

# COMP217

# JAVA Programming

## Summer 2018

*Day 14*

Exception handling

Textbook:

Power Java 2: Chapters 21

# Goals

- 이번주에 배우게 될 내용 :

- 예외의 종류
- 예외 처리
- 예외 생성

```
피제수 : 10  
제수 : 0  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
at Bicycle.main(Bicycle.java:10)
```

- Motivation

- 프로그램은 실행 중에 (이를 처리할 적절한 코드가 없다면) 예외적인 상황을 만나면, 비정상 종료
- 어떻게 실행 중 에러를 만났을때에도 프로그램을 진행하거나 우아하게 종료할 수 있을까?

- 파워 자바 21장(예외 처리)

# 예외(Exception)

- 잘못된 코드, 예외적인 상황에 의하여 발생하는 오류
  - 0으로 나누는 것과 같은 잘못된 연산
  - 배열의 인덱스가 한계를 넘음
  - open하려는 파일의 부재
  - 숫자를 입력하라고 했는데 문자 입력

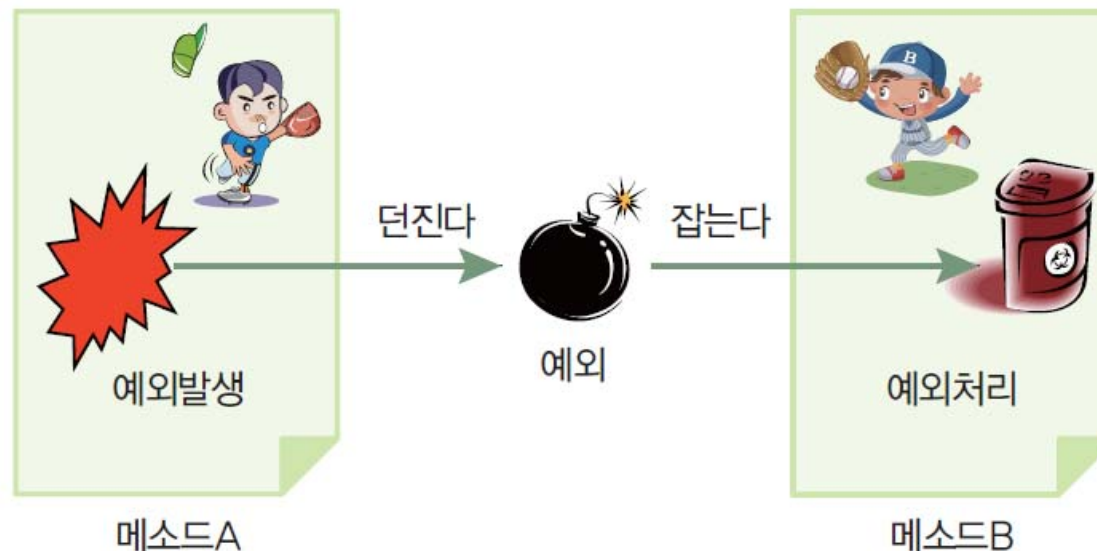


그림21-1. 자바에서는 실행 오류가 발생하면 예외 객체가 생성된다.

# 예외 예제

```
01 import java.util.Scanner;
02 public class DivideByZero {
03     public static void main(String[] args) {
04         int x, y;
05         Scanner sc = new Scanner(System.in);
06         System.out.print("피젯수: ");
07         x = sc.nextInt();
08         System.out.print("젯수: ");
09         y = sc.nextInt();
10         int result = x / y; ← 예외가 발생할 수 있는 문장
11         System.out.println("나눗셈 결과: " + result);
12     }
13 }
```

## 실행결과

피젯수: 10

젯수: 0

Exception in thread "main" java.lang.ArithmeticException:  
/ by zero at DivideByZero.main(DivideByZero.java:14)

0으로 나누면  
예외 발생

# 예외 처리 방법

## ① 예외를 잡아서 그 자리에서 처리하는 방법

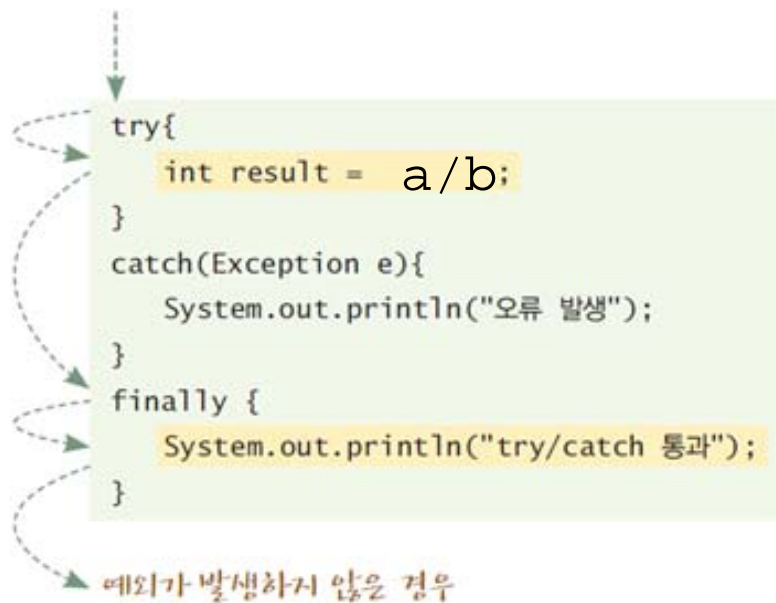
- try-catch 문을 사용하여 예외를 잡고 처리
- try{  
    예외가 발생할 수 있는 문장;  
}catch(Exception1 e1){  
    예외처리 문장;  
}catch(Exception2 e2){  
    예외처리 문장;  
}finally{ //option  
    try문에 진입하면 반드시 실행되는 문장; //주로 자원정리 코드가 작성됨  
}

## ② 발생한 예외를 직접 처리하지 않고, 자신을 호출한 메소드에게 처리를 떠넘기는 방법

- throws를 사용하여, 다른 메소드한테 예외 처리를 맡긴다.

