

제7장

예외 처리(Exception Handling)



1. 오류의 종류

▣ 에러(Error)

- 의도치 않게 프로그램이 종료되는 것을 에러라 한다.
- 하드웨어의 잘못된 동작 또는 고장으로 인한 오류(ex. 정전, 배드섹터 등)
- **에러가 발생하면 프로그램이 비정상 종료가 된다.**
- 정상 실행 상태로 돌아갈 수가 없다.

▣ 예외(Exception)

- **사용자의 잘못된 조작 또는 개발자의 잘못된 프로그래밍으로 인한 오류를 말한다.**
- 예외가 발생하면 프로그램이 종료된다.
- 단, 예외 처리를 추가하면 정상 실행 상태로 되돌릴 수가 있다.

*** 예외 처리의 목적 : 프로그램을 정상 실행 상태로 돌리는 것!!(중요함)**

2. 예외의 종류

▣ 일반 예외(컴파일 예외, **checked exception**)

- 예외 처리 코드가 없다면, 컴파일 예외 발생함(**try~catch문 이용**)

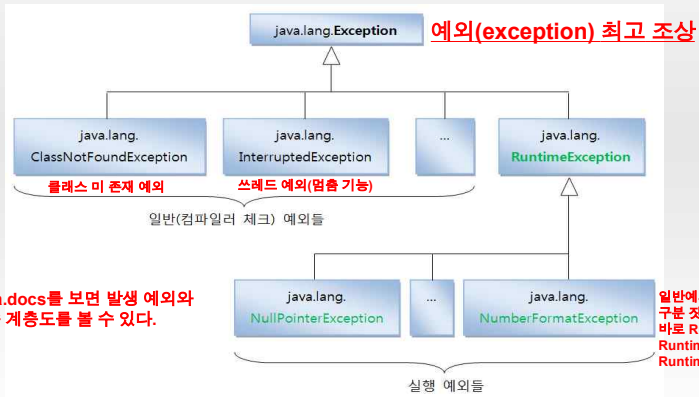
▣ 실행 예외(Runtime-Exception, **unchecked exception**)

- 예외 처리 코드를 생략하더라도 컴파일이 되는 예외를 말한다.

(ex. NullPointerException, ArrayIndexOutOfBoundsException, NumberFormatException 등)

- 단, 프로그래머의 경험 혹은 노하우에 따라 예외 처리 코드 작성이 필요하다.

3. 예외 클래스



* java.docs를 보면 발생 예외와 상속 계층도를 볼 수 있다.

일반예외인가? 런타임 예외인가?
구분 짓는 기준이 되는 클래스가 바로 RuntimeException이다.
RuntimeException을 상속 받으면 RuntimeException이 된다.

4. 실행 예외(Runtime exception) - 1

■ ArrayIndexOutOfBoundsException

- 배열에서 인덱스를 초과하여 접근할 때, 발생하는 예외이다.

```
int[] score = new int[5];  
//방이 5개인데 5번 방이 없다.  
score[5] = 99;
```



Console

<terminated> Array_Example02 [Java Application] C:\Program Files\Java\jdk1.8.0_131\bin\javaw.exe (2019. 8. 21. 오후 1:59:11)

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
at sec01_firstArray.Array_Example02.main(Array_Example02.java:15)

■ NullPointerException

- 인스턴스가 없는데, 참조하려고 할 때, 발생하는 예외이다.

```
String str = null;  
str.equals("값1");
```



Console

<terminated> Array_Example02 [Java Application] C:\Program Files\Java\jdk1.8.0_131\bin\javaw.exe (2019. 8. 21. 오후 2:02:12)

Exception in thread "main" java.lang.NullPointerException
at sec01_firstArray.Array_Example02.main(Array_Example02.java:13)

4. 실행 예외(Runtime exception) - 2

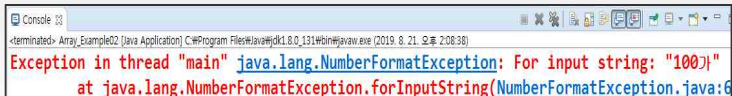
■ NumberFormatException

– 문자를 숫자로 바꾸는 메서드에서 숫자로 바꿀 수 없는 문자가 존재할 때 발생하는 예외이다.

