

# (INTERMEDIATE) JAVA PROGRAMMING

---

## 11. Exceptional Handling Chapter 3

# JAVA: EXCEPTION HANDLING

---

# 예외란?

- 예외(exception): 잘못된 코드, 부정확한 데이터, 예외적인 상황에 의하여 발생하는 오류
  - (예) 0으로 나누는 것과 같은 잘못된 연산이나 배열의 인덱스가 한계를 넘을 수도 있고, 디스크에서는 하드웨어 에러가 발생할 수 있다.

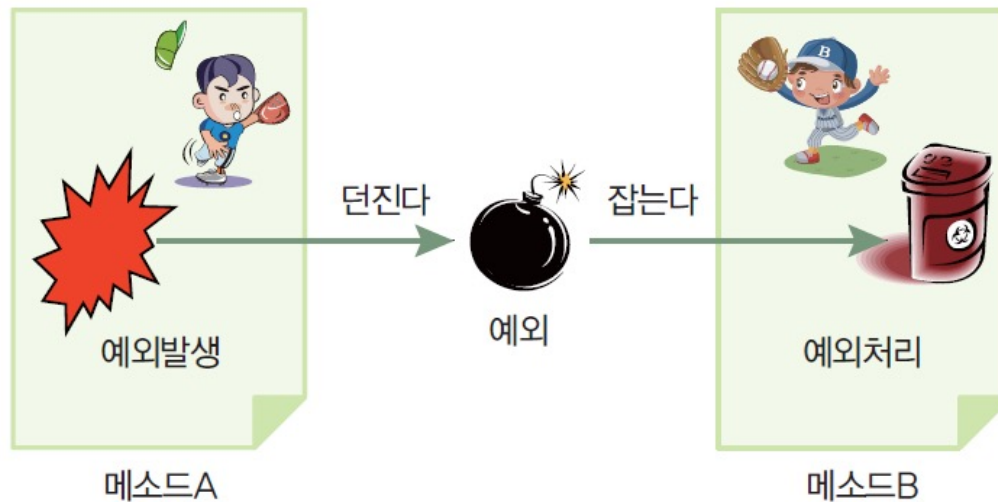


그림21-1. 자바에서는 실행 오류가 발생하면 예외가 생성된다.

# 예외의 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;
11         System.out.println("나눗셈 결과: " + result);
12     }
13 }
```

예외가 발생할 수 있는 문장

## 실행결과

0으로 나누면  
예외 발생

피젯수: 10  
젯수: 0

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

# 예외 처리기



try 블록에서  
오류가 발생하면  
처리합니다.

그림21-2. try 블록은 예외가 발생할 수 있는 위험한 코드이다. catch블록은 예외를 처리하는 코드이다.

# 예외 처리기의 기본 형식

