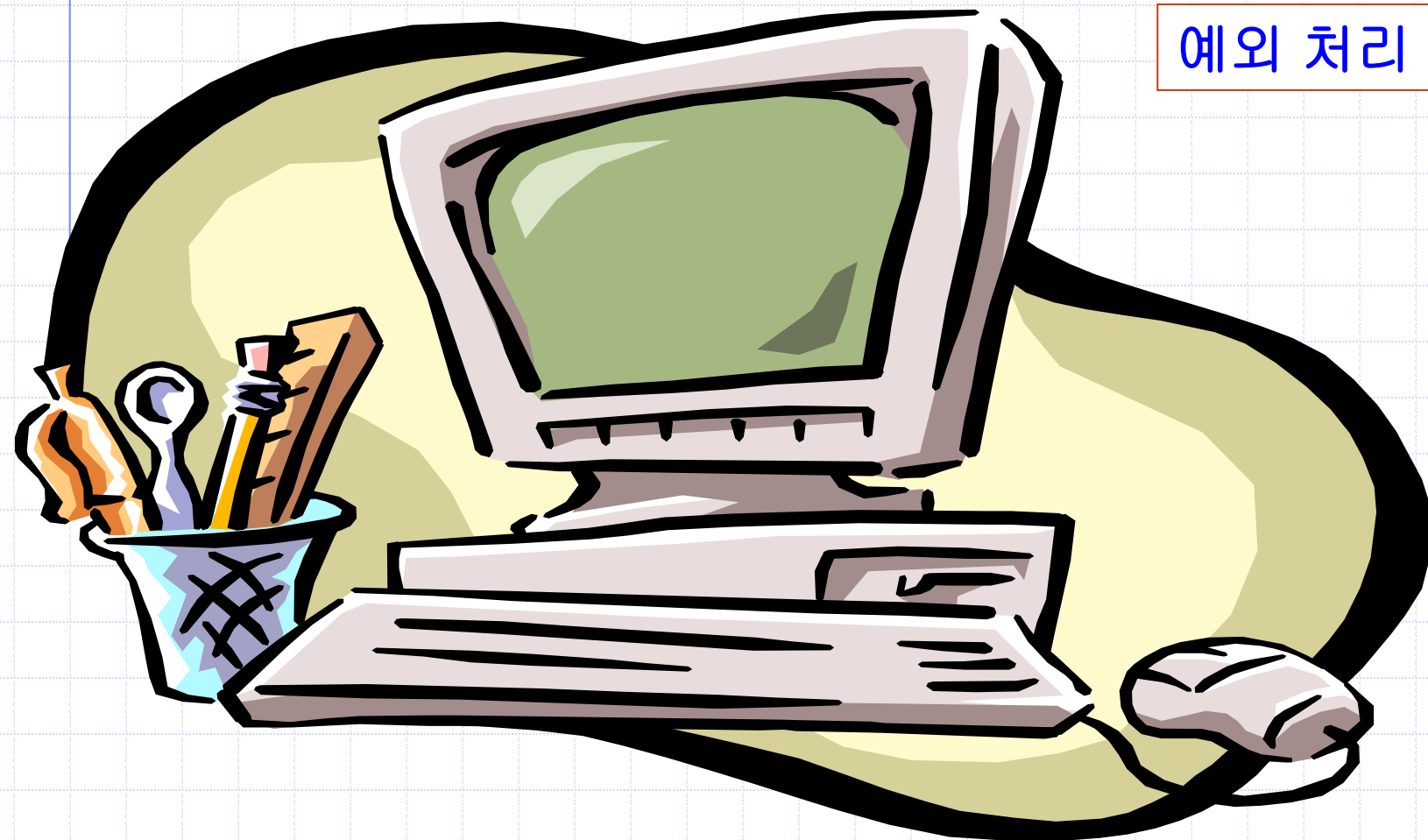
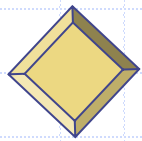


# 예외

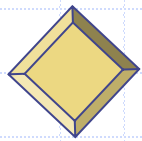
예외 처리





## 6장 예외 처리

1. 예외(exception)
2. 예외 잡기
3. 메소드에서 던지는 예외 알리기
4. 예외 던지기
5. 예외 만들기
6. 예외를 처리하는 3가지 방법



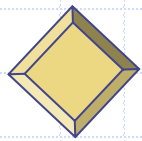
## 6장 예외 처리

### ◆ 예외(Exception)

- 예외적인 사건.
- 프로그램 수행 중에 프로그램의 정상적인 명령 수행 흐름을 방해하는 사건.