# try-catch 문의 실행 흐름

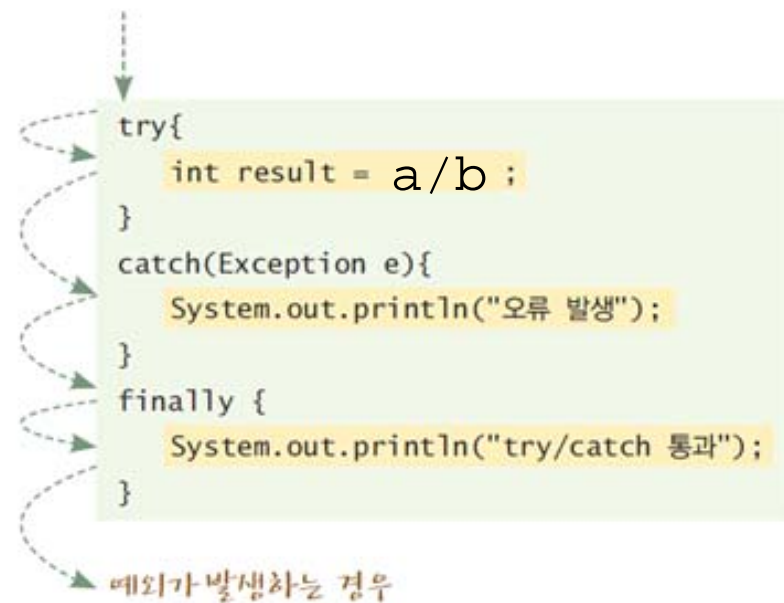
```
Scanner s = new Scanner(System.in);  
int a = s.nextInt();  
int b = s.nextInt();
```



사용자 입력

a = 10

b = 5



사용자 입력

a = 10

b = 0

# try 블록

- 예외 처리는 try 블록으로부터 시작
  - 일반적으로는 제대로 실행되지만,
  - 예외가 발생할 수도 있는 코드가 작성됨
    - e.g. 사용자로부터 파일명 입력받아 파일을 열려는데 그런 이름을 가진 파일이 없음
- 만약 예외가 발생하지 않는다면,
  - try 블록 뒤의 catch 블록은 실행되지 않음
  - finally 블록은 실행됨
- 만약 예외가 발생하면,
  - try 블록의 나머지 부분은 실행되지 않음
  - try 블록뒤의 첫번째 catch 블록부터 시작하여 차례로 적합한 예외 처리 catch 블록을 찾음

# catch 블록

- try 블록 뒤에는 반드시 하나 이상의 catch 블록이 존재
  - 없어도 가능 : but 일반적인 경우는 아님
- catch 블록은 catch(예외 참조 변수){//예외 처리 코드}로 구성
  - catch(Exception e){e.printStackTrace();}
- try 블록 실행과정에서 예외가 발생했다면,
  - 발생한 예외에 적합한 catch 블록이 반드시 있어야 함
  - 어떤 catch 블록이 실행되었다면, 다음 처리 문장은 마지막 catch 블록 다음 문장
  - 만약 적합한 catch 블록이 없어 어떤 catch 블록도 실행되지 않았다면,
    - 프로그램은 Stack trace를 출력하고 종료
    - stack trace is the sequence of method calls



# 예제 #1

## DivideByZeroOK.java

```
01 import java.util.Scanner;
02 public class DivideByZeroOK {
03     public static void main(String[] args) {
04         int x, y;
05         Scanner sc = new Scanner(System.in);
06         System.out.print("피젯수: ");
07         x = sc.nextInt();
08         System.out.print("젯수: ");
09         y = sc.nextInt();
10         try {
11             int result = x / y;           // 예외 발생!
12         } catch (ArithmeticException e) {
13             System.out.println("0으로 나눌 수 없습니다.");
14         }
15         System.out.println("프로그램은 계속 진행됩니다.");
16     }
17 }
```

피젯수: 10

젯수: 0

Exception in thread "main" java.lang.ArithmeticException:  
/ by zero at DivideByZero.main(DivideByZero.java:14)

여기서 오류를 처리한다. 현재는 그냥 콘솔에  
오류 메시지를 출력하고 계속 실행한다.

## 실행결과

피젯수: 10

젯수: 0

0으로 나눌 수 없습니다.

프로그램은 계속 진행됩니다.

예외가 발생해도 프로그램은 종료하지 않는다.

# 예제#2

ArrayError.java

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5  
at ArrayError.main(ArrayError.java:8)

```
01 public class ArrayError {
02     public static void main(String[] args) {
03         int[] array = { 1, 2, 3, 4, 5 };
04         int i = 0;
05         try {
06             for (i = 0; i < array.length; i++)
07                 System.out.print(array[i] + " ");
08         } catch (ArrayIndexOutOfBoundsException e) {
09             System.out.println("인덱스 " + i + "는 사용할 수 없네요!");
10         }
11     }
12 }
```

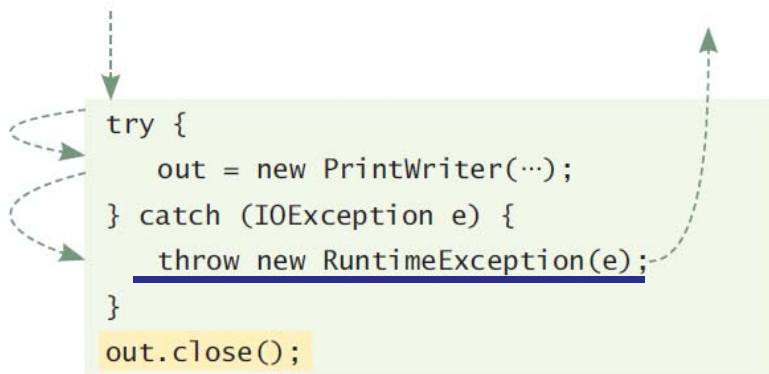
i의 값이 array.length와 같아지면 오류가 발생한다.

