

자바 프로그래밍

제14장 오류 처리하기

학습 내용

학습목차

- 01 디버깅
- 02 예외 처리
- 03 예외의 종류
- 04 예외와 메소드
 - LAB 예외 처리하기
- 05 예외 생성 하기
- 06 예외 처리의 장점
 - LAB 예외 처리하기
- 07 단언
- 08 로깅

자바에서 오류가 발생하면
프로그래밍하기 힘들어요.
좋은 방법이 있나요?

네, 예외 처리를 사용해
보세요. 우아하게 오류를
처리할 수 있습니다.



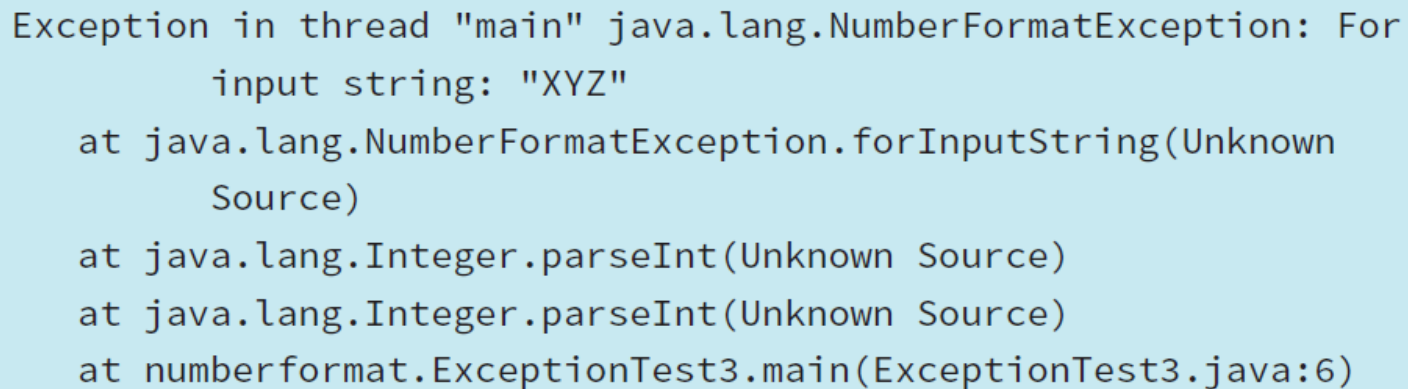
디버깅

- 우리가 사는 세상은 완벽하지 않다!
 - 사용자가 잘못된 데이터 입력하는 경우
 - 열고자 하는 파일이 컴퓨터에 없는 경우
 - 인터넷이 다운되는 경우
 - 프로그램에 버그가 있는 경우
 - ...



오류 메시지

- 프로그램 실행시 오류가 발생한 경우, 오류 메시지에서 많은 내용을 알 수 있다.

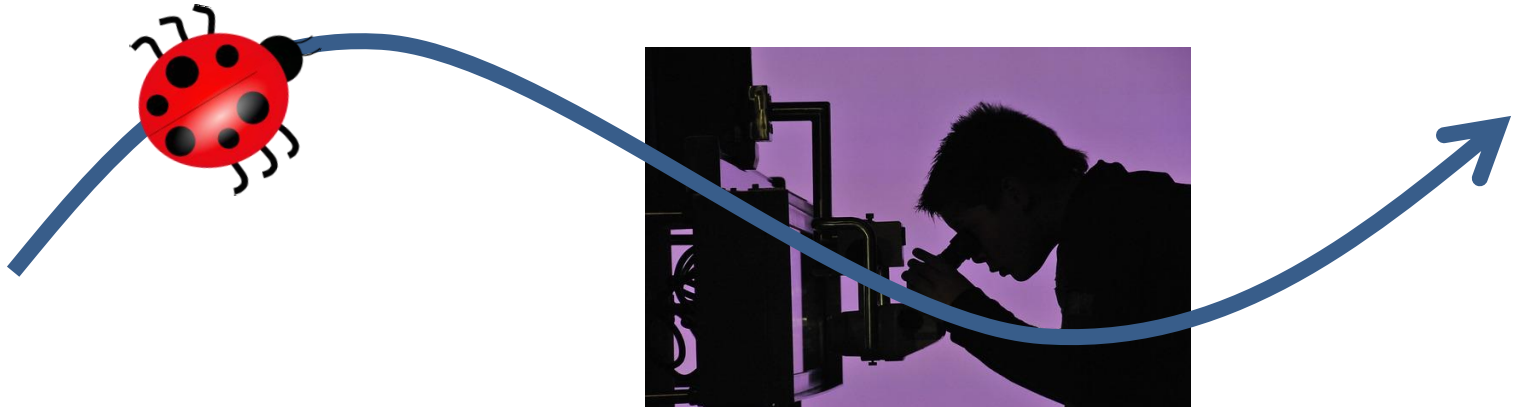
A screenshot of a Java exception message displayed in a window. The window has a standard macOS-style title bar with red, yellow, and green buttons. The text inside the window is as follows:

```
Exception in thread "main" java.lang.NumberFormatException: For  
    input string: "XYZ"  
    at java.lang.NumberFormatException.forInputString(Unknown  
        Source)  
    at java.lang.Integer.parseInt(Unknown Source)  
    at java.lang.Integer.parseInt(Unknown Source)  
    at numberformat.ExceptionTest3.main(ExceptionTest3.java:6)
```

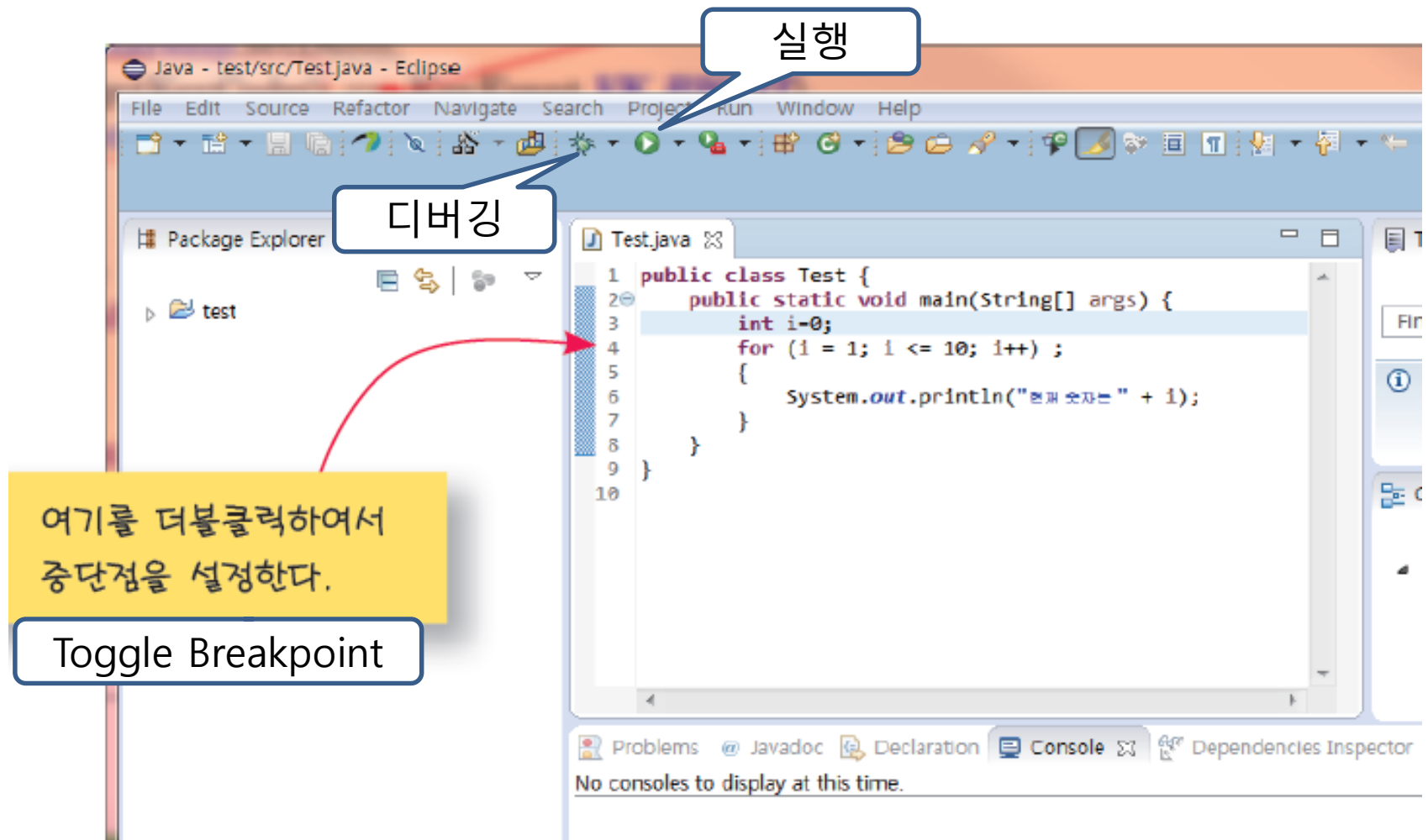
- 오류 메시지만 보고 오류 원인을 알 수 없는 경우, 디버거를 이용하여 디버깅할 수 있다.