### ◆ 예외 분류

- 프로그램에서 반드시 처리해야 하는 예외
  - ◆ **검사 예외**
  - ◆ 처리해야 하는 예외는 컴파일러가 검사(check)하여 예외 처리를 하지 않은 경우에 오류 메시지를 내고 컴파일이 실패함.
- 처리할 필요가 없는 예외
  - ◆ **비검사 예외**



# 1. 예외(exception)

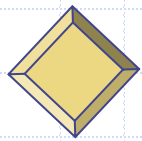
## ◆ 메소드

- 수행 중에 오류 발생하면 → 예외 객체를 생성하여 런타임 시스템에 전달

## ◆ 예외 객체

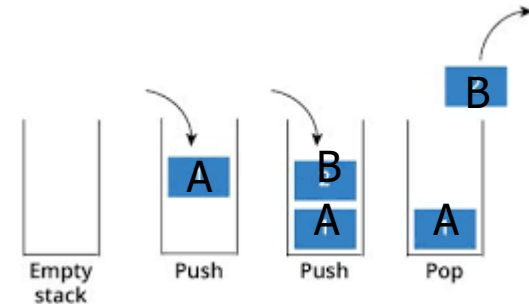
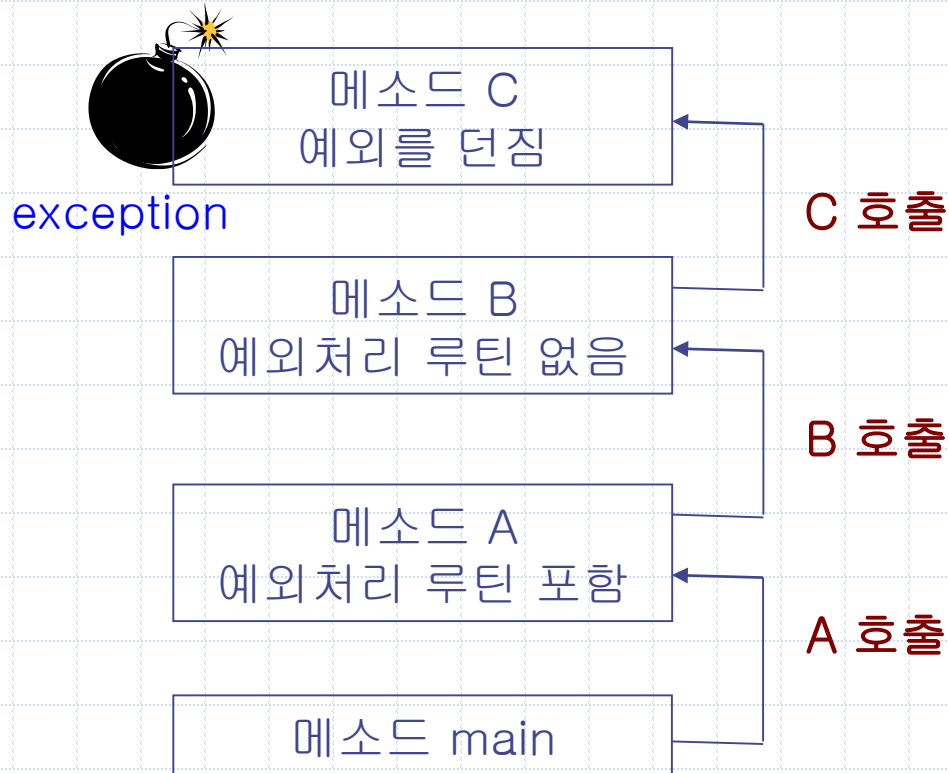
- 오류에 대한 정보, 오류가 발생하였을 때 프로그램의 상태 등을 포함.

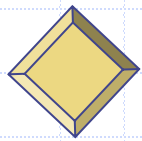
## ◆ 예외 객체를 생성하고 런타임 시스템에 알리는 작업을 예외 던지기(throw)라고 함.