```
try {  
    // 예외가 발생할 수 있는 코드  
} catch (예외종류 참조변수) {  
    // 예외를 처리하는 코드  
}  
} finally {  
    // 여기 있는 코드는 try 블록이 끝나면 무조건 실행된다. ← 생략이 가능하다.  
}
```

# try/catch 블록에서의 실행 흐름

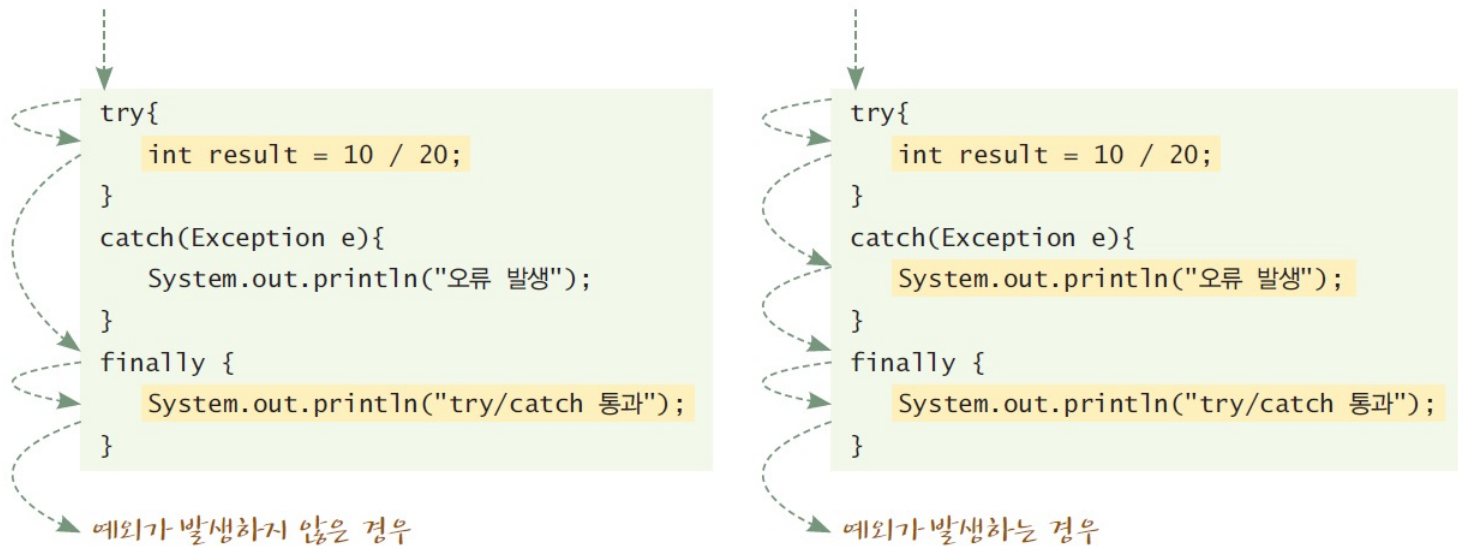


그림21-3. try/catch 블록에서의 실행 흐름

# 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 }
```

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



# 실행 결과

## 실행결과

피젯수: 10

젯수: 0

0으로 나눌 수 없습니다.

프로그램은 계속 진행됩니다.

예외가 발생해도 프로그램은 종료하지 않는다.

## 예제 #2

ArrayError.java

```
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 }
```

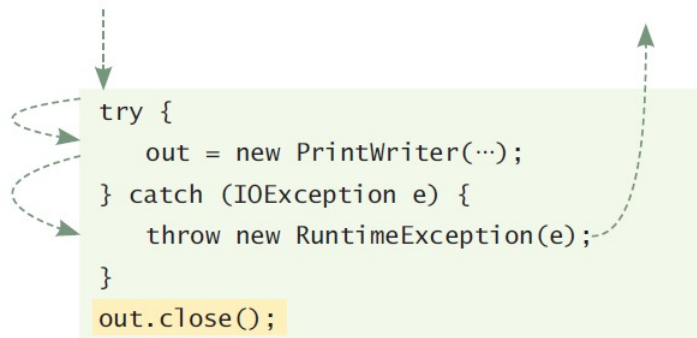
i의 값이 array.length와 같아지면 오류가 발생한다.

실행결과

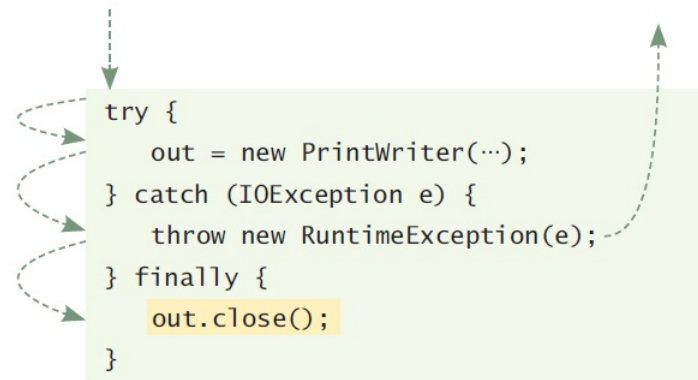
1 2 3 4 5 인덱스 5는 사용할 수 없네요!

# finally 블록

- 오류가 발생하였건 발생하지 않았건 항상 실행되어야 하는 코드는 finally 블록에 넣을 수 있다.



예외가 발생하면 자원이 반납되지 않을 수 있다.



예외가 발생하더라도 확실하게 자원이 반납된다.

# 예제 #3

FileError.java

```
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: ");
```

←-----2가지의 오류가 발생할 수 있다.

←-----배열 인덱스 오류가 발생하면 실행된다.

