

2025년 상반기 K-디지털 트레이닝

# 예외 처리

문법에러 보단 런타임에러 주의 런타임에러->앱 죽음

[KB] IT's Your Life



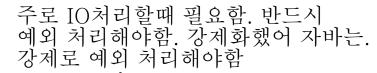
### 🧿 예외와 에러

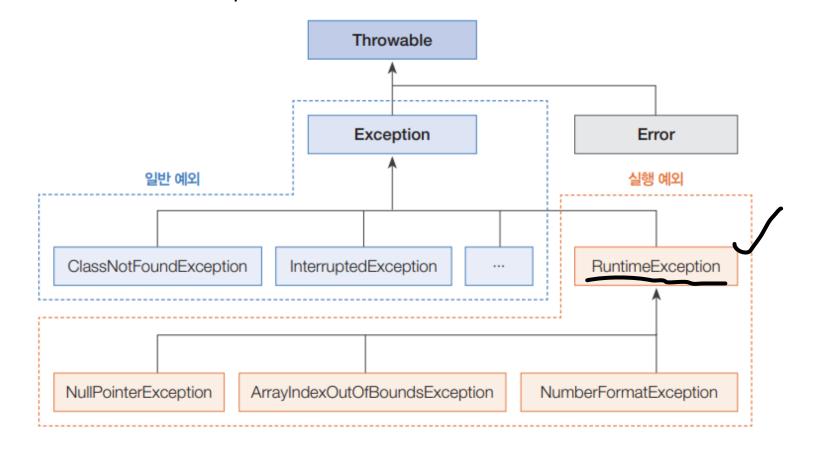
이 예외: 잘못된 사용 또는 코딩으로 인한 오류

에러와 달리 예외 처리를 통해 계속 실행 상태를 유지할 수 있음

일반 예외(Exception): 컴파일러가 예외 처리 코드 여부를 검사하는 예외

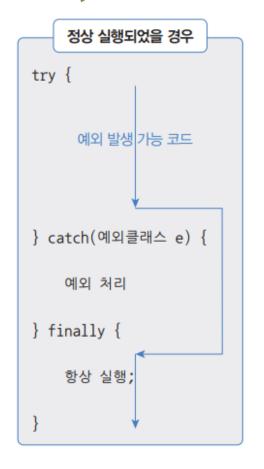
실행 예외(Runtime Exception): 컴파일러가 예외 처리 코드 여부를 검사하지 않는 예외 🗸

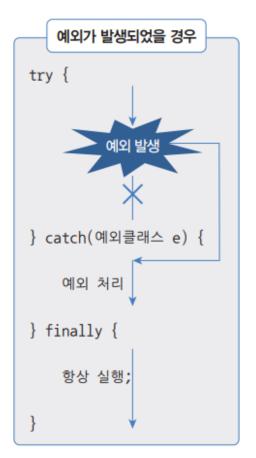




## 🧿 예외 처리

- 예외 발생 시 프로그램의 갑작스러운 종료를 막고 정상 실행을 유지할 수 있게 처리하는 코드
- 예외 처리 코드는 try-catch-finally 블록으로 구성
- trycatch-finally 블록은 생성자 내부와 메소드 내부에서 작성





[프로그램 시작]

## ExceptionHandlingExample1.java

```
package ch11.sec02.exam01;
public class ExceptionHandlingExample1 {
 public static void printLength(String data) {
   int result = data.length(); // data가 null일 경우 NullPointerException 발생
   System.out.println("문자 수: " + result);
 public static void main(String[] args) {
   System.out.println("[프로그램 시작]\n");
   printLength("ThisIsJava");
   printLength(null); // 매개값으로 null 대입
   System.out.println("[프로그램 종료]");
```