디버깅(debugging)

- 디버거(debugger)를 사용하면 프로그램에서 쉽게 오류를 감지하고 진단할 수 있다.
 - 중단점(breakpoint)을 설정하여 프로그램의 실행을 잠시 멈추게 할 수 있다.
 - 변수에 저장된 값을 살펴볼 수 있다.
 - 문장 단위로 실행할 수 있다.



이클립스에서 디버깅



Debug perspective

The screenshot shows the Eclipse IDE in the Debug perspective. The top toolbar includes icons for File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The 'Debug' tab is active, showing a tree view of the debug session. The 'Variables' window is open, displaying the state of the program's variables. The 'Test.java' editor shows the source code with a breakpoint at line 4. A yellow callout box with Korean text explains how to view variable values by hovering over the variable name.

Debug - test/src/Test.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Quick Access Java Debug

Debug

- test [Java Application]
 - Test at localhost:55381
 - Thread [main] (Suspended (breakpoint at line 4 in Test))
 - Test.main(String[]) line: 4

C:\Program Files\Java\jre1.8.0_51\bin\javaw.exe (2015. 8. 15. 오후 3:56:33)

(x)= Variables Breakpoints

Name	Value
args	String[0] (id=16)
i	0

Test.java

```
1 public class Test {  
2     public static void main(String[] args) {  
3         int i=0;  
4         for (i = 1; i <= 10; i++) ;  
5         {  
6             System.out.println("현재 수: " + i);  
7         }  
8     }  
9 }  
10
```

Outline

- Test
 - main(String[]) : void

Console

test [Java Application] C:\Program Files\Java\jre1.8.0_51\bin\javaw.exe (2015. 8. 15. 오후 3:56:33)

Writable Smart Insert 4 : 1

변수의 값을 보려면 변수 이름 위에 마우스 커서를 놓아도 되고 아니면 변수 창을 보아도 된다.

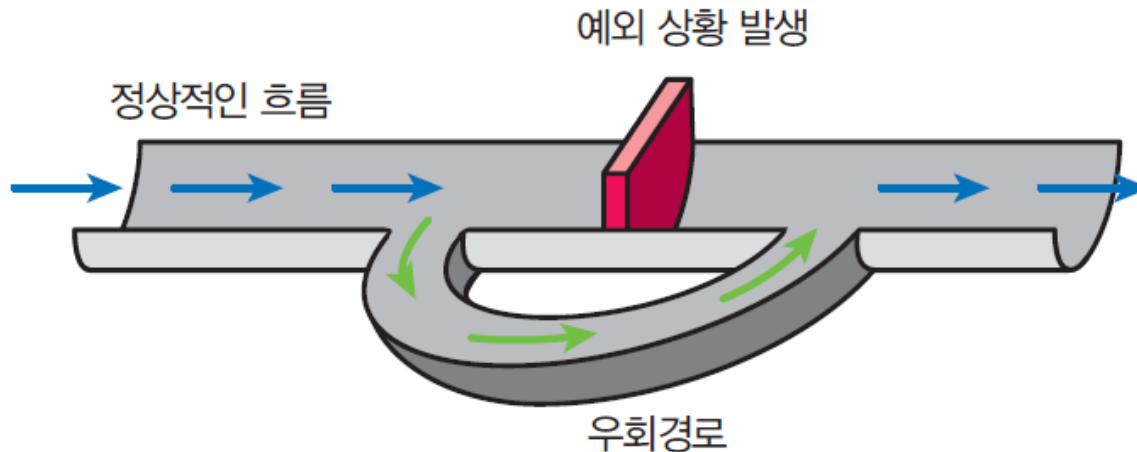
디버깅 명령어



	설명	방법
Step Into	한 문장씩 실행, 메소드를 만나면 메소드 안으로 진입	F6 또는  아이콘 클릭
Step Over	한 문장씩 실행, 메소드를 만나면 메소드 진입 안함	F5 또는  아이콘 클릭
Run to Line	지정된 문장까지 실행	Run->Run to Line 메뉴 또는 Ctrl+R
Resume	중단된 프로그램 다시 실행	F8 또는  아이콘 클릭
Terminate	프로그램 종료	Run->Terminate

예외 처리

- 프로그램 실행 중에 오류가 발생하는 경우
 - 대개 프로그램이 즉시 종료된다.
 - 오류를 사용자에게 알려 주고 모든 데이터를 저장하게 한 후에 사용자가 우아하게(gracefully) 프로그램을 종료할 수 있도록 하는 것이 바람직하다.



예외(exception)