## 예제 #3

```
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 }
```

←-----입출력 오류가 발생하면 실행된다.

←-----try 블록이 종료되면 항상 실행되어서 자원을 반납한다.

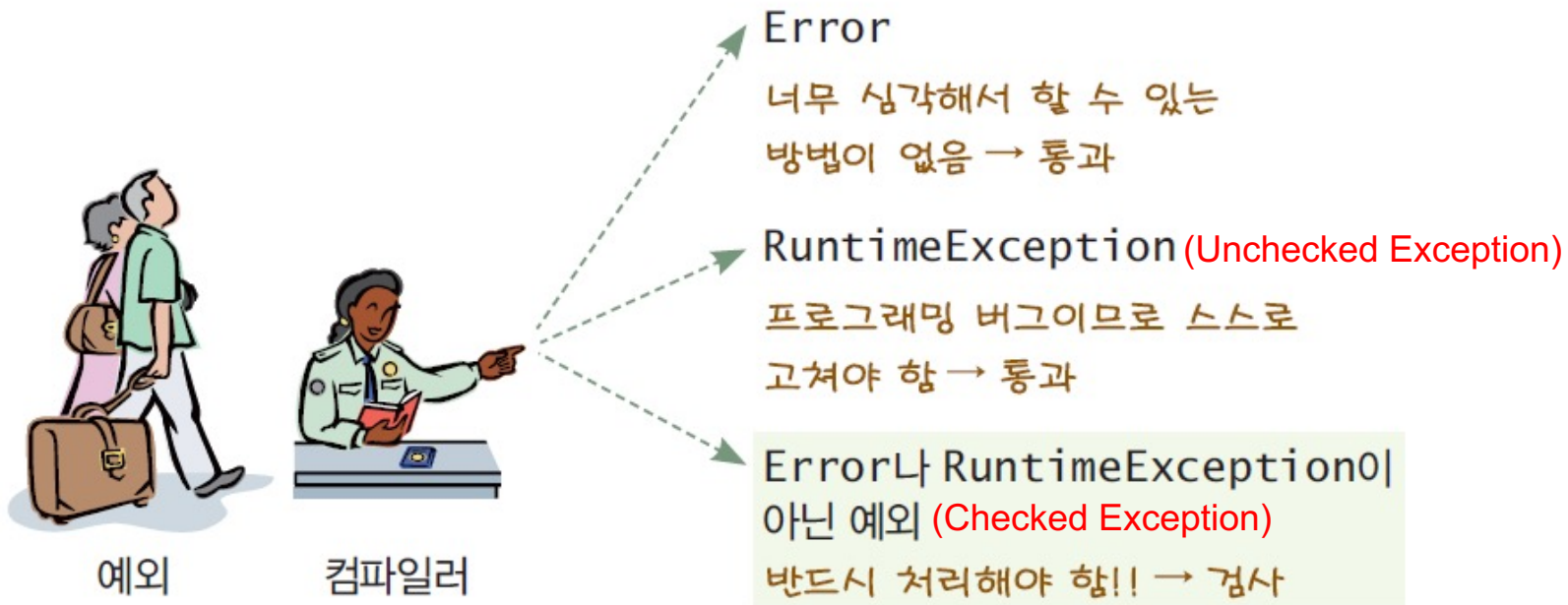
# 중간 점검



## 중간점검

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.
4. 2번 문제에 배열 참조 변수에 `null`을 대입하여 배열을 삭제하는 문장을 `finally` 블록으로 만들어서 추가하여 보라.