문자 수: 10 Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.length()" because "data" is null at ch11.sec02.exam01.ExceptionHandlingExample1.printLength(ExceptionHandlingExample1.java:5) at ch11.sec02.exam01.ExceptionHandlingExample1.main(ExceptionHandlingExample1.java:12)

stack trace 함수 호출 정보들

## ExceptionHandlingExample2.java

```
package ch11.sec02.exam01;
public class ExceptionHandlingExample2 {
 public static void printLength(String data) {
   try {
     int result = data.length();
     System.out.println("문자 수: " + result);
   } catch(NullPointerException e) {
     System.out.println(e.getMessage()); //①
     //System.out.println(e.toString()); //2
     //e.printStackTrace(); //③
   } finally {
     System.out.println("[마무리 실행]\n");
 public static void main(String[] args) {
   System.out.println("[프로그램 시작]\n");
   printLength("ThisIsJava");
   printLength(null);
   System.out.println("[프로그램 종료]");
```

catch절에서 출력시에는 System.err.println(형식이 좋다. 정상적인 메세지와 에러와 구분하는 것이 좋다. 에러 메세지는 후에 로그 파일로 기록하는 것이 대부분인데 이것이 구분하는 이유

[프로그램 시작]
문자 수: 10
[마무리 실행]

Cannot invoke "String.length()" because "data" is null
[마무리 실행]

[프로그램 종료]

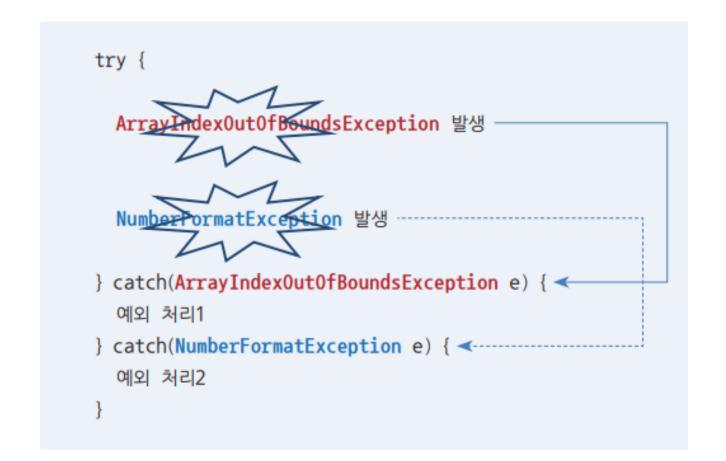
## ExceptionHandlingExample.java

```
package ch11.sec02.exam02;
public class ExceptionHandlingExample {
 public static void main(String[] args) {
   try {
                                     파일로 존재하는 클래스를 메모리에 로드 시킨는 함수
    Class.forName("java.lang.String");
    System.out.println("java.lang.String 클래스가 존재합니다.");
   } catch(ClassNotFoundException e) {
     e.printStackTrace();
                                               강제로 예외처리해야하는 에러
   System.out.println();
   try {
    Class.forName("java.lang.String2");
    System.out.println("java.lang.String2 클래스가 존재합니다.");
   } catch(ClassNotFoundException e) {
     e.printStackTrace();
```

```
java.lang.String 클래스가 존재합니다.
java.lang.ClassNotFoundException: java.lang.String2
               at java.base/jdk.internal.loader.BuiltinClassLoader.loadClass(BuiltinClassLoader.java:641)
java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(ClassLoaders.java:188)
              at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:520)
               at java.base/java.lang.Class.forName0(Native Method)
               at java.base/java.lang.Class.forName(Class.java:375)
               at ch11.sec02.exam02.ExceptionHandlingExample.main(ExceptionHandlingExample.java:16)
```

## ♡ 다중 catch로 예외 처리하기

- o catch 블록의 예외 클래스는 try 블록에서 발생된 예외의 종류를 말함.
- o 해당 타입의 예외가 발생하면 catch 블록이 선택되어 실행
- o catch 블록이 여러 개라도 catch 블록은 단 하나만 실행됨



## ExceptionHandlingExample.java

```
package ch11.sec03.exam01;
public class ExceptionHandlingExample {
 public static void main(String[] args) {
   String[] array = {"100", "100"};  \(\nabla\)
   for(int i=0; i<=array.length; i++) {
     try {
      int value = Integer.parseInt(array[i]);
      System.out.println("array[" + i + "]: " + value);
     } catch(ArrayIndexOutOfBoundsException e) {
      System.out.println("배열 인덱스가 추과됨: " + e.getMessage());
     } catch(NumberFormatException_e) {
      System.out.printtm( 숫자로 변환할 수 없음: " + e.getMessage());
         array[0]: 100
         숫자로 변환할 수 없음: For input string: "100"
         배열 인덱스가 초과됨: Index 2 out of bounds for length 2
```