# 1.1 예외처리와 호출 스택

## ◆ 호출 스택(call stack)





# 1.1 예외처리와 호출 스택

## 예외 처리 과정



예외 던지기(throw)



예외 전달



예외 잡기(catch)

메소드 C  
예외를 던짐

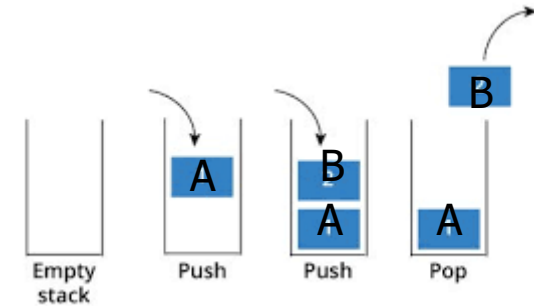
메소드 B  
예외처리 루틴 없음

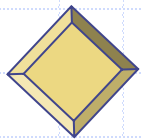
메소드 A  
예외처리 루틴 포함

메소드 main

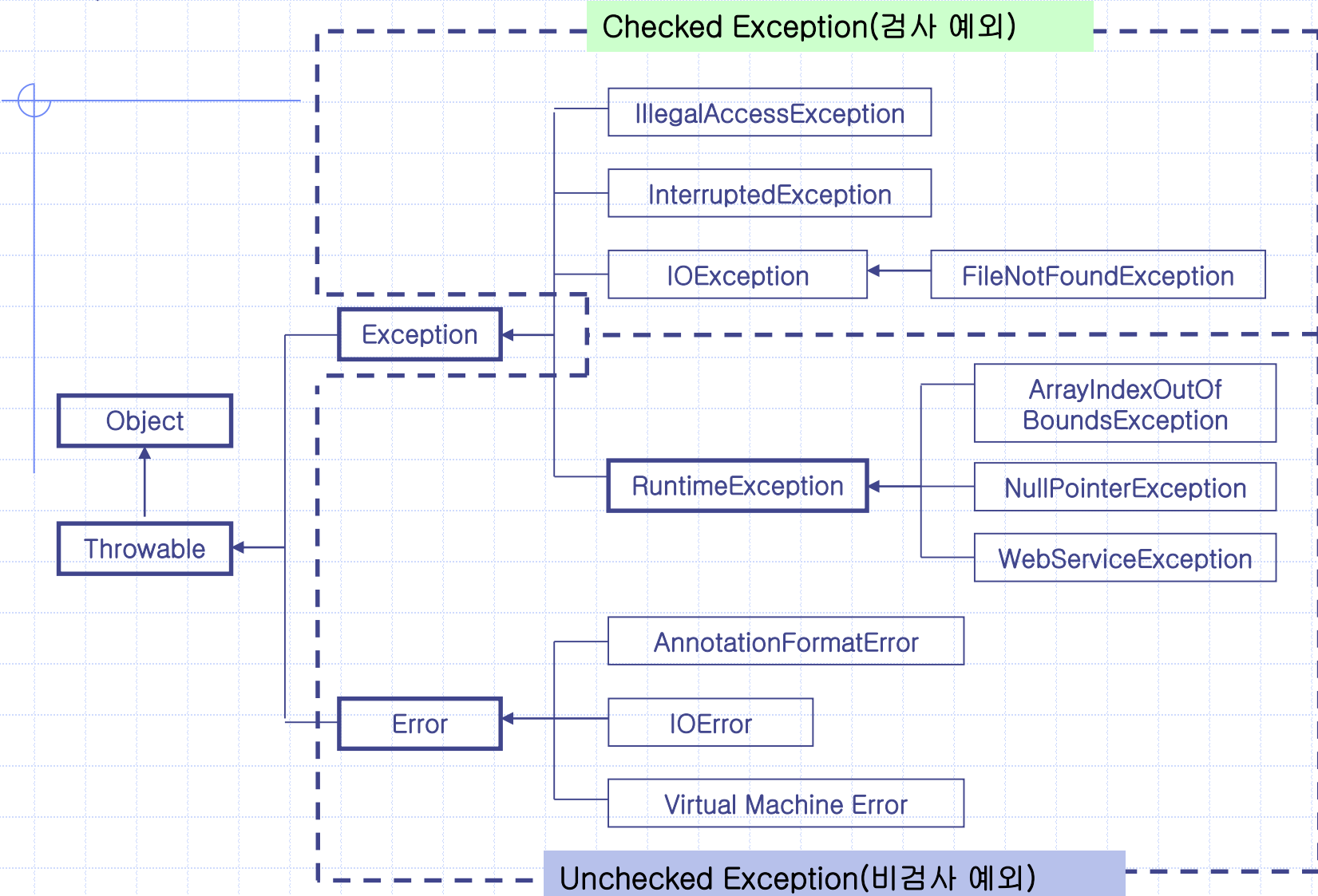
예외처리  
루틴 찾기

예외처리  
루틴 찾기





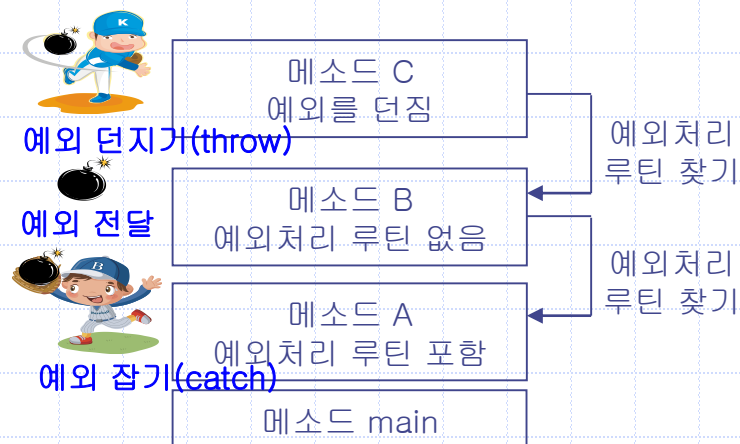
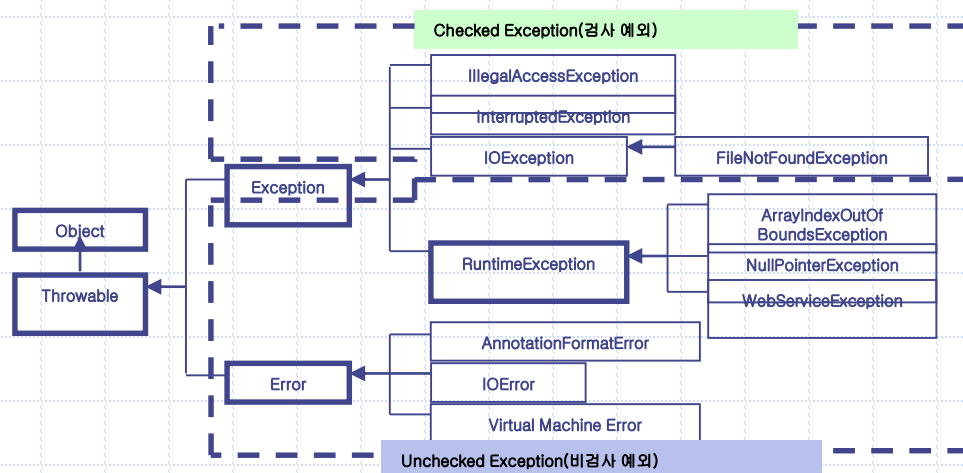
## 1.2 예외 종류



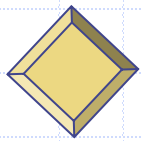