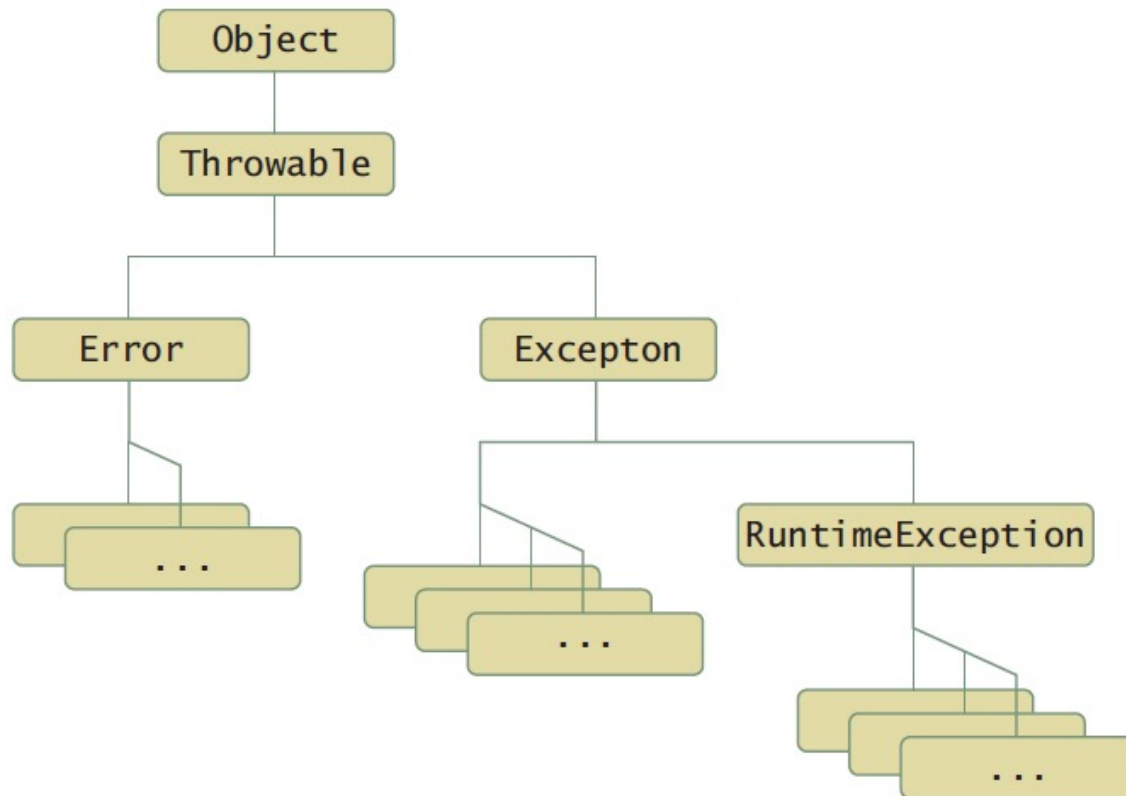
# 예외의 종류



Unchecked Exception : 반드시 처리할 필요는 없음 (처리가 강제되지 않음)  
: 매우 자주 일어나는 에러인 경우에 해당됨  
: 코딩의 편의를 위해 처리를 강제하지 않음

Checked Exception : 반드시 처리해야함 (처리가 강제됨)

# 예외의 종류





# 예외의 종류

- **Error**
  - 자바 가상 기계 안에서 치명적인 오류가 발생
- **RuntimeException (Unchecked Exception)**
  - 프로그래밍 버그나 논리 오류에서 기인한다.

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

- 기타 예외 (**Checked Exception**)

- Error와 RuntimeException을 제외한 나머지 예외
- 회복할 수 있는 예외이므로 프로그램은 반드시 처리
- (예) 사용자가 실수로 잘못된 파일 이름을 입력한다면 FileNotFoundException 예외가 발생한다.
- 체크 예외(**checked exception**)라고 불린다. → 컴파일러가 예외처리를 했는지를 체크한다는 의미 (예외처리가 강제됨)

# 다형성과 예외

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

```
try {  
    getInput();  
}  
catch(Exception e) {  
    //Exception의 모든 하위 클래스를 잡을 수 있으나 분간할 수 없다!  
}
```

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

# 다형성과 예외

```
        //TooSmallException만 잡힌다.  
    }  
    catch(NumberException e) {  
        //TooSmallException을 제외한 나머지 예외들이 잡힌다.  
    }  
}
```

```
try {  
    getInput();  
}  
catch(NumberException e) {  
    //모든 NumberException이 잡힌다.  
}  
catch(TooSmallException e) {  
    //아무 것도 잡히지 않는다!  
}  
}
```

# 중간 점검 문제



## 중간점검

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

# 예외와 메소드

- ① 예외를 잡아서 그 자리에서 처리하는 방법: **try-catch** 블록을 사용하여서 예외를 잡고 처리한다.
- ② 메소드가 예외를 발생시킨다고 기술하는 방법: **throws**를 사용하여, 다른 메소드한테 예외 처리를 맡긴다.