## ♡ 다중 catch로 예외 처리하기

- 처리해야 할 예외 클래스들이 상속 관계에 있을 때
  - 하위 클래스 catch 블록을 먼저 작성하고 상위 클래스 catch 블록을 나중에 작성해야 함

```
try {
            OutOfBoundsException 발생
} catch(Exception e) { 🛬
  예외 처리1
 catch(ArrayIndexOut) (BoundsException e) {
  예외 처리2
```

## ExceptionHandlingExample.java

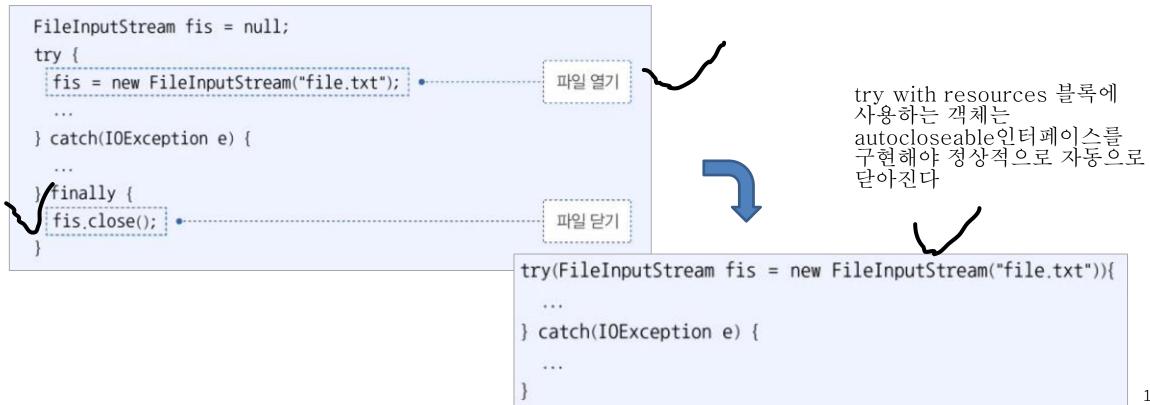
```
package ch11.sec03.exam02;
public class ExceptionHandlingExample {
 public static void main(String[] args) {
   String[] array = {"100", "100"};
   for(int i=0; i<=array.length; i++) {</pre>
    try {
      int value = Integer.parseInt(array[i]);
      System.out.println("array[" + i + "]: " + value);
     } catch(ArrayIndexOutOfBoundsException e) {
      System.out.println("배열 인덱스가 초과됨: " + e.getMessage());
     } catch(Exception e) {
      System.out.println("실행에 문제가 있습니다.");
         array[0]: 100
         실행에 문제가 있습니다.
         배열 인덱스가 초과됨: Index 2 out of bounds for length 2
```

## ExceptionHandlingExample.java

```
package ch11.sec03.exam03;
public class ExceptionHandlingExample {
 public static void main(String[] args) {
   String[] array = {"100", "100", null, "200"};
   for(int i=0; i<=array.length; i++) {</pre>
    try {
      int value = Integer.parseInt(array[i]);
      System.out.println("array[" + i + "]: " + value);
    } catch(ArrayIndexOutOfBoundsException e) {
      System.out.println("배열 인덱스가 초과됨: " + e.getMessage());
     } catch(NullPointerException ¦ NumberFormatException e) { // 2가지 예외를 동일학 처리
      System.out.println("데이터에 문제가 있음: " + e.getMessage());
         array[0]: 100
         데이터에 문제가 있음: For input string: "100"
         데이터에 문제가 있음: Cannot parse null string
         array[3]: 200
         배열 인덱스가 초과됨: Index 4 out of bounds for length 4
```

## 리소스 자동 닫기

- 리소스 : 데이터를 제공하는 객체
- 리소스는 사용하기 위해 열어야(open) 하며, 사용이 끝난 다음에는 닫아야(close) 함
- 리소스를 사용하다가 예외가 발생될 경우에도 안전하게 닫는 것이 중요
- try-with-resources 블록을 사용하면 예외 발생 여부와 상관없이 리소스를 자동으로 닫아줌



