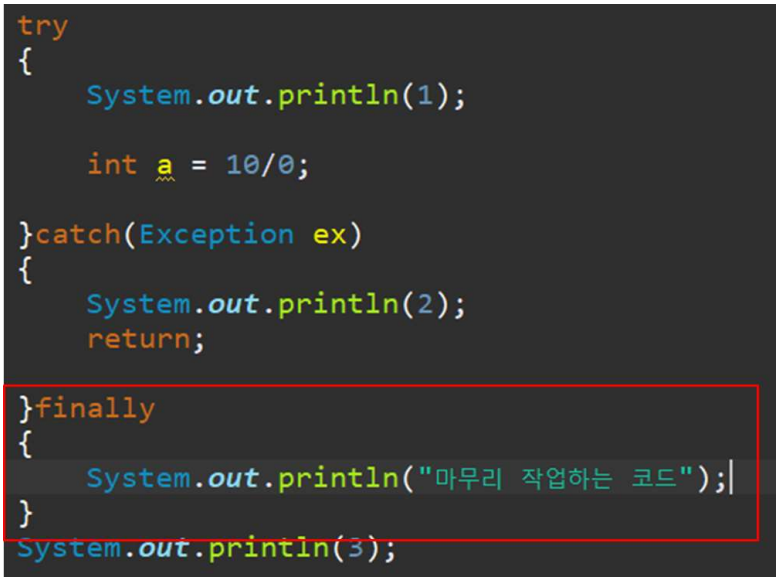


그림 18

그렇다고 catch문에 해당 코드를 넣어둔다면 정상 실행됐을경우에 실행되지 않는다.
그렇다면 그림18에서 println(3);의 위에 작성한다면 어떨까? 만약 스트림이나 등등 마무리 작업을 해야 할 참조변수가 try문 내부에서 만들어졌다면 try문 밖에서는 변수의 라이프사이클이 종료되어 사용할 수가 없다.

이럴 경우 예외가 발생하든 안하든 상관없이 무조건 마지막에 실행할수 있는 방법이 있다.



finally 키워드는 예외처리 영역에서 정상실행되든 예외가 발생하든 심지어 catch에서 return;으로 종료를 하더라도 finally는 무조건 마지막에 실행을 하고 return이 되는 기능을 한다.

3. 사용자 정의 예외

JAVA에서는 예외 역시 클래스라고 했다. 따라서 해당 클래스를 상속받아 사용자 정의 예외를 만들 수 있다.
여기서 예외는 최대한 발생하지 않아야 하고 발생하더라도 예외처리를 통해 프로그램을 지속시켜야 하는데 예외를 일부러 발생시킨다는 것이 이해가 가지 않을 수 있다.이유는 조금 뒤에 보도록하고 우선은 문법을 먼저 보도록 하자

```
public class MyException extends Exception{


    public MyException(String msg) {
        super(msg);
    }

}
```

그림 20

사용자 정의 예외를 만드는 방법은 간단하다 원하는 예외 클래스를 상속받고 생성자에서는 부모의 생성자를 호출해주면 된다. 그러면 일반적인 예외 사용시 쓰는 throw throws catch키워드 등에 사용될수 있다.

```
try
{

    throw new MyException("내가만든 예외");

}catch(Exception ex)
{
    System.out.println(ex.getMessage());
}
```

그리고 이렇게 만든 사용자 정의 예외는 부모가 checked 인지 unchecked인지에 따라 그 특성이 그대로 물려받는다. 그림20처럼 Exception 클래스를 상속받았다면 그 사용자정의예외는 checked예외이다.(RuntimeException과 그 자식들은 unchecked예외이고 Exception을 포함한 나머지들은 모두 checked예외이다.)

이제 이러한 사용자정의를 왜 만들고 어디에 쓰이는지를 보도록 하자.

```
public class CanNotRepairException extends RuntimeException
{

    public CanNotRepairException(String msg)
    {
        super(msg);
    }
}
public Boolean repaire(Weapon weapon)
{
    if(weapon instanceof Repairable)
        weapon.durability+=10;
    else
        throw new CanNotRepairException("수리불가");


    return true;
}
```

반환값 보다 더욱 풍성하게
메서드 내부의 상황을
외부에 알려줄수 있다.

그림 22

위처럼 repaire메서드의 결과를 메서드를 호출한쪽에 전달하려고 하는데 반환타입인 Boolean만으로는 전달하고자하는 정보를 모두 표현 못할수도 있다. 이럴 경우 사용자정의예외를 사용한다면 좀더 풍부하게 메서드내부의 실행결과에 대해 외부로 알려 줄 수 있다.