실행결과

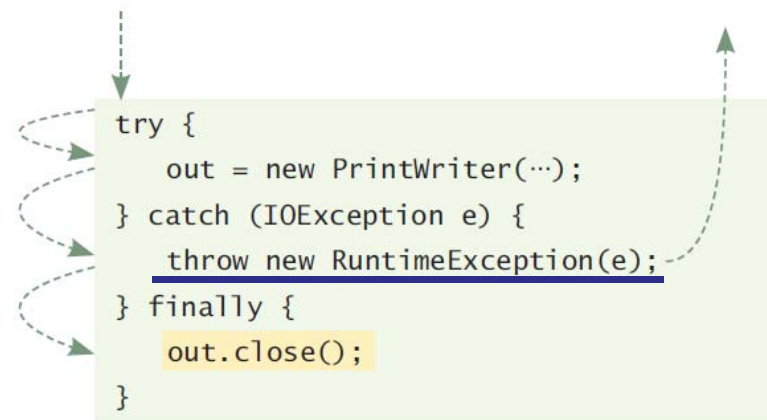
1 2 3 4 5 인덱스 5는 사용할 수 없네요!

# finally 블록

- 생략 가능
- 오류가 발생하였건 발생하지 않았건 항상 실행되어야 하는 코드는 finally 블록에 넣을 수 있다.
- try 문 구성은
  - try 블록 + {catch 블록 or finally 블록 }
  - but try 블록 + finally 블록은 일반적인 구성은 아님



예외가 발생하면 자원이 반납되지 않을 수 있다.



예외가 발생하더라도 확실하게 자원이 반납된다.

throw 는 예외상황이  
발생하였음을  
JVM에게 알리는  
키워드

# 예제 #3

## - finally - 여러 예외가 있는 경우

```
01 ...
02 public class FileError {
03     private int[] list;
04     private static final int SIZE = 10;
05
06     public FileError() {
07         list = new int[SIZE];
08         for (int i = 0; i < SIZE; i++)
09             list[i] = i;
10         writeList();
11     }
12
13     public void writeList() {
14         PrintWriter out = null;
15         try {
16             out = new PrintWriter(new FileWriter("outfile.txt"));
17             for (int i = 0; i <= SIZE; i++)
18                 out.println("배열 원소 " + i + " = " + list[i]);
19
20         } catch (ArrayIndexOutOfBoundsException e) {
21             System.err.println("ArrayIndexOutOfBoundsException: ");
22
23         } catch (IOException e) {
24             System.err.println("IOException");
25
26         } finally {
27             if (out != null)
28                 out.close();
29         }
30     }
31
32     public static void main(String[] args) {
33         new FileError();
34     }
35 }
```

←-----2가지의 오류가 발생할 수 있다.

←-----배열 인덱스 오류가 발생하면 실행된다.

←-----입출력 오류가 발생하면 실행된다.

←-----try 블록이 종료되면 항상 실행되어서 자  
원을 반납한다.

# 예외 처리의 장점

- 에러를 논리 코드로 처리할 경우
- Less readable!!
- 프로그램이 복잡해지면, 에러를 처리하는 코드도 복잡해짐.
- 유지/보수가 어려움.

```
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;
}
```

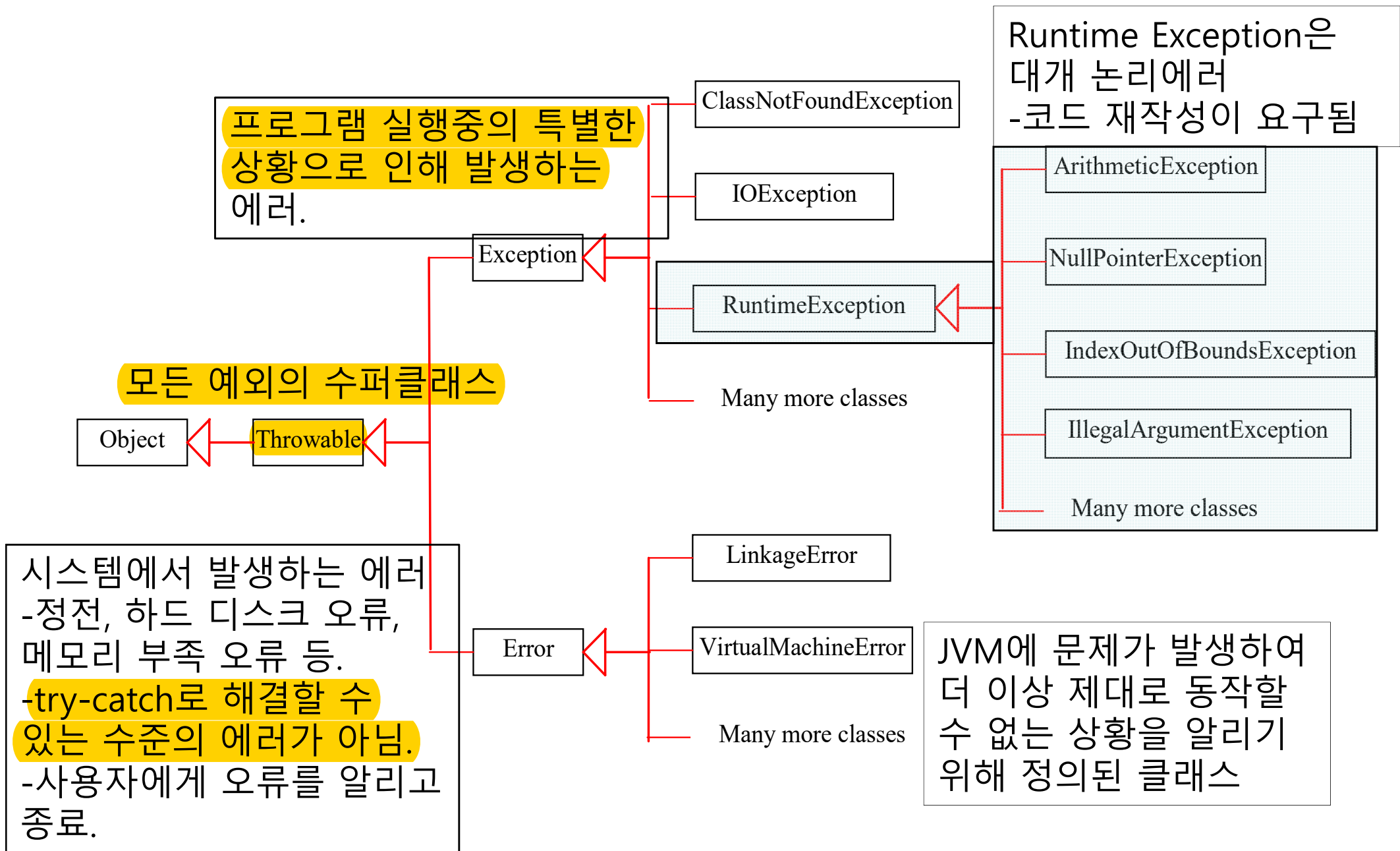
```
readFile() 처리 작업
{
    파일을 오픈한다;
    파일의 크기를 결정한다;
    메모리를 할당한다;
    파일을 메모리로 읽는다;
    파일을 닫는다;
}
```