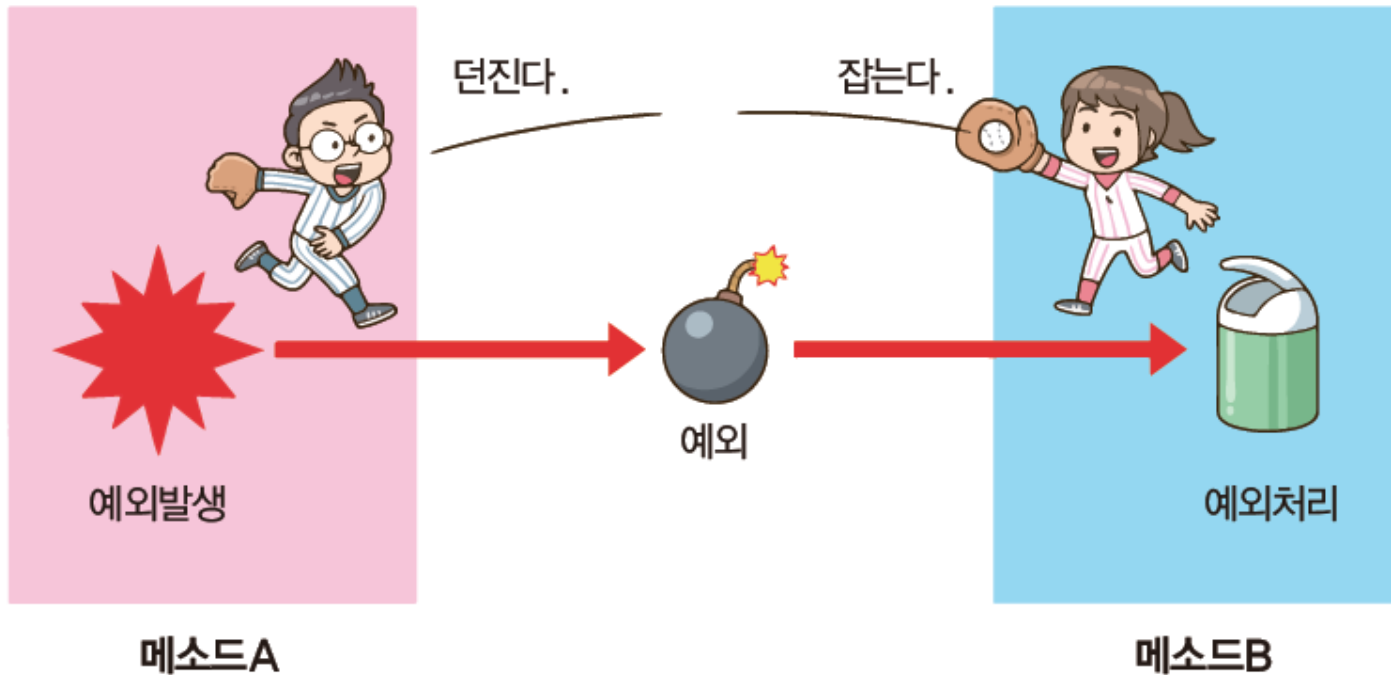
- 예외(exception)란 프로그램 실행 중에 발생하는 비정상적인 사건을 의미
 - 일어나서는 안되는 잘못된 상황(invalid 상황)
 - 일어날 수 있지만 예외적인 상황(unusual 상황)
- 자바는 실행 중 예외가 일어나는 상황을 미리 정의해 둬
 - 0으로 나누는 정수 나눗셈을 한 경우
 - 배열의 인덱스가 범위를 벗어나는 경우
 - 입력 받으려고 지정한 파일이 존재하지 않는 경우, ...
- 자바는 예외도 객체로 표현함
 - 예외를 객체로 표현한다는 것은 예외 클래스를 제공한다는 의미
 - 프로그래머가 예외 클래스를 정의할 수도 있음

예외 처리

- 메소드 실행 중에 예외가 발생한 경우
 - 프로그램의 정상 실행 흐름이 중단되며, 대부분 프로그램이 종료된다.
 - 이 때 자바 런타임 시스템으로 예외 객체를 넘긴다.
- 프로그램을 종료하는 대신에 프로그램에서 예외 상황을 해결하고 프로그램을 계속 실행하는 것이 가능할까?
 - 답: 가능하다. 예외 처리(exception handling)를 하면 된다.
- 자바는 프로그램에서 직접 예외를 다룰 수 있다.
 - 프로그래머가 예외를 만들어 발생시킬 수도 있고,
 - 발생한 예외를 처리할 수도 있다.

예외 발생과 예외 처리

- 실행시간에 오류가 발생하면 예외가 생성된다.
 - 프로그램에서 예외를 잡아 처리할 수 있다.



예외 발생: 예외 처리하지 않은 예

```
package exception1;
```

```
public class BadIndex {
```

```
    public static void main(String[] args) {
```

```
        int[] array = new int[10];
```

```
        for (int i = 0; i <= 10; i++)
```

```
            array[i] = i;
```

```
        System.out.println("과연 이 문장이 실행될까요?"); // 실행 안됨
```

```
    }
```

```
}
```

i가 10이 되면
ArrayIndexOutOfBoundsException
타입의 예외가 발생하고, 즉시 프로
그램이 종료됨

실행결과

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10  
at exception1.BadIndex.main(BadIndex.java:8)
```

예외 처리기 : try/catch 블록

try 블록은 예외가
발생할 수 있는 위험
한 코드

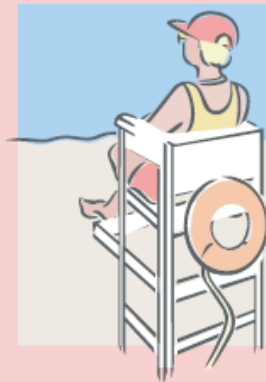
```
try {
```



```
}
```

catch 블록은 예외를
처리하는 코드

```
catch { 물에빠짐 예외 ) {
```



```
}
```

try 블록에서
오류가 발생하면
처리합니다.

try/catch 블록

형식

```
try {  
    // 예외가 발생할 수 있는 코드  
}  
catch ( 예외타입 참조변수 ) {  
    // 예외를 처리하는 코드  
}
```

try 블록 실행 중에 예외가 발생하면 try 블록의 나머지 부분은 실행하지 않음

try 블록에서 발생한 예외의 타입이 일치하면 예외를 잡아 예외 처리 코드를 실행하고, 일치하지 않으면 잡지 않음

예외 발생 : 예외 처리한 예

```
public class BadIndex2 {  
    public static void main(String[] args) {  
        int[] array = new int[10];  
        try {  
            for (int i = 0; i <= 10; i++)  
                array[i] = i;  
        } catch (ArrayIndexOutOfBoundsException e) {  
            System.out.println("배열의 인덱스가 잘못됨");  
        }  
        System.out.println("과연 이 문장이 실행될까요?"); // 실행됨  
    }  
}
```

예외 객체를 가리킬
참조 변수 e

실행결과

배열의 인덱스가 잘못됨
과연 이 문장이 실행될까요?

finally 블록

형식

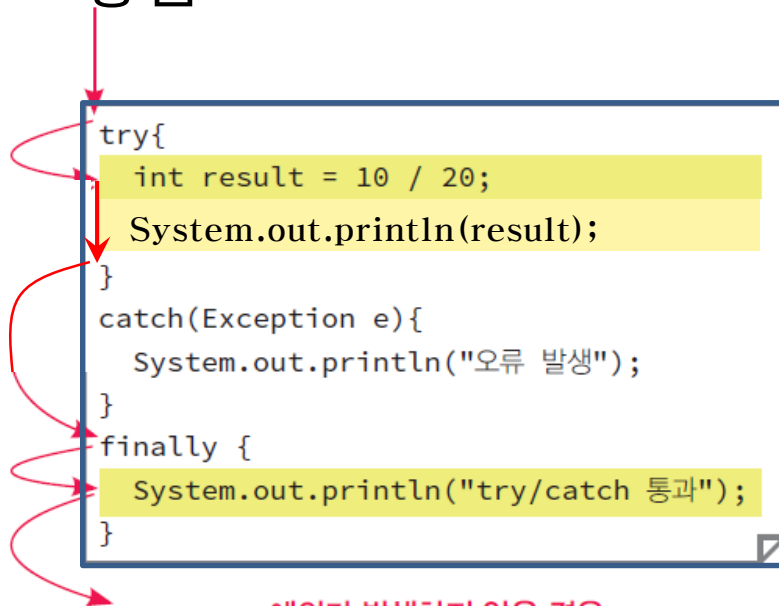
```
try {  
    // 예외가 발생할 수 있는 코드  
} catch (예외타입 참조변수) {  
    // 예외를 처리하는 코드  
}
```