## 1.2 예외 종류

◆ 자바에서는 검사 예외에 대하여 “예외 처리 또는 예외 전달의 요구사항”

- 둘 중 하나 선택
  - ◆ 자신이 예외를 처리하든지
  - ◆ 호출 스택에 있는 다른 메소드가 예외를 처리하도록 예외를 전달하든지
- <그림 6.2>에서 메소드 B는 예외를 전달하고 메소드 A가 예외를 처리한다







## 1.2 예외 종류

### ◆ “예외 처리 또는 예외 전달의 요구사항”에서 검사 예외를 던지는 코드 작성

#### ■ 둘 중 하나의 코드 작성

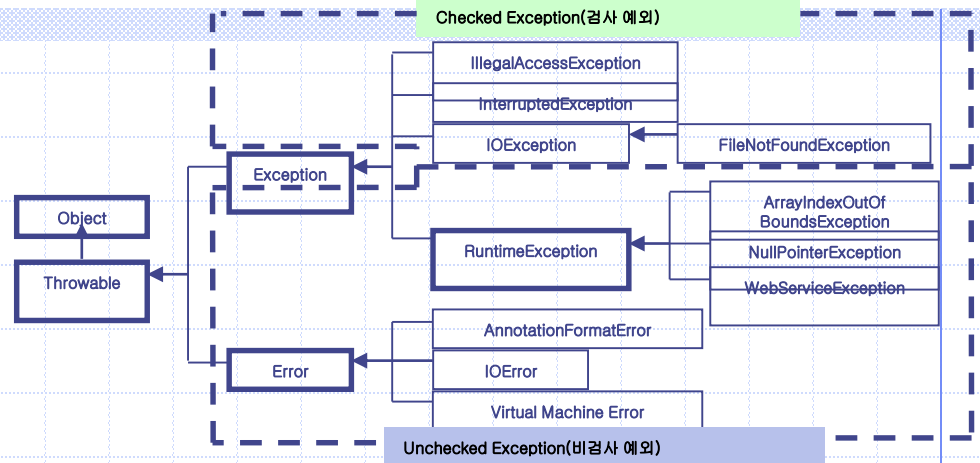
- ◆ ① try 블록 내에 예외를 던지는 코드를 작성하고, catch 블록 내에 예외를 처리한다. (p.242 try 블록)

```
try {  
    code //프로그램 코드(예외를 던지는 명령이 있음)  
} catch ( ) { . . . . }
```

