

java

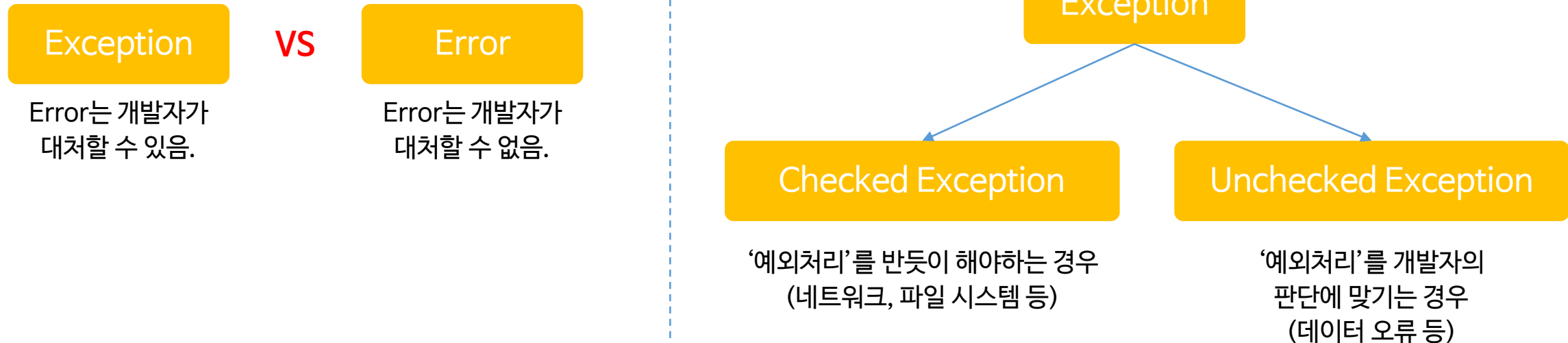
26강_예외처리

프로그램에 문제가 발생했을 때 시스템 동작에 문제가 없도록 사전에 예방하는 코드를 작성하는 방법에 대해서 학습합니다.

- 26-1 예외란?
- 26-2 Exception 클래스
- 26-3 try ~ catch
- 26-4 다양한 예외처리
- 26-5 finally
- 26-6 throws

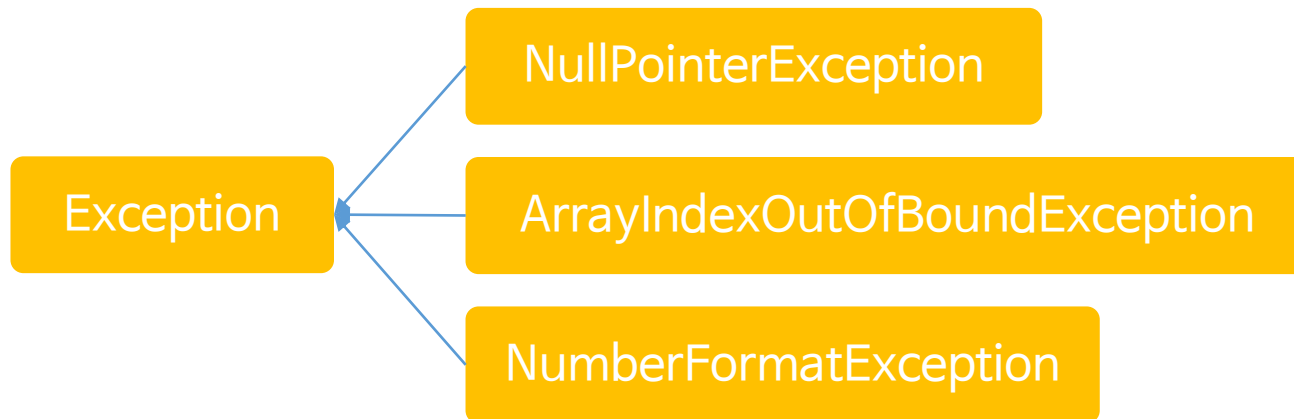
26-1 : 예외란?

프로그램에 문제가 있는 것을 말하며, 예외로 인해 시스템 동작이 멈추는 것을 막는 것을 ‘예외처리’라고 한다.



26-2 : Exception 클래스

Exception클래스 하위클래스로 NullPointerException, NumberFormatException 등이 있다.