- Readable!!
- 프로그램의 주된 코드와 오류처리를 구분

```
readFile {
    try { //주처리 작업
        파일을 오픈한다;
        파일의 크기를 결정한다;
        메모리를 할당한다;
        파일을 메모리로 읽는다;
        파일을 닫는다;
```

```
    } catch (fileOpenFailed) { //예외 처리 코드
        ...
    } catch (sizeDeterminationFailed) {
        ...
    } catch (memoryAllocationFailed) {
        ...
    } catch (readFailed) {
        ...
    } catch (fileCloseFailed) {
        ...
    }
}
```

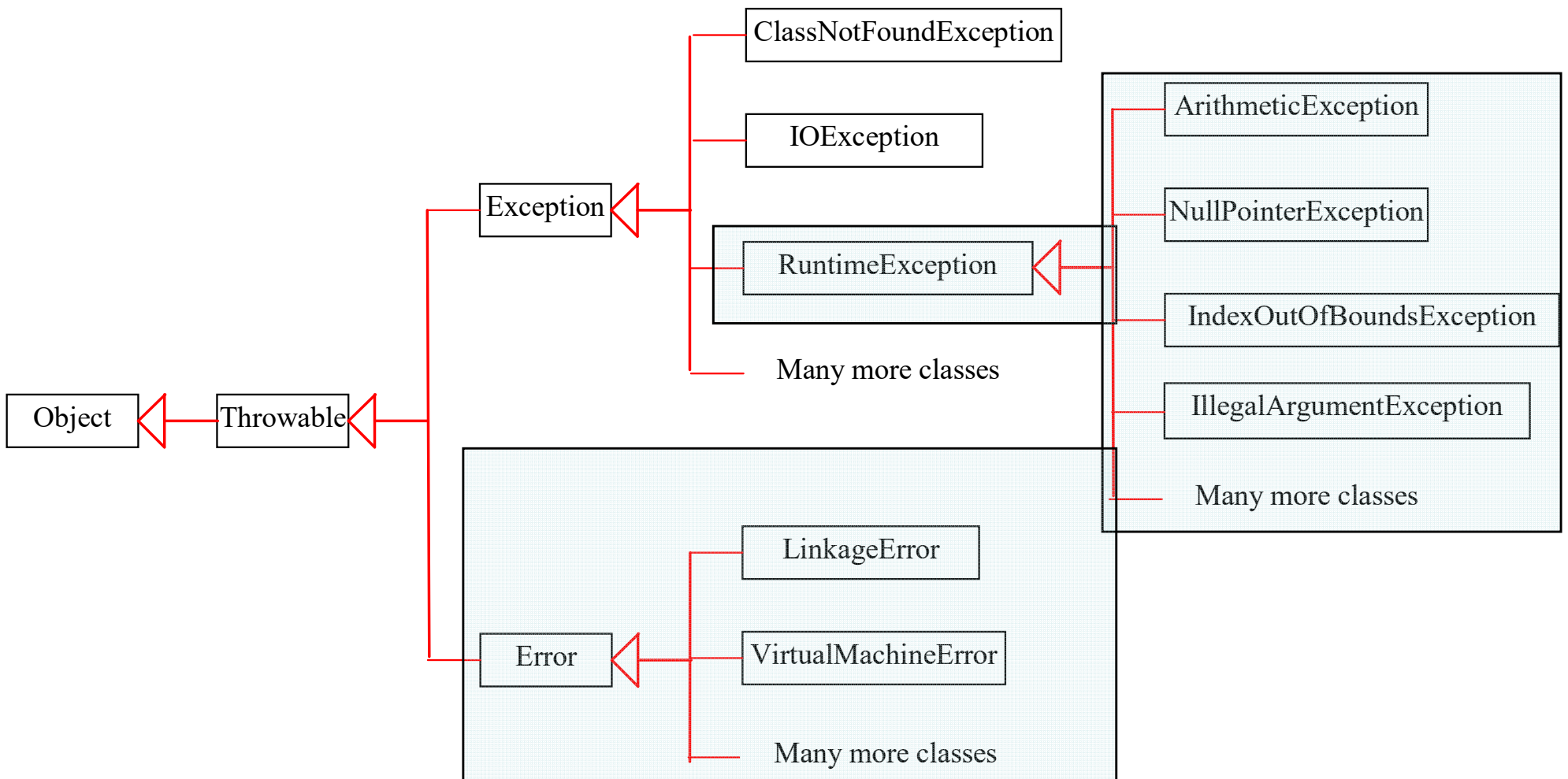
# 예외 클래스 계층도





# Unchecked Exceptions

- Unchecked Exceptions
  - (try-catch나 throws를 통해)예외 처리를 하지 않아도 컴파일 가능
- Unchecked Exceptions외의 예외는 모두 적절한 예외처리를 해야 함
  - 체크 예외(checked exception)
  - 이 예외들을 적절한 처리를 하지 않는다면 컴파일 불가