```
finally {  
    // try-catch 블록이 끝나면 무조건 실행되는 코드  
}
```

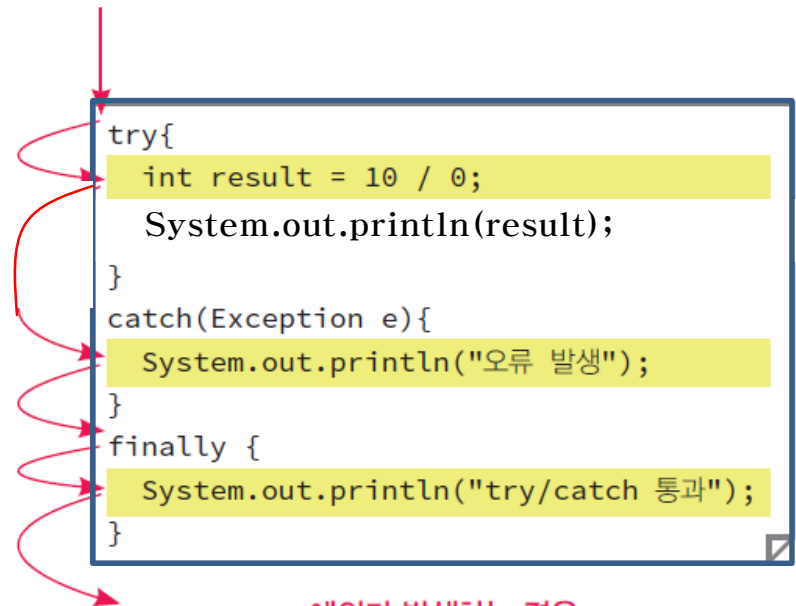
생략이 가능하다.

예외 처리 실행 흐름

- try 블록 실행 중에 예외가 발생하면 try 블록의 나머지 부분은 실행하지 않음
- 예외 발생 여부, 예외 처리 여부와 무관하게 try/catch 블록을 마치고 항상 실행되어야 하는 코드는 finally 블록에 넣음



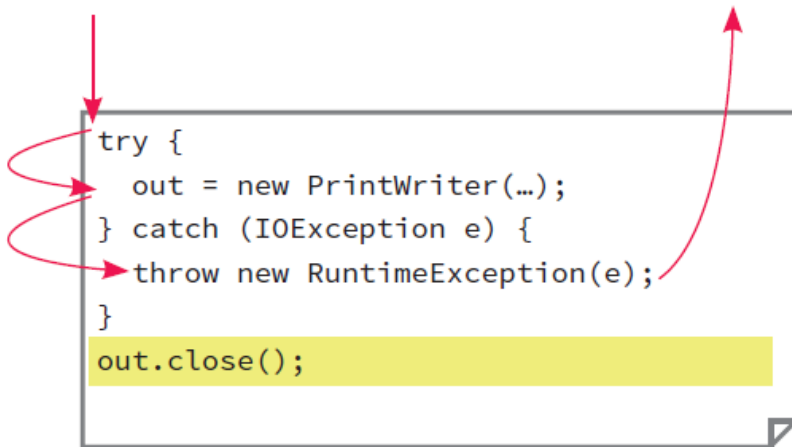
예외가 발생하지 않은 경우



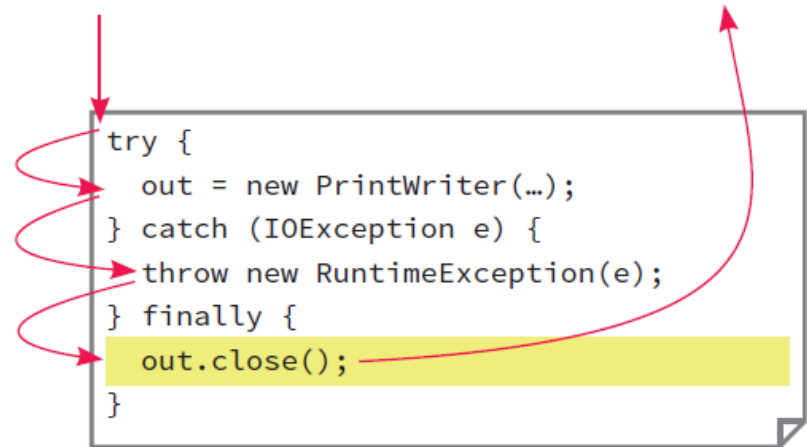
예외가 발생하는 경우

finally 블록 사용 예

- 파일과 같은 자원을 반납하는 코드는 finally 블록에 넣는 것이 좋다.
 - 예: 예외 발생하여 프로그램이 중단되더라도 반드시 PrintWriter 객체를 닫으려면 finally 블록 내에 닫는 문장을 넣으면 됨



예외가 발생하면 자원이 반납되지 않을 수 있다.



예외가 발생하더라도 확실하게 자원이 반납된다.

여러 개의 catch 블록

```
int[] list = new int[size];
PrintWriter out = null;

try {
    out = new PrintWriter("outfile.txt");
    for (int i = 0; i < 10; i++)
        out.println(list[i]);
} catch (ArrayIndexOutOfBoundsException e) {
    System.err.println(e);
} catch (IOException e) {
    System.err.println(e);
} finally {
    if (out != null)
        out.close();
}
```

try 블록에서 여러 타입의
예외 발생 가능

배열 인덱스 예외
발생시 실행

입출력 예외 발생시
실행

항상 실행되어 자원을
반납

여러 개의 catch 블록 – 예외발생 예1

```
int[] list = new int[size];
PrintWriter out = null;

try {
    out = new PrintWriter("outfile.txt");
    for (int i = 0; i < 10; i++)
        out.println(list[i]); // 배열 크기 size가 9라면
} catch (ArrayIndexOutOfBoundsException e) {
    System.err.println(e);
} catch (IOException e) {
    System.err.println(e);
} finally {
    if (out != null)
        out.close();
}
```

배열 인덱스 예외
발생시 실행

항상 실행되어 자원을 반납

java.lang.ArrayIndexOutOfBoundsException: 9

여러 개의 catch 블록 – 예외발생 예2

```
int[] list = new int[size];
PrintWriter out = null;