반환 타입	메서드명(매개 변수)	설명
int	Integer.parseInt(String s)	주어진 문자열을 정수로 변환하여 리턴함
double	Double.parseDouble(String s)	주어진 문자열을 실수로 변환하여 리턴함

```
String str1 = "100";  
String str2 = "100가";  
int a = Integer.parseInt(str1);  
int b = Integer.parseInt(str2);
```

"100가"는 숫자로 변환이 가능한가?



```
Console  
<terminated> Array_Example02 [Java Application] C:\Program Files\Java\jdk1.8.0_131\bin\javaw.exe (2019. 8. 21. 오후 2:08:38)  
Exception in thread "main" java.lang.NumberFormatException: For input string: "100가"  
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)  
    at java.lang.Integer.parseInt(Integer.java:592)  
    at java.lang.Integer.parseInt(Integer.java:615)  
    at Array_Example02.main(Array_Example02.java:10)
```

4. 실행 예외(Runtime exception) - 3

■ ArithmeticException

- 산술적으로 연산이 되지 않을 때, 발생하는 예외이다.

```
int result = 10/0;  
System.out.println(result);
```

모든 숫자가 0으로 나누어지는가?

```
Console  
<terminated> Array_Example02 [Java Application] C:\Program Files\Java\jdk1.8.0_131\bin\javaw.exe (2019. 8. 21. 오후 2:11:44)  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
at sec01_firstArray.Array_Example02.main(Array_Example02.java:13)
```

■ ClassCastException

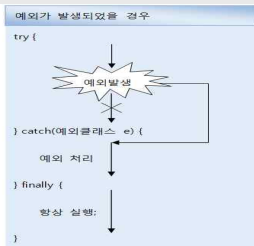
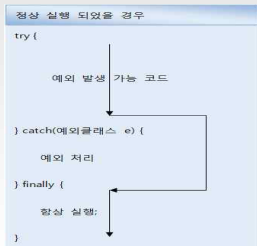
- 타입 변환이 되지 않을 때, 발생하는 예외이다.(다형성에서 많이 보았다.)

```
class A{  
class B extends A{  
class C extends A{  
  
public class Ex {  
    public static void main(String[] args) {  
  
        A a = new B();  
        C c = (C)a;  
    }  
}  
a가 누구의 인스턴스를 참조하고 있는가?
```

```
Console  
<terminated> Ex [Java Application] C:\Program Files\Java\jdk1.8.0_131\bin\javaw.exe (2019. 8. 21. 오후 2:18:06)  
Exception in thread "main" java.lang.ClassCastException  
at sec01_firstArray.Ex.main(Ex.java:13)
```

5. 예외 처리 코드(try~catch~finally)

- 예외 발생 시 예외 처리 코드를 작성하면, 프로그램의 종료를 막고 정상적으로 실행을 유지할 수 있도록 처리한다.
 - 일반 예외 : 반드시 작성해야 한다.(컴파일 조차 시켜주지 않는다.)
 - 실행 예외 : 컴파일러가 체크를 해주지 않는다. 하여, 개발자의 경험으로 작성한다.
- try-catch-finally구문으로 예외 처리 코드 작성한다.



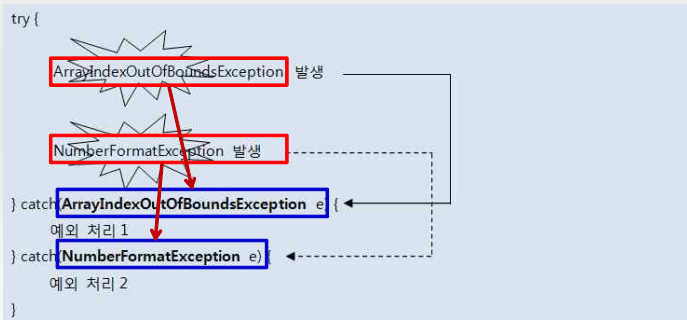
Thread.sleep(1000);

일반 예외 발생하는 예이다.

* finally는 선택사항이다.
작성시 매번 실행된다.
예외가 발생을 하든
안하든 무조건 실행된다.

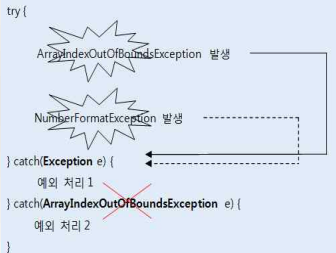
6. 다중 catch

▣ 예외 별로 예외 처리 코드를 다르게 구현하고자 할 때, 사용한다.