# 메소드가 예외를 발생시킨다고 기술하는 방법


```
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



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

IOException 예외를 던질 수 있는  
메소드라는 것을 기술한다.



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

## 예외 발생 메소드 정의

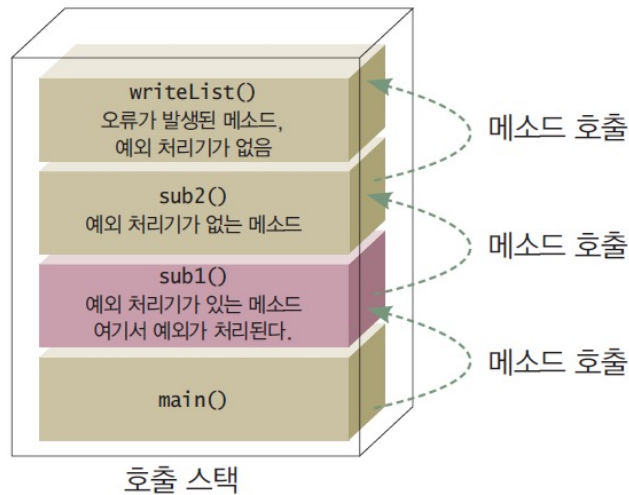
```
int sub() throws a, b  
{  
    ...  
}
```

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



# 예외 처리 과정

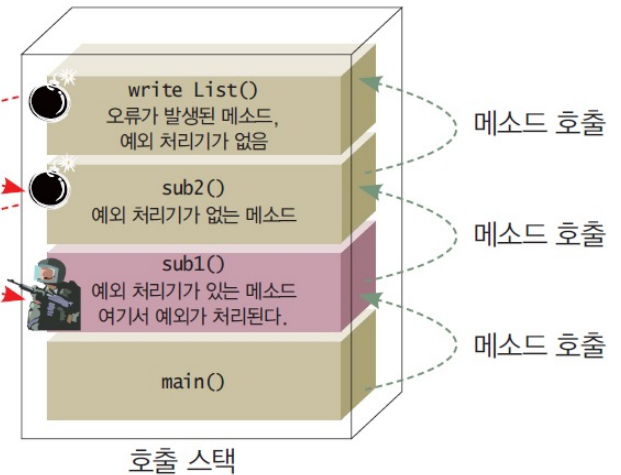
- 호출 스택을 거슬러가면서 예외 처리기가 있는 메소드를 찾는다.



예외가 발생한다.

예외가 전달된다.

예외를 잡는다.



# 예제

- 예외를 발생하는 메소드

The screenshot shows the Java API documentation for the `read()` method in the `InputStream` class. The browser window title is "InputStream (Java Platform SE 6) - Microsoft Internet Explorer". The address bar shows the URL `http://java.sun.com/javase/6/docs/api/`. The left sidebar lists various Java packages and classes, with `java.io` selected. The main content area displays the `read()` method signature: `public abstract int read() throws IOException`. A callout box points to `IOException` in the signature, stating: "IOException이라고 하는 예외를 던질 수 있음을 표시한다." (Indicates that an exception called IOException can be thrown). Below the signature, the text describes the method's behavior: "Reads the next byte of data from the input stream. The value byte is returned as an int in the range 0 to 255. If no byte is available because the end of the stream has been reached, the value -1 is returned. This method blocks until input data is available, the end of the stream is detected, or an exception is thrown." It also states: "A subclass must provide an implementation of this method." The "Returns:" section says: "the next byte of data, or -1 if the end of the stream is reached." The "Throws:" section says: "`IOException` - if an I/O error occurs." A second callout box points to this "Throws:" section, stating: "어떤 경우에 예외가 발생되는지가 설명되어 있다." (It is explained which cases the exception occurs).

- 처리 방법
  - 예외를 try/catch로 처리하는 방법
  - 예외를 상위 메소드로 전달하는 방법

# 예제

Test.java

```
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 입력하십시오:");  
09         System.in.read(buf);  
10         return new String(buf);  
11     }  
12 }
```

## 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)
```