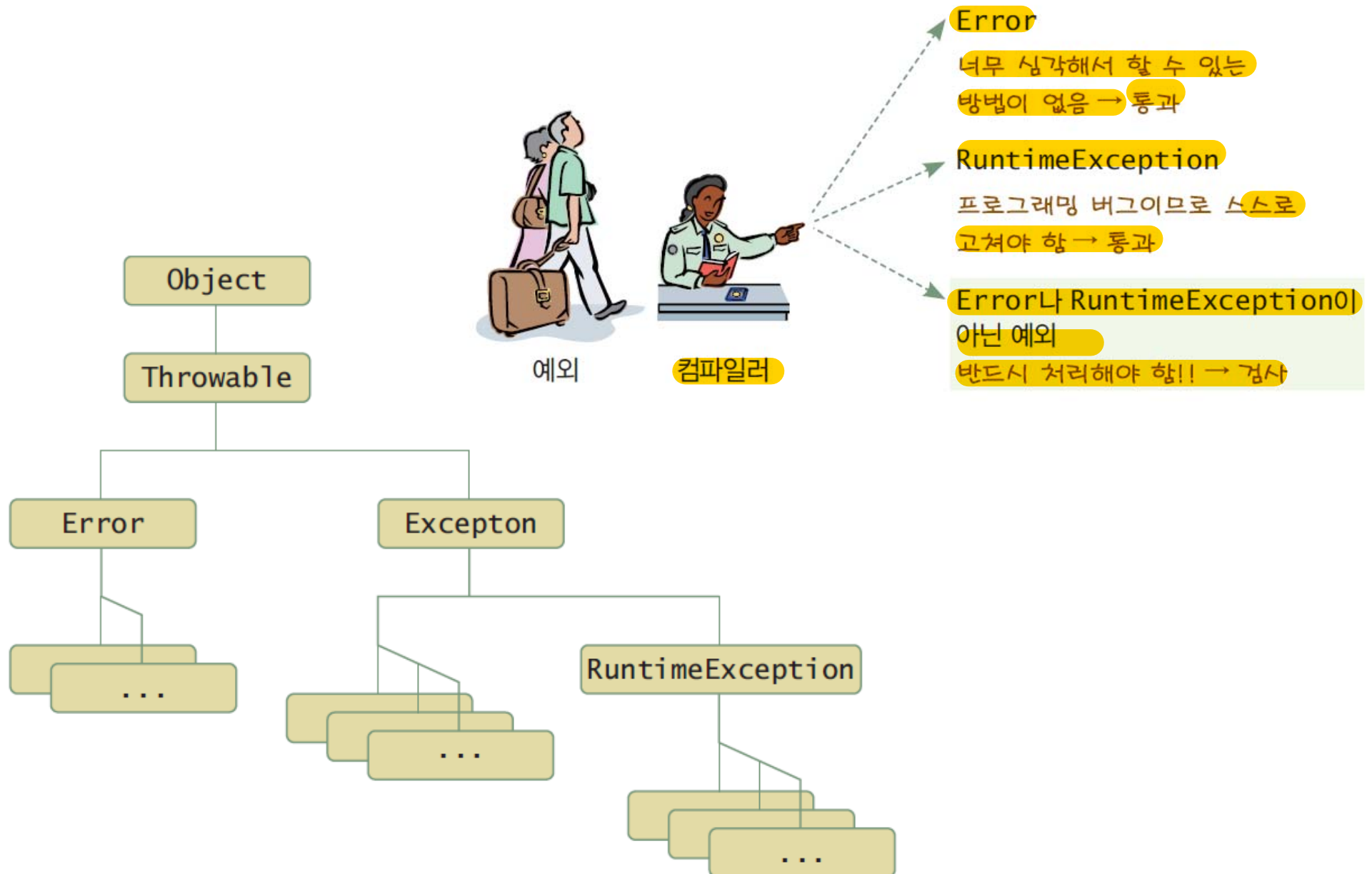


# Runtime Exceptions

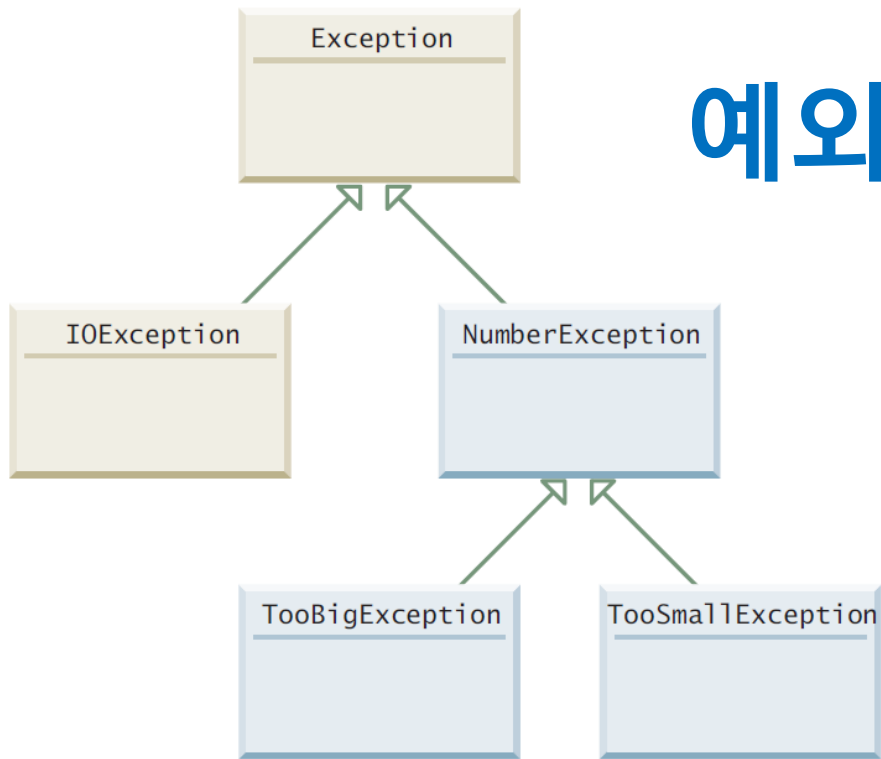
분류	예외	설명
RuntimeException	ArithmeticException	어떤 수를 0으로 나눌 때 발생한다.
	NullPointerException	널 객체를 참조할 때 발생한다.
	ClassCastException	적절치 못하게 클래스를 형변환하는 경우
	NegativeArraySizeException	배열의 크기가 음수값인 경우
	OutOfMemoryException	사용 가능한 메모리가 없는 경우
	NoClassDefFoundException	원하는 클래스를 찾지 못하였을 경우
	ArrayIndexOutOfBoundsException	배열을 참조하는 인덱스가 잘못된 경우



# 예외의 종류



# 예외와 다형성



```
try { // 예외를 발생하는 메소드
    getInput();
}
catch(NumberException e) {
}
```

1

```
try {
    getInput();
}
catch(Exception e) {
}
```

2

```
try {
    getInput();
}
catch(TooSmallException e) {
}
catch(NumberException e) {
}
```

3

```
try {
    getInput();
}
catch(NumberException e) {
}
catch(TooSmallException e) {
}
```

4

컴파일러

# 발생된 예외를 던지는(throws) 경우

- 발생된 예외를 직접 try-catch를 통해 처리하지 않고
- 자신을 호출한 메서드로 처리를 떠넘기는 경우