try {
    out = new PrintWriter("outfile.txt"); // outfile.txt가 읽기 전용이라면
    for (int i = 0; i < 10; i++)
        out.println(list[i]);
} catch (ArrayIndexOutOfBoundsException e) {
    System.err.println(e);
} catch (IOException e) {
    System.err.println(e);
} finally {
    if (out != null)
        out.close();
}
```

입출력 예외 발생시 실행

항상 실행되어 자원을 반납

java.io.FileNotFoundException: outfile.txt (엑세스가 거부되었습니다)

try-with-resources 문장

- try-with-resources 문장을 사용하면 문장의 끝에서 리소스들을 자동으로 close 한다.
 - 단, 리소스(자원)가 AutoCloseable 인터페이스 구현해야 함

형식

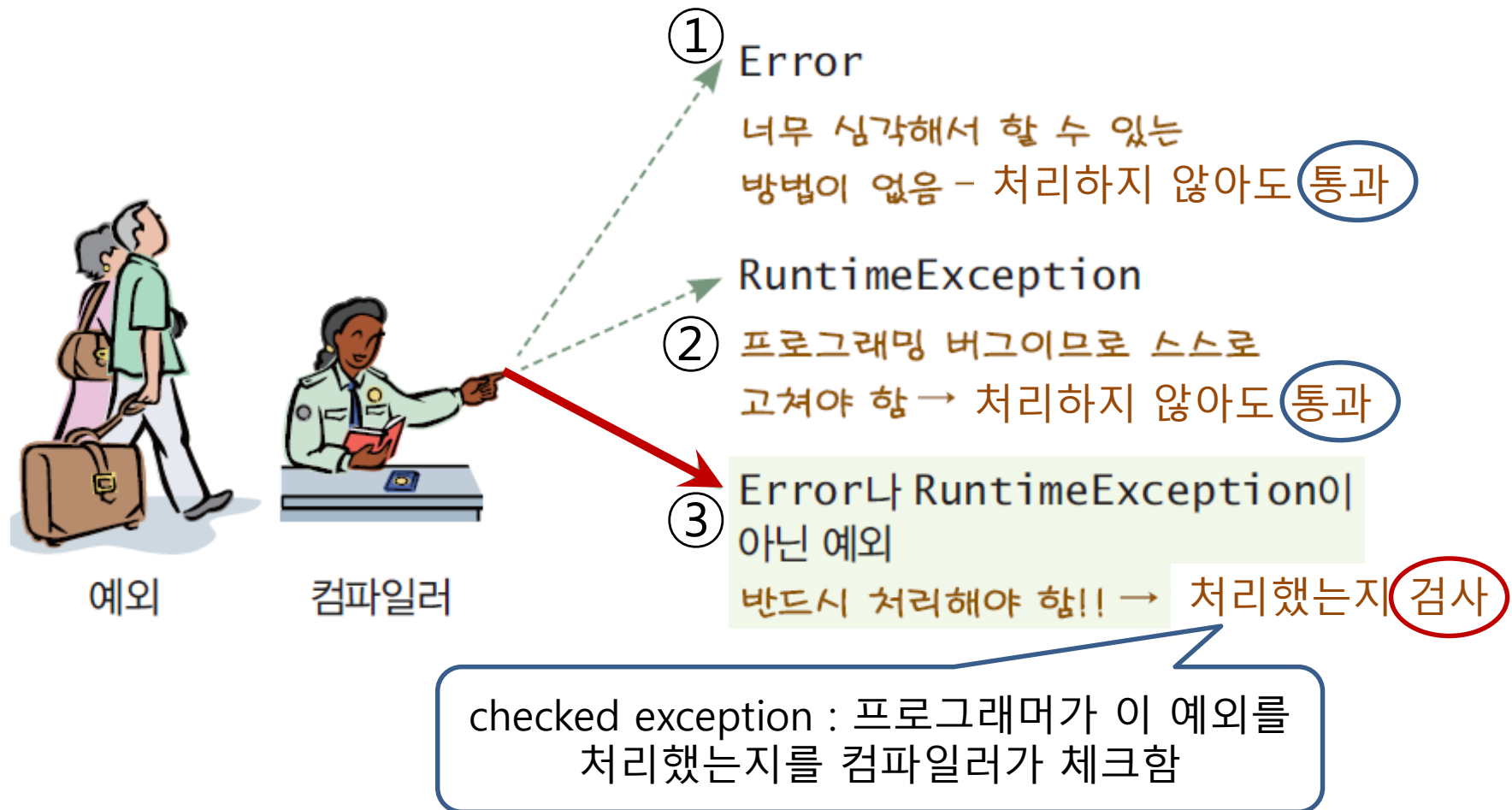
```
try (리소스자료형1 변수1 = 초기값1; 리소스자료형2 변수2 = 초기값2; ...) {  
    ...  
}
```