# 예제

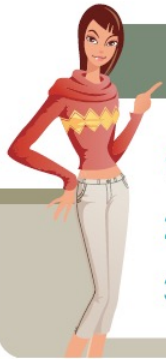
```
import java.io.IOException;

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

# 중간점검



## 중간점검

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?

# 예외 생성하기

- 예외는 `throw` 문장을 이용하여 생성한다.

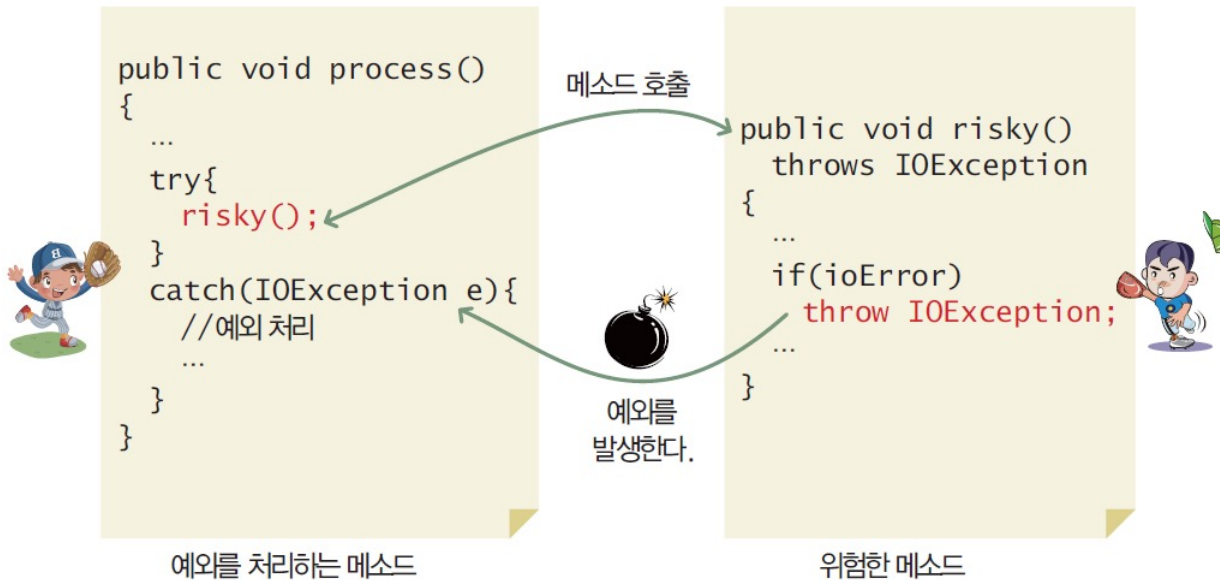


그림20-2. 예외를 던지고 받기

# throw문장

- 예외는 throw 문장으로 발생한다.

```
throw someThrowableObject;
```

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

이 문장에 의하여 예외 객체가 생성된다.

# 연속적인 예외 발생

- 어떤 애플리케이션은 예외를 처리하면서 다른 예외를 발생시킨다.

```
try {  
    ...  
} catch (IOException e) {  
    throw new SampleException("다른 예외", e);  
}
```

← 예외를 처리하는 과정에서  
다른 예외를 발생시킨다.



# 사용자 정의 예외

- 사용자가 예외를 정의할 수 있다.

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

# 예제

## ExceptionTest1.java

```
01 class MyException extends Exception {
02     public MyException()
03     {
04         super( "사용자 정의 예외" );
05     }
06 }
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:" );
16             e.printStackTrace();
17         }
18     }
19
20     public static void method1() throws MyException
21     {
```

# 예제

```
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)

# 중간 점검 문제



## 중간점검

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.

# 예외 처리의 장점

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

```
errorCodeType readFile {  
    int errorCode = 0;  
  
    파일을 오픈한다;  
    if (theFileIsOpen) {  
        파일의 크기를 결정한다;  
        if (gotTheFileLength) {
```

# 예외 처리의 장점

```
    메모리를 할당한다;
    if (gotEnoughMemory) {
        파일을 메모리로 읽는다;
        if (readFailed) {
            errorCode = -1;
        }
    } else {
        errorCode = -2;
    }
} else {
    errorCode = -3;
}
    파일을 닫는다.
} else {
    errorCode = -5;
}
return errorCode;
}
```

# 예외 처리의 장점

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