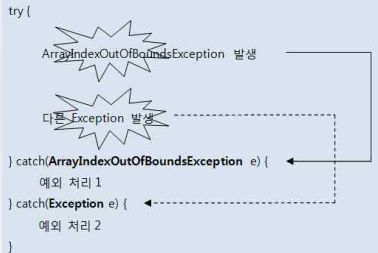


7. catch구문의 순서

▣ 다중 catch구문을 작성할 때, 상위 클래스가 위에 있으면 안된다.(다형성으로 기인함)



틀린 코드



올바른 코드

- * `Exception` 클래스는 예외의 최상위 클래스이다. 발생 예외는 무조건 `Exception` 클래스를 상속 받으므로 하위 catch구문은 절대 실행이 안 된다.
즉, 다시 말해 `Exception`의 작성은 catch문의 맨 아래에 작성을 해야 한다.

8. 멀티(multi) catch구문

- ▣ JDK1.7부터 하나의 catch구문에서 복수 개 이상의 예외를 처리하고자 할 때 사용한다.
같이 처리하고 싶다면, | (파이프) 로 연결한다.

```
try {
```

ArrayIndexOutOfBoundsException 또는 NumberFormatException 발생

다른 Exception 발생

```
} catch(ArrayIndexOutOfBoundsException | NumberFormatException e) {
```

예외 처리 1

```
} catch(Exception e) {
```

예외 처리 2

```
}
```

OR 개념(코드 중복 제거) → 파이프라고도 함

9. 예외 떠 넘기기 - throws

■ 메서드 선언부 끝에 throws를 붙여서 작성함.

- 메서드를 호출한 곳에서 예외를 처리하라고 떠 넘긴다.
- 호출한 곳에서는 반드시 예외처리 코드가 있어야 한다.

```
public class Ex {  
    public static void main(String[] args) {  
        try {  
            method();  
        } catch (Exception e) {}  
    }  
  
    public static void method() throws Exception {  
        System.out.println("예외를 던집니다.");  
    }  
}
```

10. 강제로 예외 발생시키기 - throw

■ 코드에서 강제로 예외를 발생시킨다.

```
public class Ex {  
    public static void main(String[] args) {  
  
        try {  
            method();  
        } catch (Exception e) {  
            System.out.println(e.getMessage());  
        }  
    }  
  
    public static void method() throws Exception {  
        System.out.println("예외를 던집니다.");  
        throw new Exception("예외 발생!!!");  
    }  
}
```

Console

<terminated> Ex [Java Application] C:\Program

예외를 던집니다.

예외 발생!!!

throw new XXXException("메시지") 예외 발생 코드를 가지고 있으면 throws 키워드를 선언부에 반드시 기재하거나 자신의 메서드에서 예외처리를 해주어야 하며, 만약 자신의 메서드에서 예외처리를 하기 싫다면, 호출한 곳에서는 반드시 예외 처리를 해주어야 한다.

throws : 예외를 던지는 것(호출한 곳으로 예외를 떠넘김)

throw : 예외를 발생시키는 것

상기 두 개의 키워드는 완전히 다른 개념이다.

11. 사용자 정의 예외 클래스 만들기

- 자바 표준 API에서 제공하지 않는 예외는 직접 프로그래밍 해서 만들어야 한다.
- 응용 어플리케이션 서비스에서 발생하는 예외
ex) 슈팅게임에서 비행기가 체력이 다 되어도 터지지 않는 경우,
회원 가입실패, 계좌 잔고 부족, 계좌 이체 실패 등
- 사용자 정의 예외 클래스 선언 방법은 아래와 같다.

```
public class NotExistIDException extends Exception {  
  
    public NotExistIDException() {  
    }  
  
    public NotExistIDException(String message){  
        super(message);  
    }  
}
```

순서

1. 관례적으로 Exception을 붙여 준다.(예외니깐)
2. 일반예외나? 런타임 예외나? 선택하여 상속받자.
3. 기본 생성자를 하나 선언하자.
4. 조상(Exception) 생성자를 반드시 호출하자.(발생 이유와 함께)

12. 예외 정보 얻기

■ getMessage()

- 예외를 발생시킬 때, 생성자의 매개값으로 사용한 String값 리턴

■ catch구문에서 활용한다.

```
catch (NotExistIDException e) {  
    System.out.println(e.getMessage());  
}
```

■ printStackTrace()

- 개발 시에 자주 사용하며, 개발이 끝나고 배포시 주석처리나 제거한다.
- 예외 발생 코드를 추적하여, 그 로그들을 전부 콘솔에 표시한다.
- 개발자 전용이며, 프로그램 테스트 하며 디버깅할 때 유용하다.

```
catch (NotExistIDException e) {  
    e.printStackTrace();  
}
```

감사합니다.