### 4 리소스 자동 닫기

### ✓ AutoCloseable 인터페이스

- o AutoCloseable 인터페이스를 구현한 객체만 리소스 자동 닫기 가능
  - void close() thorws Exception 추상 메서드 구현

```
public class FileInputStream implements AutoCloseable {
    ...
    @Override
    public void close() throws Exception { ... }
```

### 💟 복수개의 리소스 처리하기

o try() 괄호 안에 ';'로 구분하여 작성

```
try(
   FileInputStream fis1 = new FileInputStream("file1.txt");
   FileInputStream fis2 = new FileInputStream("file2.txt")
) {
   ...
} catch(IOException e) {
   ...
}
```

#### 💟 외부 리소스 변수 사용하기

- Java8 이전 버전은 try() 안에서 리소스 변수를 반드시 선언
- Java9 이후 부터 외부 리소스 변수 사용 가능

```
FileInputStream fis1 = new FileInputStream("file1.txt");
FileInputStream fis2 = new FileInputStream("file2.txt");
try(fis1; fis2) {
    ...
} catch(IOException e) {
    ...
}
```

# MyResource.java

```
직접 auto closable 활용한 코드
package ch11.sec04;
public class MyResource implements AutoCloseable {
 private String name;
 public MyResource(String name) {
   this.name = name;
   System.out.println("[MyResource(" + name + ") 열기]");
 public String read1() {
   System.out.println("[MyResource(" + name + ") 읽기]");
   return "100";
 public String read2() {
   System.out.println("[MyResource(" + name + ") 읽기]");
   return "abc";
 @Override
 public void close() throws Exception {
   System.out.println("[MyResource(" + name + ") 닫기]");
```

## TryWithResourceExample.java

```
package ch11.sec04;
public class TryWithResourceExample {
 public static void main(String[] args) {
   try (MyResource res = new MyResource("A")) {
     String data = res.read1();
     int value = Integer.parseInt(data);
   } catch(Exception e) {
     System.out.println("예외 처리: " + e.getMessage());
   System.out.println();
   try (MyResource res = new MyResource("A"))
     String data = res.read2();
     //NumberFormatException 발생
     int value = Integer.parseInt(data);
   } catch(Exception e) {
     System.out.println("예외 처리: " + e.getMessage());
   System.out.println();
```

```
[MyResource(A) 열기]
[MyResource(A) 읽기]
[MyResource(A) 닫기]
[MyResource(A) 열기]
[MyResource(A) 읽기]
[MyResource(A) 닫기]
예외 처리: For input string: "abc"
```

## TryWithResourceExample.java

```
/*try (
 MyResource res1 = new MyResource("A");
 MyResource res2 = new MyResource("B")
 String data1 = res1.read1();
 String data2 = res2.read1();
} catch(Exception e) {
 System.out.println("예외 처리: " + e.getMessage());
}*/
                                               [MyResource(A) 열기]
MyResource res1 = new MyResource("A"); )
                                               [MyResource(B) 열기]
                                               [MyResource(A) 읽기]
MyResource res2 = new MyResource("B");
                                               [MyResource(B) 읽기]
try (res1; res2) {
                                               [MyResource(B) 닫기]
 String data1 = res1.read1();
                                               [MyResource(A) 닫기]
 String data2 = res2.read1();
} catch(Exception e) {
 System.out.println("예외 처리: " + e.getMessage());
```

예외를 처리하는 주체에 대한 고민을 많이 해야 한다. 실제 예외가 발생한 곳에서 처리하면 상위 스코프에서는 예외가 발생한지도 모르고 다음 코드를 실행할 수도 있다. 이때 예외처리 주체를 '던지는' 연산을 활용하여 처리를 맡는 주체를 다르게 하자

## 예외 떠넘기기

- <u>메소드 내부에서 예외 발생 시 th</u>rows <u>키</u>워드 이용해 메소드를 호출한 곳으로 예외 떠넘기기
- throws는 메소드 선언부 끝에 작성
- 떠넘길 예외 클래스를 쉼표로 구분해서 나열

```
리턴타입 메소드명(매개변수, \cdots) throws 예외클래스1, 예외클래스2, \cdots {
public void method1() {
 try {
   method2(); //method2() 호출
 } catch(ClassNotFoundException e) {
                                                        호출한 곳에서 예외 처리
   System.out.println("예외 처리: " + e.getMessage());
public void method2() throws ClassNotFoundException {
 Class.forName("java.lang.String2");
```