객체를 가리키지 않고 있는 레퍼런스를 이용할 때

배열에서 존재하지 않는 인덱스를 가리킬 때

숫자데이터에 문자데이터등을 넣었을 때

26-3 : try ~ catch

개발자가 예외처리하기 가장 쉽고, 많이 사용되는 방법 이다.

```
try {  
    예외가 발생할 수 있는 코드  
} catch (Exception e) {  
    예외가 발생했을 때 처리할 코드  
}
```

```
int i = 10;  
int j = 0;  
int r = 0;
```

```
System.out.println("Ecexption BEFORE");
```

```
try {  
    r = i / j;  
} catch (Exception e) {  
    e.printStackTrace();  
    String msg = e.getMessage();  
    System.out.println("Exception: " + msg);  
}
```

```
System.out.println("Ecexption AFTER");
```

26-4 : 다양한 예외처리

Exception 및 하위 클래스를 이용해서 예외처리를 다양하게 할 수 있다.

```
try {
    System.out.println("input i : ");
    i = scanner.nextInt();
    System.out.println("input j : ");
    j = scanner.nextInt();
    System.out.println("i / j = " + (i / j));

    for (int k = 0; k < 6; k++) {
        System.out.println("iArr["+ k + "] : " + iArr[k]);
    }

    System.out.println("list.size() : " + list.size());
} catch (InputMismatchException e) {
    e.printStackTrace();
} catch (ArrayIndexOutOfBoundsException e) {
    e.printStackTrace();
} catch (Exception e) {
    e.printStackTrace();
}
```

[java.util.InputMismatchException](#)

Exception AFTER

```
at java.util.Scanner.throwFor(Unknown Source)
at java.util.Scanner.next(Unknown Source)
at java.util.Scanner.nextInt(Unknown Source)
at java.util.Scanner.nextInt(Unknown Source)
at lec26Pjt001.MainClass002.main(MainClass002.java:25)
```

[java.lang.ArrayIndexOutOfBoundsException: 5](#)

```
at lec26Pjt001.MainClass002.main(MainClass002.java:29)
```

[java.lang.NullPointerException](#)

```
at lec26Pjt001.MainClass002.main(MainClass002.java:32)
```

26-5 : finally

예외 발생 여부에 상관없이 반드시 실행된다.

```
try {  
    ...  
} catch (InputMismatchException e) {  
    ...  
} catch (ArrayIndexOutOfBoundsException e) {  
    ...  
} catch (Exception e) {  
    ...  
} finally {  
    System.out.println("예외 발생 여부에 상관없이 언제나 실행 됩니다.");  
}
```



```
iArr[4] : 4  
예외 발생 여부에 상관없이 언제나 실행 됩니다.  
Exception AFTER  
java.lang.NullPointerException  
    at lec26Pjt001.MainClass003.main(MainClass003.java:33)
```

26-6 : throws

예외 발생 시 예외 처리를 직접 하지 않고 호출한 곳으로 넘긴다.

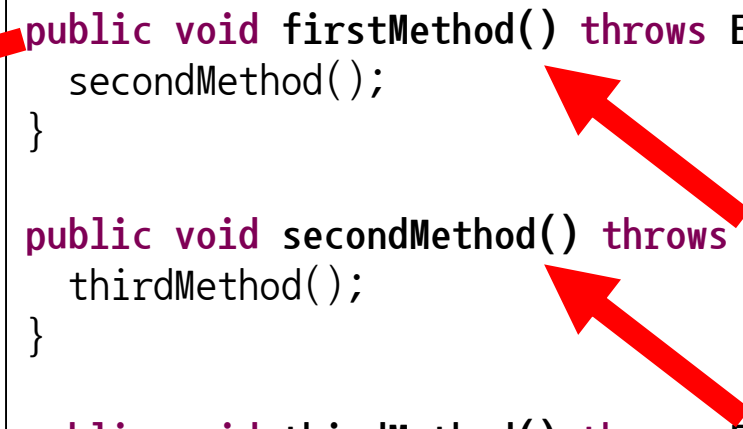
```
MainClass004 mainClass004 = new MainClass004();

try {
    mainClass004.firstMethod();
} catch (Exception e) {
    e.printStackTrace();
}
```

```
public void firstMethod() throws Exception {
    secondMethod();
}

public void secondMethod() throws Exception {
    thirdMethod();
}

public void thirdMethod() throws Exception {
    System.out.println("10 / 0 = " + ( 10 / 0 ));
}
```

A diagram consisting of three red arrows. The first arrow originates from the 'throws Exception' clause in the 'firstMethod()' signature and points to the 'catch (Exception e)' block in the main code. The second arrow originates from the 'throws Exception' clause in the 'secondMethod()' signature and points to the 'throws Exception' clause in the 'firstMethod()' signature. The third arrow originates from the 'throws Exception' clause in the 'thirdMethod()' signature and points to the 'throws Exception' clause in the 'secondMethod()' signature. This illustrates the chain of exception handling where the exception is passed back up the call stack.