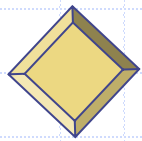
- ◆ ② throws 구문으로 메소드가 예외를 던진다고 선언하여 예외를 전달한다. (p.257 6.3절)

```
public void printVector() throws Exception {  
    .....  
}
```

# ◆ (1) 검사 예외

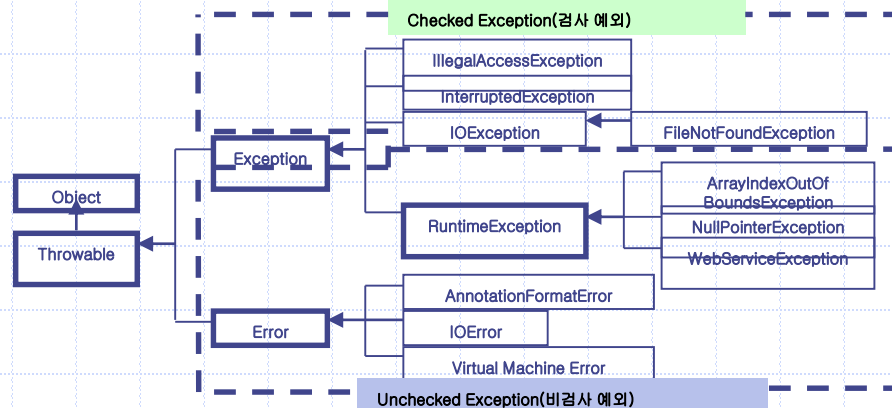


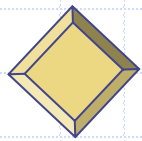
- ◆ 프로그램에서 예측이 가능하며 처리도 가능한 예외
- ◆ “예외 처리 또는 예외 전달의 요구사항”을 준수하여야 한다.
  - 호출자는 검사 예외를 처리하든지(나중에 배울 try/catch 구문으로) 또는 다른 루틴이 예외를 처리하도록 명시하여야(나중에 배울 throws 구문으로) 한다.
  - 만약 이렇게 하지 않으면 컴파일 오류가 발생



## (2) 실행시점 예외(runtime exception)

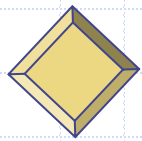
- ◆ 프로그램 내부와 연관되어 있으며, 프로그램이 미리 예외 발생을 예측하거나 예외 처리를 할 수 없다.
  - 프로그램 버그, API 등을 잘 못 사용할 때 발생
  - Ex) 어떤 메소드의 매개변수에 객체를 할당해야 하는데 로직 에러에 의해 null 값이 전달되는 경우에 NullPointerException 예외가 던져진다.
    - ◆ 프로그램에서 이 예외를 처리할 수도 있겠지만 예외를 발생시킨 버그를 제거하는 것이 올바른 해결 방법.
- ◆ 실행시점 예외는 검사 예외가 준수해야 하는 “예외 처리 또는 예외 전달의 요구사항”을 준수할 필요가 없다.