- 메서드 선언시 **throws** 키워드를 사용하여 발생 가능한 예외도 같이 선언

```
int sub() throws a, b
{
    ...
}
```

메소드 sub()가 a와 b라는 예외를 발생시킬 수 있음을 나타낸다.

- 따라서 예외를 처리하려면, (a) 또는 (b)처럼

```
void p1() {
    try {
        p2();
    }
    catch (IOException ex) {
        ...
    }
}
```

(a) Catch

```
void p1() throws IOException {
    p2();
}
```

(b) Declare

# 예외 선언

```
public void writeList() {  
    PrintWriter = new PrintWriter(new FileWriter("outfile.txt"));  
    for (int i = 0; i < SIZE; i++)  
        out.println("배열 원소 " + i + " = " + list[i]);  
    out.close();  
}
```

오류 발생: Unhandled exception  
type IOException

## 컴파일에러

```
public void writeList() throws IOException  
{  
    ...  
}
```

IOException 예외를 던질 수 있는  
메소드라는 것을 기술한다.

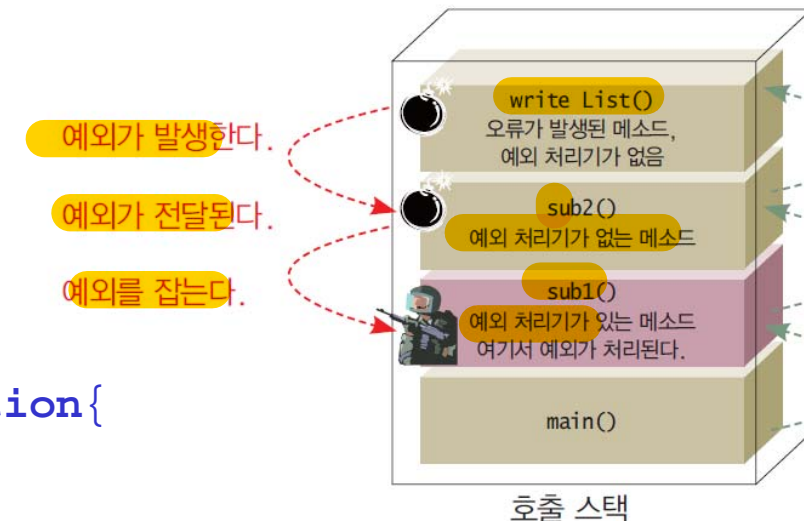
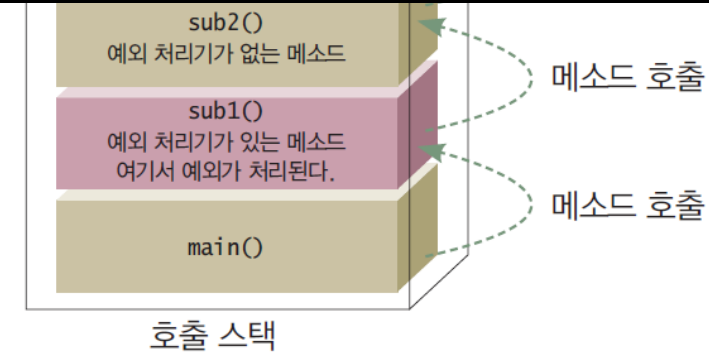
```
public void writeList() throws IOException, ArrayIndexOutOfBoundsException
```