```
int[] list = new int[10];
```

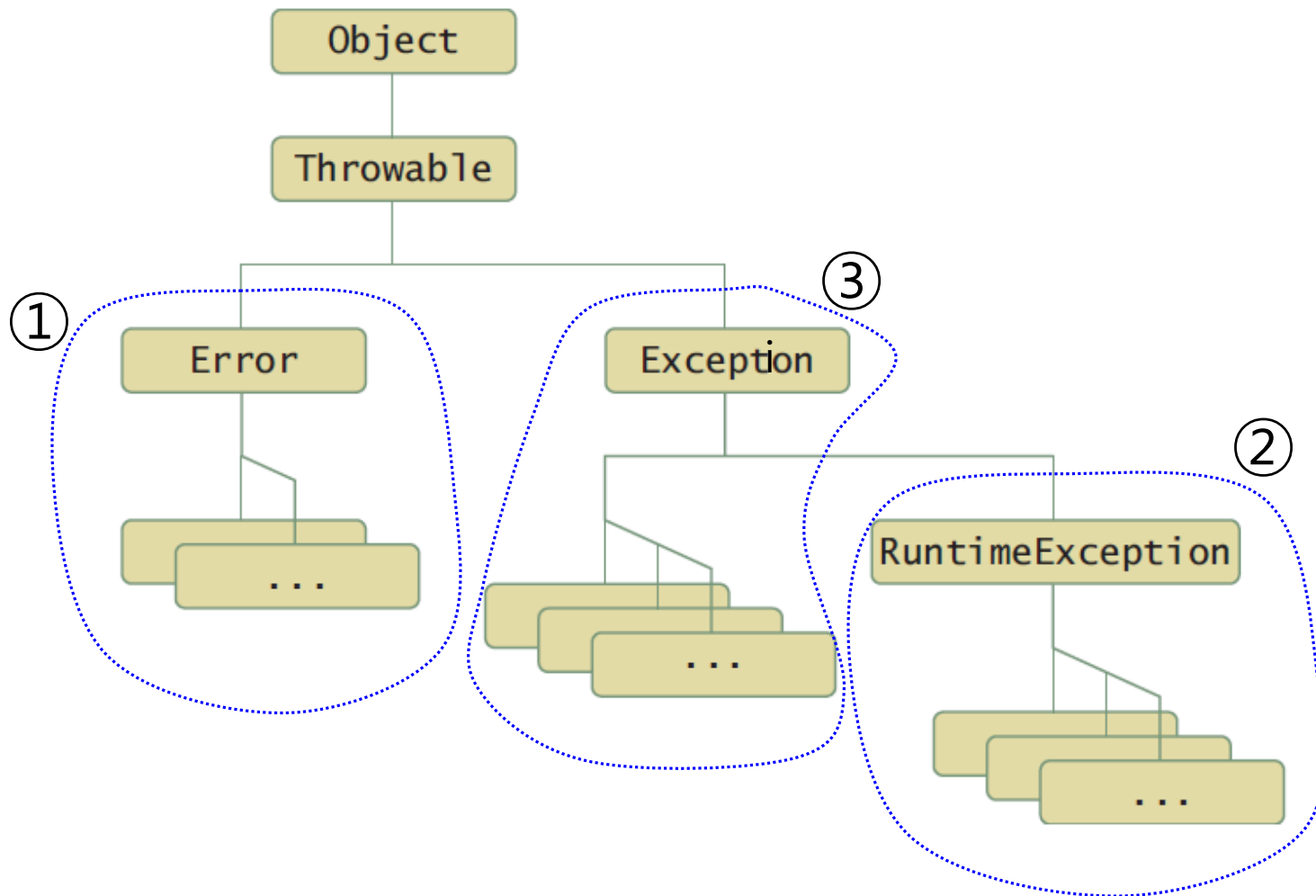
```
try (PrintWriter out = new PrintWriter("outfile.txt")) {  
    for (int i = 0; i < 10; i++) {  
        out.println(list[i]);  
    }  
}
```

PrintWriter 클래스는
AutoCloseable을 구현함

예외의 종류



예외 클래스의 계층구조



예외 클래스의 계층구조

① Error

- 발생하더라도 프로그램이 처리할 수 없는 치명적인 오류

② RuntimeException

- 프로그래밍 버그나 논리 오류로 인한 예외
- 예) `ArrayIndexOutOfBoundsException`
- 프로그래머가 예외를 처리해도 되지만, 처리했는지를 컴파일러가 체크하지는 않음

③ 기타 예외

- RuntimeException과 그 자손을 제외한 Exception의 자손
- 예) `FileNotFoundException` - 사용자가 실수로 잘못된 파일 이름을 입력하면 발생
- 회복할 수 있는 예외이므로 프로그래머가 반드시 처리해야 함
- **checked exception**이라고 부른다.

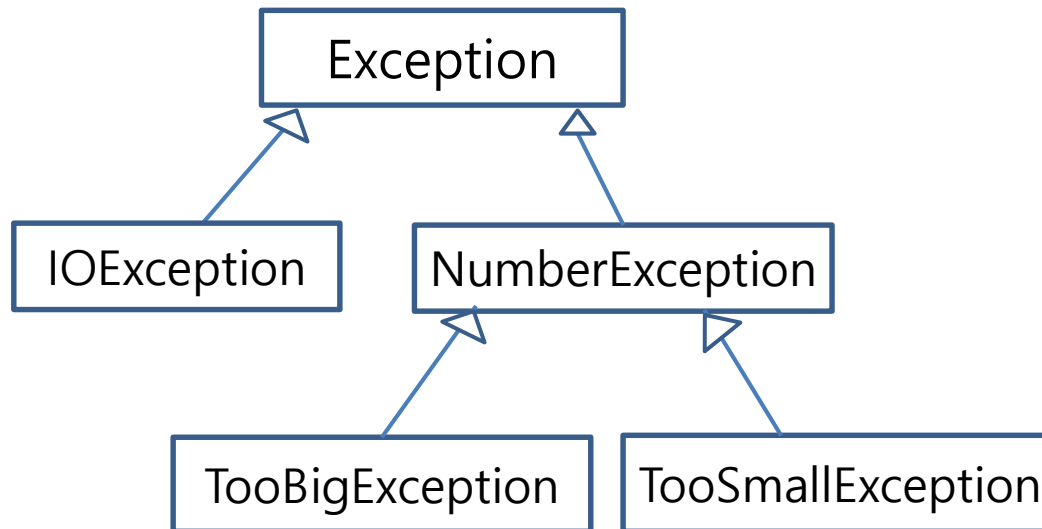
Unchecked Exception

- ① Error OutOfMemoryError 사용 가능한 메모리가 없는 경우
 NoClassDefFoundError 클래스 정의를 찾지 못한 경우

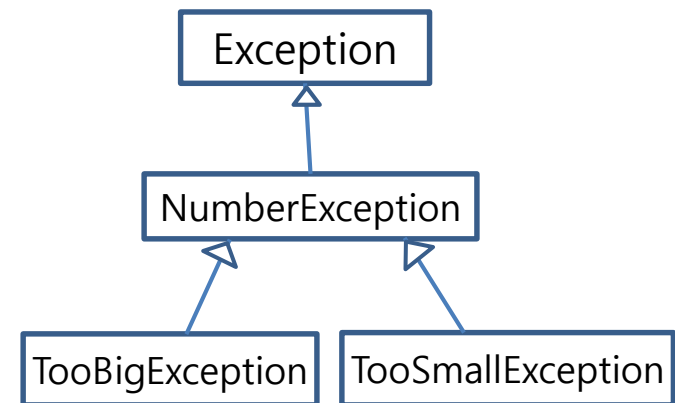
분류	예외	설명
② RuntimeException	ArithmeticException	어떤 수를 0으로 나눌 때 발생한다.
	NullPointerException	널 객체를 참조할 때 발생한다.
	ClassCastException	적절치 못하게 클래스를 형변환하는 경우
	NegativeArraySizeException	배열의 크기가 음수값인 경우
	ArrayIndexOutOfBoundsException	배열을 참조하는 인덱스가 잘못된 경우

다형성과 예외

- 예외도 객체로 취급
 - 다형성에 따라 상위 클래스 참조 변수는 하위 클래스 객체를 참조할 수 있다.
 - 이는 catch 블록에서 예외를 잡을 때도 해당된다.
- 예: 예외 클래스 계층도가 다음과 같을 때,
 - Exception 타입 변수로 하위 클래스 타입 예외를 잡을 수 있다.



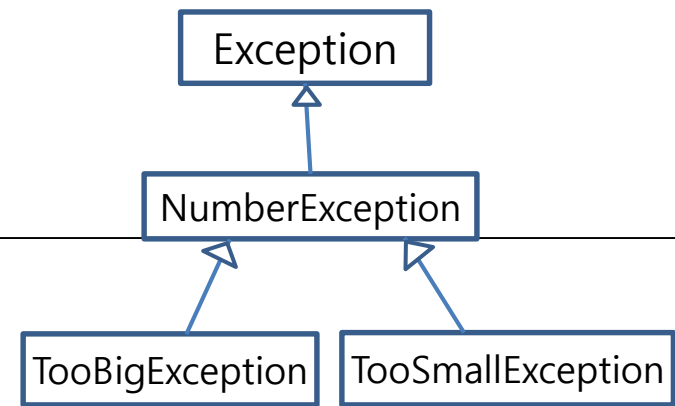
다형성과 예외



```
try {  
    getInput();    // 예외를 발생하는 메소드  
}  
catch (NumberFormatException e) {  
    // NumberException의 하위 클래스를 모두 잡을 수 있다.  
}
```

```
try {  
    getInput();    // 예외를 발생하는 메소드  
}  
catch (Exception e) {  
    // Exception의 하위 클래스를 모두 잡을 수 있으나 분간할 수 없다.  
}
```

다형성과 예외



```
try {  
    getInput();    // 예외를 발생하는 메소드  
}  
catch (TooSmallException e) {  
    // TooSmallException만 잡는다.  
}  
catch (NumberException e) {  
    // NumberException, TooBigException을 잡는다.  
}
```

```
try {  
    getInput();    // 예외를 발생하는 메소드  
}  
catch (NumberException e) {  
    // NumberException, TooBigException, TooSmallException을 잡는다.  
}  
catch (TooSmallException e) {  
    // 아무 것도 잡히지 않는다. → 에러  
}
```

예외와 메소드

- checked exception은 프로그래머가 반드시 처리해야 함
- 프로그램에서 예외를 처리하는 방법
 1. 예외를 잡아서 그 자리에서 처리하는 방법
 - **try/catch** 블록을 사용하여서 예외를 잡고 처리한다. (이제 까지 살펴본 방법이 이에 해당한다.)
 2. 메소드가 예외를 발생시킨다고 선언하는 방법
 - 메소드 헤더에 **throws** 절을 두어, 다른 메소드에게 예외 처리를 맡긴다.



메소드가 오류를 발생
할 수 있다고 말하면
오류가 처리 됩니다.



예제 : throws절 사용

컴파일 에러 : Unhandled exception type
FileNotFoundException