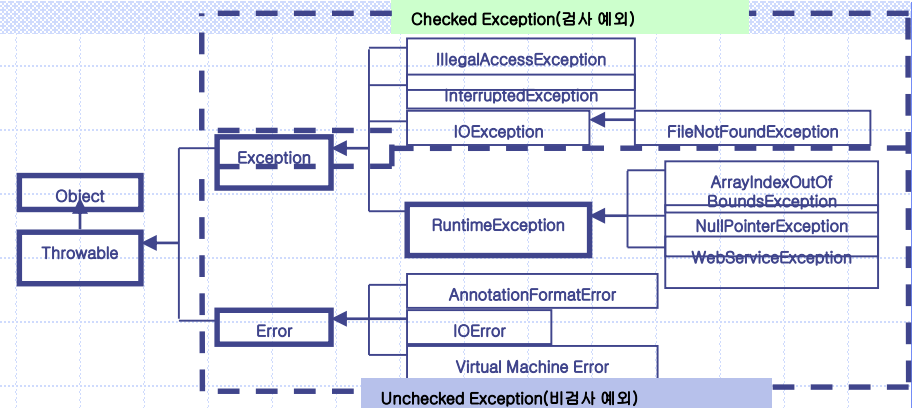


## (2) 실행시점 예외(runtime exception)

예외	설명
ArithmeticException	어떤 수를 0으로 나눌 때 발생하는 예외.
ArrayIndexOutOfBoundsException	배열을 참조하는 인덱스가 잘못되어 발생하는 예외.
ClassCastException	클래스의 적절치 못한(상속 관계가 아닌 클래스 간) 형변환을 하여 발생하는 예외.
IncompatibleClassChangeException	클래스의 변수선언이 static에서 nonstatic이나 반대로 변경되었는데 다른 클래스가 이 변수를 참조하여 발생하는 예외.
NegativeArraySizeException	배열에 크기가 음수값인 경우에 발생하는 예외.
NoClassDefFoundException	원하는 클래스를 찾지 못했을 때 발생하는 예외.
NullPointerException	null 객체를 참조할 때 발생하는 예외.
NumberFormatException	문자를 Integer.parseInt를 이용해 숫자로 변경시도시 발생하는 예외.
OutOfMemoryException	사용가능한 메모리가 없는 경우에 발생하는 예외.
WebServiceException	JAX-WS를 통한 실행 예외가 발생하였을 때.



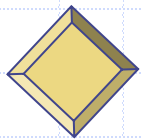
### (3) 오류(error)



◆ 프로그램 외부에서 발생하며, 프로그램에서 일반적으로 예상할 수 없고 예외 처리도 불가능하기 때문에 예외를 던지거나 잡지 않는다.

- Ex) 프로그램이 입력 파일을 열었는데 하드웨어나 시스템의 오동작으로 파일을 읽을 수 없는 경우
  - ◆ IOError가 발생
  - ◆ 프로그램에서 이 예외를 잡고(catch) 문제를 사용자에게 알리는 정도의 작업은 할 수 있겠지만, 예외 처리는 불가능하기 때문에 **스택 트레이스**를 출력하고 프로그램을 종료시키는 것이 최선의 방법

◆ 실행시점 예외와 마찬가지로 “예외 처리 또는 예외 전달의 요구사항”을 준수할 필요가 없다.



## 2. 예외 잡기 (Catching and Handling Exceptions)



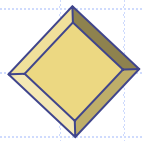
### 예외처리 개요

#### ■ 예외처리가 없는 경우

```
methodA() {  
    ....  
    methodB(); //methodB() 호출  
    ....  
}  
  
methodB() {  
    ....  
    methodC(); //methodC() 호출  
    ....  
}  
  
methodC() {  
    ....  
}
```

#### << stack trace >>

- ~ 빠른 디버깅에 필요.
- ~ 예외 처리를 제대로 작성한 코드라면 거의 대부분의 문제는 스택 트레이스 안에 답이 있다.
- ~ `e.printStackTrace()`



## 2. 예외 잡기 (Catching and Handling Exceptions)

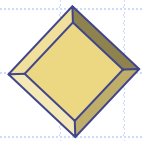
### 예외처리 개요

- 예외처리가  
있는 경우

```
methodC() throws ExceptionC{ ①
    ....
    ExceptionC e = new ExceptionC(); ②
    throw e; ③
    ....
}

methodB() throws ExceptionC{ ④
    ....
    methodC();
    ....
}

methodA() {
    ....
    try { ⑤
        ....
        methodB();
        ....
    }catch(ExceptionC e){ ⑥
        .... (최종적으로 해결하는 코드 부분)
    }
}
```



# 예외처리 없는 프로그램

## 컴파일 오류 발생

```
import java.io.*;
import java.util.Vector;
public class VectorOfIntegers {
    private Vector vec;
    private static final int initSize = 20;
    public VectorOfIntegers () {
        vec = new Vector(initSize); //①
        for (int i = 0; i < initSize; i++) { //②
            vec.addElement(new Integer(i));
        }
    }
    public void printVector() {
        PrintWriter out = new PrintWriter( //④
            new FileWriter("Vector.txt")); //③
        for (int i = 0; i < initSize; i++) {
            out.println( i + "번째 원소값은 " + vec.elementAt(i)); //⑤
        }
        out.close();
    }
    public static void main(String[] args) {
        VectorOfIntegers vc = new VectorOfIntegers();
        vc.printVector();
    }
}
```

Problems Javadoc Declaration Console

<terminated> VectorOfIntegers [Java Application] C:\Program Files\Java\jre1.8.0\_211\bin\javaw.exe (2020. 6. 11. 오후 1:28:08)