# 예외 처리 과정

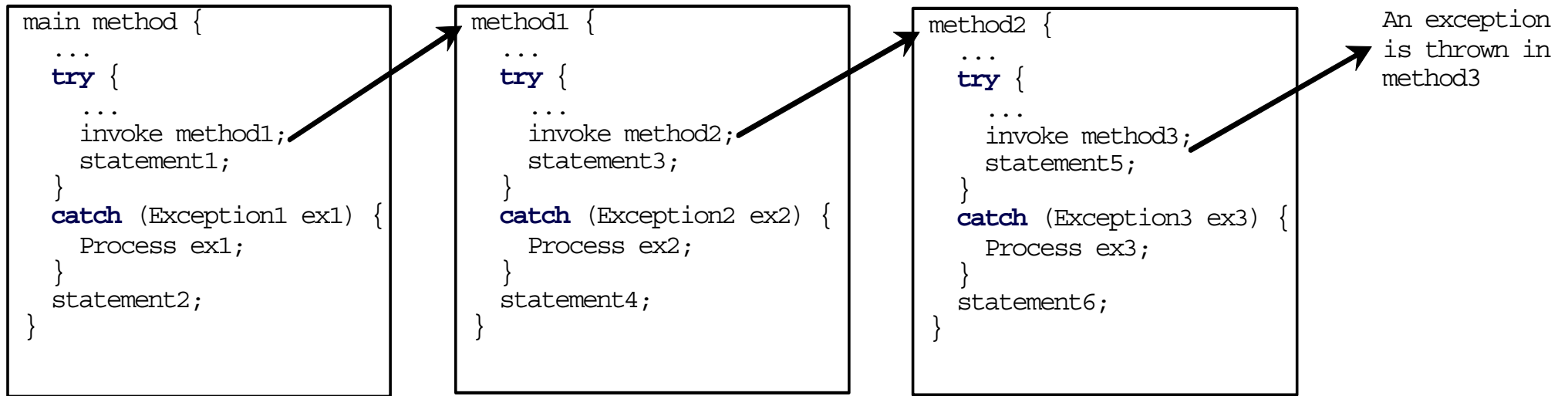
예외가 발생하면 호출 스택을 거슬러가면서 예외 처리기(try-catch)가 있는 메소드를 찾는다.

```
import java.io.IOException;
class TryCatch{
    public static void main(String [] args){
        sub1();
        ...
    }
    public static void sub1(){// exception catcher
        try{
            sub2();
            ...
        }catch(IOException ioe){
            ioe.printStackTrace();
        }
    }
    public static void sub2() throws IOException{
        writeList();
        ...
    }
    public static void writeList() throws IOException{
        ...//exception propagator
    }
}
```

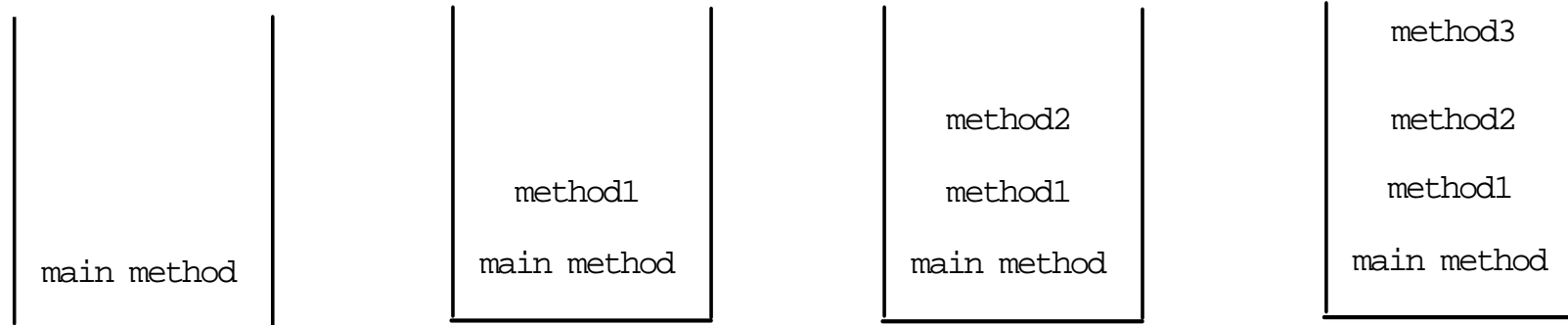
```
java.io.IOException
    at TryCatch.writeList(TryCatch.java:17)
    at TryCatch.sub2(TryCatch.java:14)
    at TryCatch.sub1(TryCatch.java:8)
    at TryCatch.main(TryCatch.java:4)
```



# Catching Exceptions



Call Stack



# 예제

```
01 public class Test {
02     public static void main(String[] args) {
03         System.out.println(readString());
04     }
05
06     public static String readString() {
07         byte[] buf = new byte[100];
08         System.out.println("문자열을 입력하시오:");
09         System.in.read(buf);
10         return new String(buf);
11     }
12 }
```

```
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()); <-- 여기서 예외가 처리된다.
            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);
    }
}
```