```
public void writeList()
{
    PrintWriter out = new PrintWriter("outfile.txt");
    for (int i = 0; i < 10; i++)
        out.println(i);
    out.close();
}
```



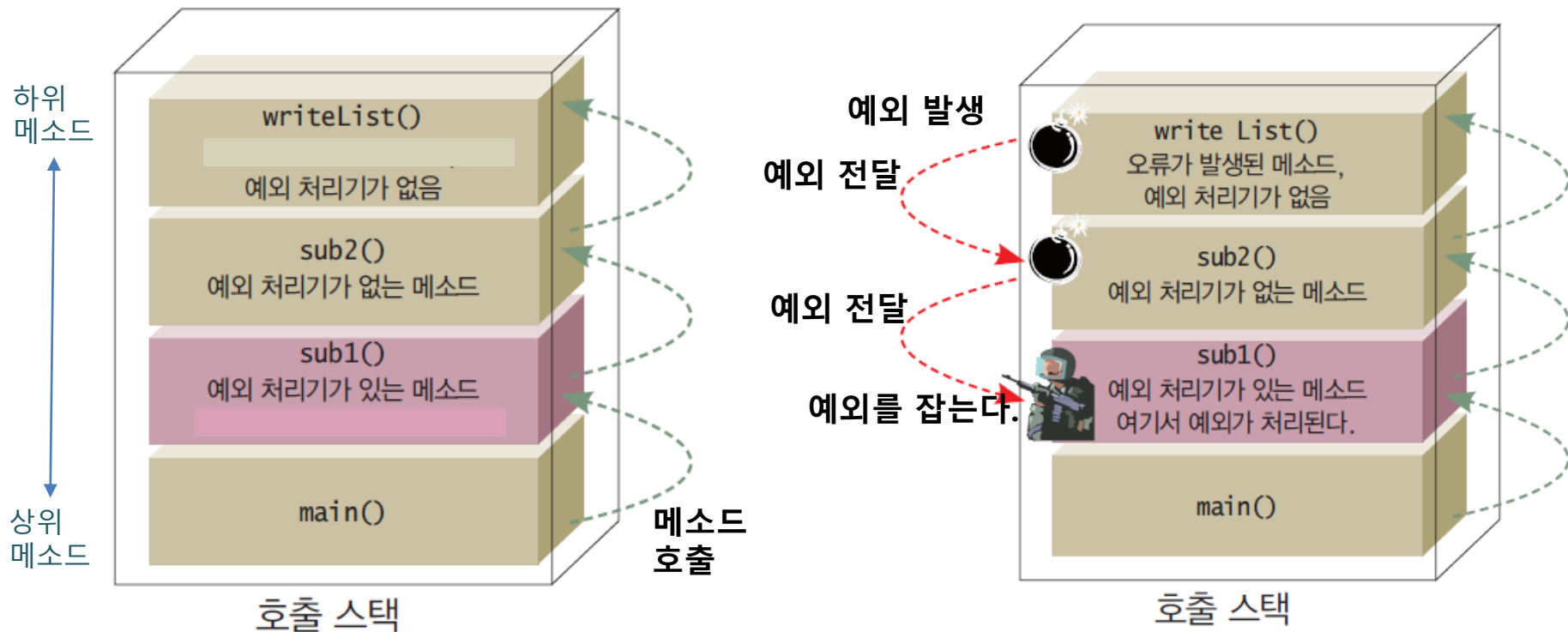
1. Surround with try/catch 또는
- 2. Add throws declaration**

```
public void writeList() throws FileNotFoundException
{
    PrintWriter out = new PrintWriter("outfile.txt");
    for (int i = 0; i < 10; i++)
        out.println(i);
    out.close();
}
```


예외 처리 과정

- 어떤 메소드에서 예외가 발생하면
 - 자바 런타임 시스템은 그 메소드 안에 예외 처리기(try/catch 블록)가 있는지를 살핀다. 예외 처리기가 있으면 예외를 처리한다.
 - 만약 그 자리에 예외 처리기가 없으면 호출 스택(call stack)에 있는 상위 메소드를 조사한다. 상위 메소드에 예외 처리기가 있으면 예외를 처리한다.
 - 전체 호출 스택을 뒤져도 예외 처리기를 찾지 못하면 프로그램을 종료시킨다.

예외 처리 과정



LAB: 예외 처리하기

```
public class Test {  
    public static void main(String[] args) {  
        System.out.println(readString());  
    }  
    public static String readString() {  
        byte[] buf = new byte[100];  
        System.out.println("문자열을 입력하시오:");  
        System.in.read(buf);  
        return new String(buf);  
    }  
}
```

컴파일 에러 :
Unhandled exception
type IOException

SOLUTION 1

```
import java.io.IOException;
public class Test {
    public static void main(String[] args) {
        try {
            System.out.println(readString());
        } catch (IOException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
    public static String readString() throws IOException {
        byte[] buf = new byte[100];
        System.out.println("문자열을 입력하시오:");
        System.in.read(buf);
        return new String(buf);
    }
}
```

SOLUTION 2

```
import java.io.IOException;

public class Test {
    public static void main(String[] args) throws IOException {
        System.out.println(readString());
    }
    public static String readString() throws IOException {
        byte[] buf = new byte[100];
        System.out.println("문자열을 입력하시오:");
        System.in.read(buf);
        return new String(buf);
    }
}
```

예: 예외를 발생하는 메소드

The screenshot shows the Oracle Java API documentation for the `read()` method of the `InputStream` class. The left sidebar lists various Java packages and classes, with `InputStream` highlighted. The main content area displays the method signature, description, parameters, returns, and throws sections. Two callouts are present: one pointing to the `throws IOException` in the signature, and another pointing to the `Throws:` section.

IOException 타입의 예외를 던질 수 있음을 표시한다.

```
public int read(byte[] b)
    throws IOException
```

Reads some number of bytes from the input stream and stores them into the buffer array `b`. The number of bytes actually read is returned as an integer. This method blocks until input data is available, end of file is detected, or an exception is thrown.

If the length of `b` is zero, then no bytes are read and `0` is returned; otherwise, there is an attempt to read at least one byte. If no byte is available because the stream is at the end of the file, the value `-1` is returned; otherwise, at least one byte is read and stored into `b`.

The first byte read is stored into element `b[0]`, the next one into `b[1]`, and so on. The number of bytes read is, at most, equal to the length of `b`. Let `k` be the number of bytes actually read; these bytes will be stored in elements `b[0]` through `b[k-1]`, leaving elements `b[k]` through `b[b.length-1]` unaffected.

The `read(b)` method for class `InputStream` has the same effect as:

```
read(b, 0, b.length)
```

Parameters:
`b` - the buffer into which the data is read.

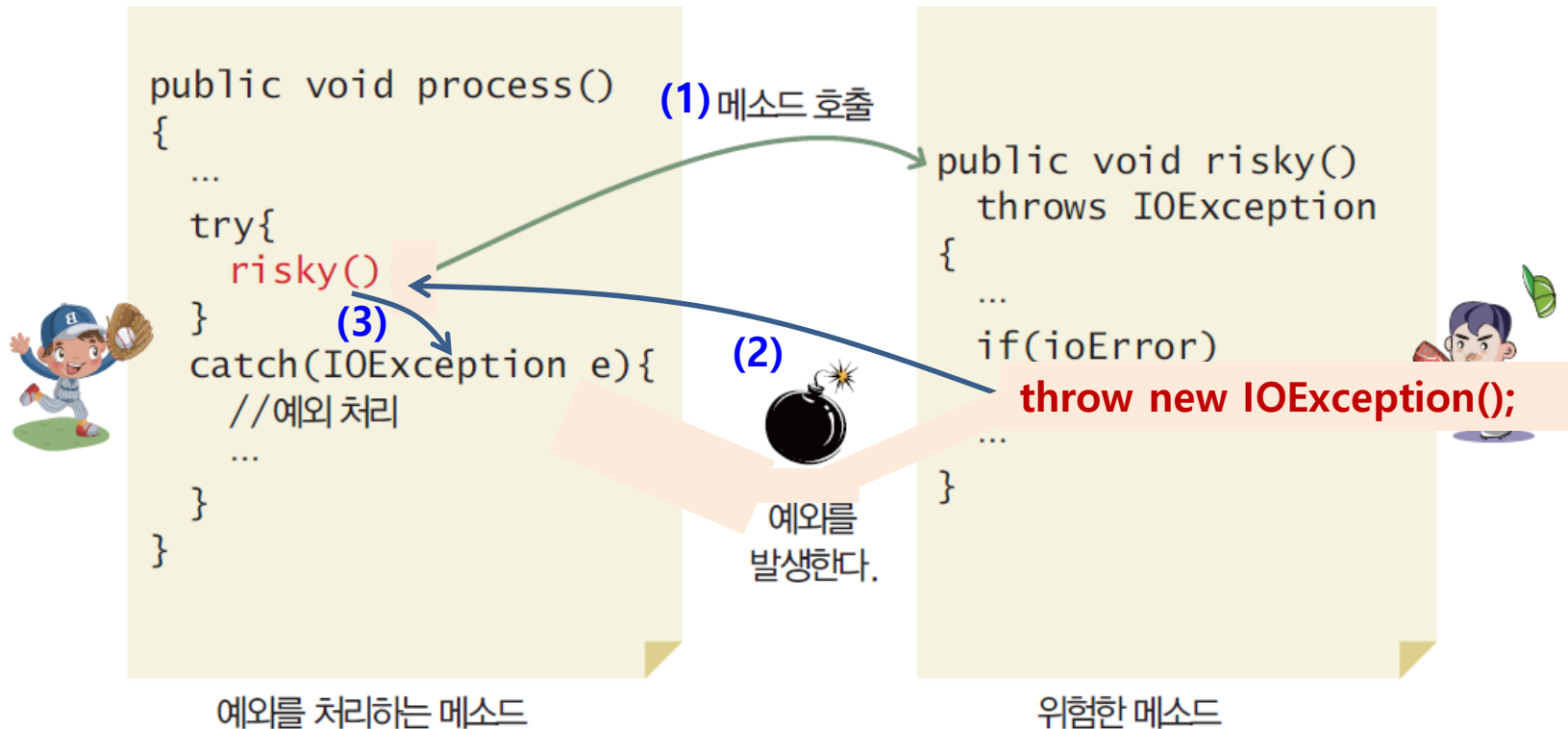
Returns:
the total number of bytes read into the buffer, or `-1` if there is no more data because the end of the stream has been reached.

Throws:
`IOException` - If the first byte cannot be read for any reason other than the end of the file, if the input stream has been closed, or if some other I/O error occurs.
`NullPointerException` - if `b` is null.

어떤 경우에 예외가 발생하는지 설명되어 있다.

예외 발생과 처리

- 예외는 주로 자바 라이브러리에서 발생시킴
- 프로그래머가 직접 예외를 발생시킬 수도 있음



예외 발생

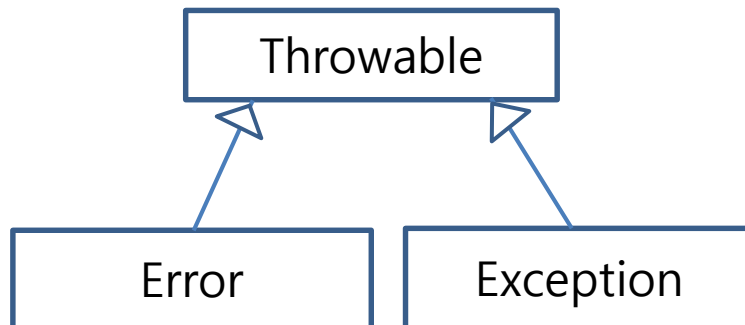
- throw 문장을 사용하여 예외를 발생시킬 수 있다.
 - 즉, 예외 객체를 던진다.

형식

```
throw someThrowableObject;
```

Throwable 객체

- 예외 클래스 계층구조



예외 발생

- 예외 조건을 검사하여 예외 상황인 경우, 예외 객체를 생성하여(new) 던짐(throw)
- 예: size가 0이면, EmptyStackException 예외를 발생시킴

```
public Object pop() {  
    Object obj;  
    if (size == 0) {  
        throw new EmptyStackException(); // 예외 발생  
    }  
    ...  
    return obj;  
}
```

사용자 정의 예외 타입

- 다른 예외와 구별하여 처리하려면 사용자 정의 예외 클래스를 작성한다.
- 보통 Exception 클래스의 서브클래스를 정의한다.

```
public class MyException extends Exception {  
    ...  
}
```

```
class MyException extends Exception {  
    public MyException() {  
        super("사용자 정의 예외");  
    }  
}
```

```
public class ExceptionTest {  
    public static void main(String[] args) {  
        try {  
            method1();  
        }  
        catch (MyException e) {  
            System.err.println(e.getMessage() + "\n호출 스택 내용:");  
            e.printStackTrace();  
        }  
    }  
    public static void method1() throws MyException {  
        throw new MyException(); // 무조건 예외를 발생  
    }  
}
```

실행결과

사용자 정의 예외

호출 스택 내용:

MyException: 사용자 정의 예외

at ExceptionTest.method1(ExceptionTest.java:17)

at ExceptionTest.main(ExceptionTest.java:5)

예외 처리의 장점

- 예외 처리를 사용하면 예외 처리 코드를 정상 코드와 분리할 수 있다.
- 다음과 같은 readFile() 메소드를 작성한다고 하자.

```
readFile()  
{  
    파일을 오픈한다;  
    파일의 크기를 결정한다;  
    메모리를 할당한다;  
    파일을 메모리로 읽는다;  
    파일을 닫는다;  
}
```

- 의사코드(pseudo-code)로 작성한 코드를 비교해보자.
 1. 예외 처리를 사용하지 않은 경우
 2. 예외 처리를 사용한 경우

1. 예외 처리를 사용하지 않은 경우

정상 코드와
예외 처리를 위한 코드가
섞여 있다.

```
errorCodeType readFile {  
    int errorCode = 0;  
  
    파일을 오픈한다;  
    if (theFileIsOpen) {  
        파일의 크기를 결정한다;  
        if (gotTheFileLength) {  
            메모리를 할당한다;  
            if (gotEnoughMemory) {  
                파일을 메모리로 읽는다;  
                if (readFailed) {  
                    errorCode = -1;  
                }  
            } else {  
                errorCode = -2;  
            }  
        } else {  
            errorCode = -3;  
        }  
        파일을 닫는다.  
    } else {  
        errorCode = -5;  
    }  
    return errorCode;  
}
```

2. 예외 처리를 사용한 경우

정상 코드가
예외 처리를 위한 코드
로부터 분리되어 있다.

```
readFile {  
    try {  
        파일을 오픈한다;  
        파일의 크기를 결정한다;  
        메모리를 할당한다;  
        파일을 메모리로 읽는다;  
        파일을 닫는다;  
    } catch (fileOpenFailed) {  
        ...  
    } catch (sizeDeterminationFailed) {  
        ...  
    } catch (memoryAllocationFailed) {  
        ...  
    } catch (readFailed) {  
        ...  
    } catch (fileCloseFailed) {  
        ...  
    }  
}
```

LAB: 예외 처리하기

- 다음 코드의 예외를 처리하여 보자.

```
public class ExceptionTest3 {  
    public static void main(String[] args) {  
        int num = Integer.parseInt("ABC");  
        System.out.println(num);  
    }  
}
```

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "ABC"  
    at java.lang.NumberFormatException.forInputString(Unknown Source)  
    at java.lang.Integer.parseInt(Unknown Source)  
    at java.lang.Integer.parseInt(Unknown Source)  
    at ExceptionTest3.main(ExceptionTest3.java:3)
```

SOLUTION

```
public class ExceptionTest3 {  
    public static void main(String[] args) {  
        try {  
            int num = Integer.parseInt("ABC");  
            System.out.println(num);  
        } catch (NumberFormatException e) {  
            System.out.println("NumberFormatException 예외 발생");  
        }  
    }  
}
```

NumberFormatException 예외 발생