Exception in thread "main" java.lang.Error: Unresolved compilation problem:  
Unhandlled exception type [IOException](#)

at VectorOfIntegers.printVector(VectorOfIntegers.java:15)  
at VectorOfIntegers.main(VectorOfIntegers.java:25)

<<vec.addElement(new Integer()); >>

~ 레퍼 클래스의 오토박상 이용  
~ 벡터 사용 다른 방법  
vec.add(i);  
vec.add(new Integer(i));



## ◆ 2.3 try 블록

◆ 예외 직접 처리

◆ 예외를 던지는 코드(예외 발생 코드)를 try 블록으로 감싼다.

```
try {  
    code //모든 예외 던지는 프로그램 코드  
        (예외를 던지는 명령이 있음)  
}
```

## ◆ 2.3 try 블록

- ◆ 예외를 던지는 코드가 여러 개 있을 때, try 블록을 만드는 2가지 방법.

- 각각을 별도의 try 블록으로 감싸 여러 개의 try 블록을 사용. 각 try 블록 다음에 catch 블록이 나옴.

```
try { ... } catch( ) { ... }  
try { ... } catch( ) { ... }
```

- 한 개의 try 블록으로 모든 예외 던지는 코드를 감싸고 발생하는 예외 종류만큼 catch 블록 작성.

```
try {  
    code //모든 예외 던지는 프로그램 코드  
        (예외를 던지는 명령이 여러 개 있음)  
} catch( ) { ...  
} catch( ) { ... }
```

## ◆ 2.3 try 블록

### ◆ 한 개의 try 블록 사용.

- try 블록 내에서 발생한 예외를 처리하려면 try 블록 바로 다음에 catch 블록이 와야 함.

```
PrintWriter out = null;
```

```
try {
```

```
    out = new PrintWriter(new FileWriter("Vector.txt"));
```

```
    for (int i = 0; i < initSize; i++) {
```

```
        out.println( i + "번째 원소값은 " + vec.elementAt(i));
```

```
    }
```

```
}
```

<<Vector.txt >>  
~ IOException 예외 발생

<<vec.elementAt(i) >>

~ ArrayIndexOutOfBoundsException 예외 발생

## ◆ 2.4 catch 블록

- ◆ try 블록 내에서 발생하는 예외를 처리하는 예외처리 루틴은 try 블록 바로 다음에 나오는 catch 블록에서 제공
- ◆ catch 블록 구조

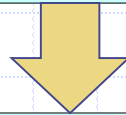
```
try {  
    ....  
} catch (Exception | Exception e) {  
    .....  
} catch (Exception | Exception e) {  
    .....  
}
```

## ◆ 2.4 catch 블록

### ◆ 멀티 catch(" | ") 지원

- catch 블록에서 처리하는 내용이 동일한 경우

```
try {  
    ....  
} catch (IOException e) { ..... }  
} catch (SQLException e) { ..... }
```

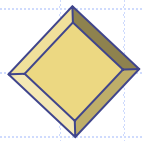


```
try {  
    ....  
} catch (IOException | SQLException e) {  
    .....  
}
```

## ◆ 2.4 catch 블록

- ◆ try 문에서 발생할 수 있는 검사 예외에 대하여 2개의 catch 블록 사용.

```
try {  
    out = new PrintWriter(new FileWriter("Vector.txt")); //1  
    for (int i = 0; i < initSize; i++) {  
        out.println( i + "번째 원소값은 " + vec.elementAt(i)); //2  
    }  
} catch (ArrayIndexOutOfBoundsException e) { //①  
    System.err.println("잡기 "  
        + "ArrayIndexOutOfBoundsException: "  
        + e.getMessage());  
} catch (IOException e) { //②  
    System.err.println("IOException 잡기: "  
        + e.getMessage());  
}
```



## 2.4 catch 블록

- ◆ Catch 블록에서 예외 처리 루틴으로 하는 일
  - 오류 메시지 출력
  - 프로그램의 수행 중단
  - 사용자에게 의사결정을 하게하여 오류 복구
  - 예외를 다시 던져 상위 예외 처리 루틴에 전달하여 오류 복구