예외를 상위 메소드로 전달

## read

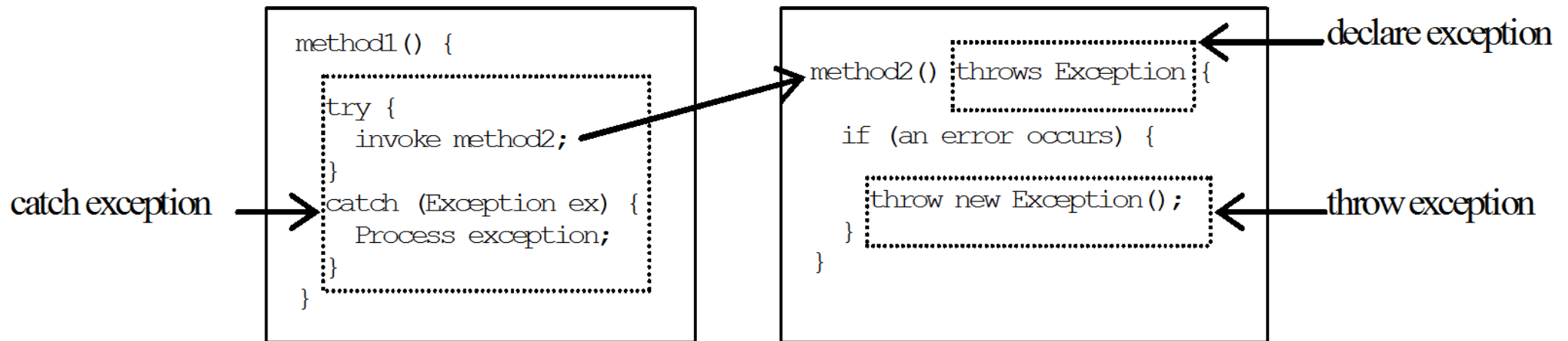
```
public abstract int read()
    throws IOException
```

## 실행결과

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
  Unhandled exception type IOException
  at Test.readString(Test.java:9)
  at Test.main(Test.java:3)
```

# 예외 생성

- 프로그램 실행도중 에러가 발생하면,
  - 그 에러에 적절한 예외 객체 생성
  - 생성한 객체를 throw
    - JVM에게 예외가 발생했음을 알리는 키워드





# 예외 생성 예제

```
throw new TheException(); //객체 생성 후 바로 알림
```

```
TheException ex = new TheException(); //예외 객체 생성 후 참조  
throw ex; //예외 알림
```

```
/** Set a new radius */  
public void setRadius(double newRadius)  
    throws IllegalArgumentException {  
    if (newRadius >= 0)  
        radius = newRadius;  
    else //반지름이 음수라면  
        throw new IllegalArgumentException("//예외생성 후 알림  
            "Radius cannot be negative");  
}
```

# 사용자 정의 예외

- 가능하다면 이미 정의된 예외 클래스를 사용
- 정의된 클래스가 불충분하다면
  - 사용자가 예외 클래스 정의 가능
  - Exception 클래스를 상속하여 작성

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

## ExceptionTest1.java

# 예제

```
01 class MyException extends Exception {
02     public MyException()
03     {
04         super( "사용자 정의 예외" ); //getMessage()를 통해 반환되는 String
05     }                               //getMessage() : 예외상황이 발생한 이유 반환
06 }                                  //Throwable 클래스로 부터 상속받음
07 public class ExceptionTest1 {
08     public static void main( String args[] )
09     {
10         try {
11             method1();
12         }
13         catch ( MyException e )
14         {
15             System.err.println( e.getMessage() + "\n호출 스택 내용:" ); //예외 객체 e의
16             e.printStackTrace(); //stack trace 출력                     getMessage() 메서드 호출
17         }
18     }
19
20     public static void method1() throws MyException
21     {
22         throw new MyException();
23     }
24 }
```

### 실행결과

사용자 정의 예외

호출 스택 내용:

MyException: 사용자 정의 예외

at ExceptionTest.method1(ExceptionTest.java:17)

at ExceptionTest.main(ExceptionTest.java:5)

# 예외 사용시 주의 사항

- 예외를 사용하면 프로그램상 주처리 작업과 에러 처리 작업을 분리할 수 있음
  - 프로그램 수정을 쉽게하고, 가독성을 높여줌
- 그러나 예외 처리는 예외 객체를 새로 생성해야 하기 때문에 시간과 자원을 필요로 하게 됨
  - 따라서, 예측 불가능한 예외 상황을 처리할때만 사용하는 것을 권장
  - 예측이 가능한 단순한 상황에서는 예외로 처리하기 보다는 if문을 통해서 처리하는 것을 권장

# 예외 사용시 주의 사항 - 예제

따라서 아래와 같이 충분히 예측가능한 예외는

```
try {  
    System.out.println(refVar.toString());  
}  
catch (NullPointerException ex) {  
    System.out.println("refVar is null");  
}
```

는 아래와 같이 if문을 통해 처리하는 것이 낫다.

```
if (refVar != null)  
    System.out.println(refVar.toString());  
else  
    System.out.println("refVar is null");
```

1. 예외는 어떤 경우에 발생하는가?
2. 예외를 처리하는 경우와 처리하는 않은 경우를 비교하여 보라. 장점은 무엇인가?
3. 배열에서 인덱스가 범위를 벗어나도 예외가 발생된다. 크기가 10인 배열을 생성하고 11번째 원소에 0을 대입하여 보라. 이 예외를 처리하는 `try-catch` 블록을 만들어 보라.

1. 만약 호출한 메소드가 예외를 발생할 가능성이 있다면 어떻게 하여야 하는가?
2. 예외를 발생할 가능성이 있는 메소드는 어떻게 정의되는가?
3. `throws`가 의미하는 것은 무엇인가?

1. `Error`와 `RuntimeException`은 언제 발생하는가?
2. 자바 코드에서 반드시 처리하여야 하는 예외는 어떤 것들인가?
3. `RuntimeException`을 처리했는지를 왜 컴파일러에서는 검사하지 않는가?

1. `DiskFailureException`을 정의하여 보라. 이 클래스는 매개 변수가 없는 생성자를 가져야 한다. 만약 예외가 매개 변수가 없는 생성자를 이용하여 발생되면 `getMessage()`는 "Disk Failure!"를 반환하여야 한다.
2. 사용자로부터 성적을 입력받아서 평균을 계산하는 프로그램을 작성하여 보자. 만약 사용자가 음수를 입력하면 `NegativeNumberException`을 발생한다. 이 예외를 `catch` 블록으로 잡아서 처리하는 코드도 추가하라.