## ThrowsExample1.java

```
package ch11.sec05;
public class ThrowsExample1 {
 public static void main(String[] args) {
   try {
     findClass();
   } catch(ClassNotFoundException e) {
     System.out.println("예외 처리: " + e.toString());
 public static void findClass() throws ClassNotFoundException {
   Class.forName("java.lang.String2");
```

예외 처리: java.lang.ClassNotFoundException: java.lang.String2

## 예외 떠넘기기

○ 나열할 예외 클래스가 많으면 throws Exception 또는 throws Throwable로 모든 예외 떠넘기기

```
리턴타입 메소드명(매개변수,…) throws Exception {
```

```
제일 상위 메소드가 던져버리면
JVM이 받는다
public static void main(String[] args) throws Exception {
```

## ThrowsExample2.java

```
package ch11.sec05;

public class ThrowsExample2 {
  public static void main(String[] args) throws Exception {
    findClass();
  }

public static void findClass() throws ClassNotFoundException {
    Class.forName("java.lang.String2");
  }
}
```

```
Exception in thread "main" java.lang.ClassNotFoundException: java.lang.String2
at java.base/jdk.internal.loader.BuiltinClassLoader.loadClass(BuiltinClassLoader.java:641)
at java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(ClassLoaders.java:188)
at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:520)
at java.base/java.lang.Class.forName0(Native Method)
at java.base/java.lang.Class.forName(Class.java:375)
at ch11.sec05.ThrowsExample2.findClass(ThrowsExample2.java:9)
at ch11.sec05.ThrowsExample2.main(ThrowsExample2.java:5)
```

# ♡ 사용자 정의 예외 ✓

- 표준 라이브러리에는 없어 직접 정의하는 예외 클래스
- 일반 예외는 Exception의 자식 클래스로 선언. 실행 예외는 RuntimeException의 자식 클래스로 선언

```
public class XXXException extends [ Exception | RuntimeException ] {
  public XXXException() {
                                       • 기본 생성자
 public XXXException(String message) {
    super(message);
```

## InsufficientException.java

```
package ch11.sec06;
public class InsufficientException extends Exception {
 public InsufficientException() {
 public InsufficientException(String message) {
   super(message);
```

## 💟 예외 발생시키기

- o throw 키워드와 함께 예외 객체를 제공해 사용자 정의 예외를 직접 코드에서 발생시킬 수 있음
- 예외의 원인에 해당하는 메시지를 제공하려면 생성자 매개값으로 전달

```
throw new Exception()
throw new RuntimeException();
throw new InsufficientException();
```

```
throw new Exception("예외메시지")
throw new RuntimeException("예외메시지);
throw new InsufficientException("예외메
시지");
```

```
void method() {

try {

throw new Exception("예외메시지");

...
} catch(Exception e) {

String message = e.getMessage();
}
}
```

```
void method() throws Exception {
...
throw new Exception("예외메시지");
...
}
```

## Account.java

```
package ch11.sec06;
public class Account {
 private long balance;
 public Account() { }
 public long getBalance() {
   return balance;
 public void deposit(int money) {
   balance += money;
                                                              에러 구체화
 public void withdraw(int money) throws InsufficientException {
   if(balance < money) {</pre>
     throw new InsufficientException("잔고 부족: "+(money-balance)+" 모자람");
   balance -= money;
```

## AccountExample.java

```
package ch11.sec06;
public class AccountExample {
 public static void main(String[] args) {
   Account account = new Account();
   //예금하기
   account.deposit(10000);
   System.out.println("예금액: " + account.getBalance());
   //출금하기
   try {
     account.withdraw(30000);
   } catch(InsufficientException e) {
     String message = e.getMessage();
     System.out.println(message);
```

정리

예외 종류 이해 exception/runtime exception 차이 try catch finally 예외 던지기 떠넘기기 중요!! 경험치 필요 사용자 정의 예외

예금액: 10000

잔고 부족: 20000 모자람