## 2.4 catch 블록

```
public class ArrayException {  
    public static void main(String[] args) {  
        String str1, str2;  
        int sum, intval1, intval2;  
        try { //①  
            str1 = args[0]; //② main() 메소드의 매개변수(문자열 배열)  
            str2 = args[1];  
            intval1 = Integer.parseInt(str1); //③  
            intval2 = Integer.parseInt(str2);  
            sum = intval1 + intval2;  
            System.out.println("두 수의 합 = "+sum);  
        } catch (ArrayIndexOutOfBoundsException | NumberFormatException e)  
        { //④  
            System.out.println("배열 인덱스의 수가 부족하거나 숫자로 변환할 수  
없는 예외 발생");  
        } catch (Exception e) { //⑤  
            System.out.println("알수 없는 예외가 발생");  
        }  
    }  
}
```

Problems @ Javadoc Declaration Console  
<terminated> ArrayException [Java Application] C:\WProgram F  
배열 인덱스의 수가 부족하거나 숫자로 변환할 수 없는 예외 발생

Problems @ Javadoc Declaration Console  
<terminated> ArrayException [Java Application] C:\WProgram F  
배열 인덱스의 수가 부족하거나 숫자로 변환할 수 없는 예외 발생

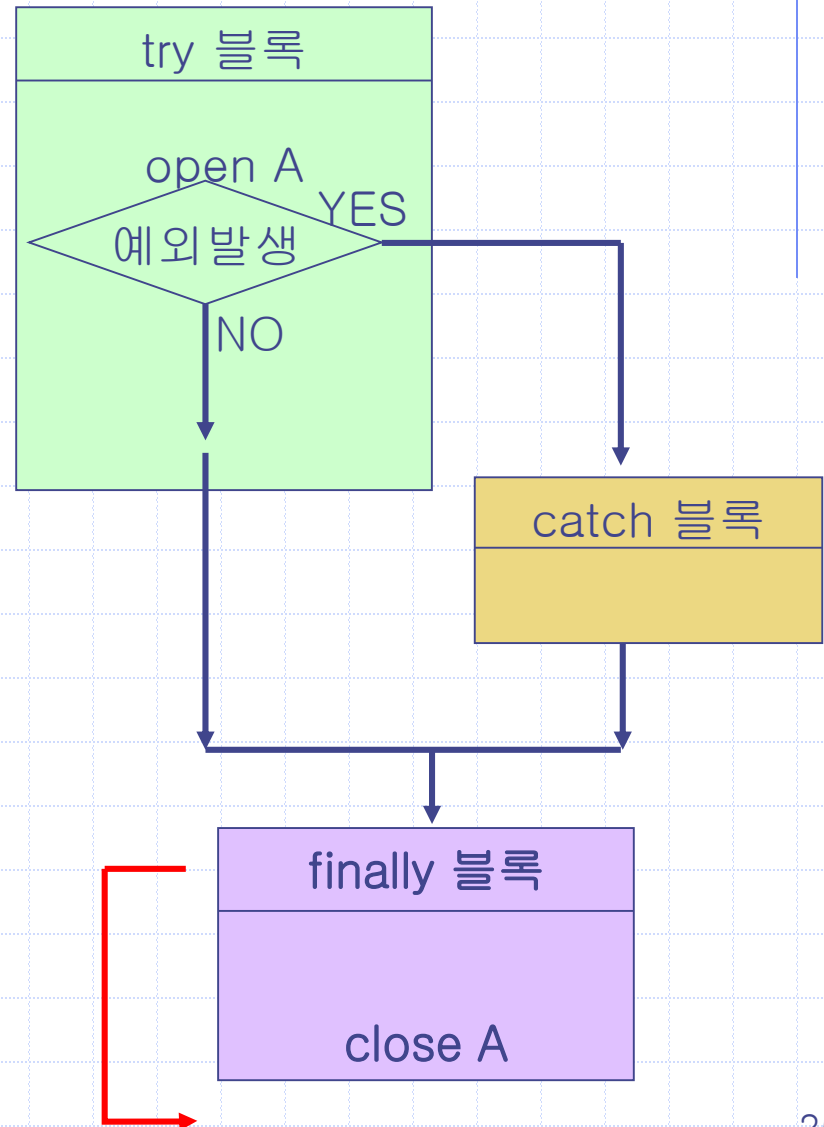
Problems @ Javadoc Declaration Console  
<terminated> ArrayException [Java Application] C:\WProgram F  
두 수의 합 = 12

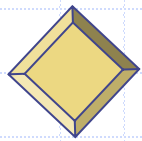


## ◆ 2.5 finally 블록

◆ try 블록 수행 후에 또는 catch 블록 수행 후에 반드시 수행되는 코드 블록이다.

- 외의 발생 여부와 무관하게 공통적으로 또는 반드시 수행되어야 하는 코드 추가



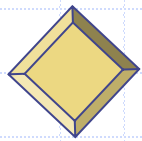


## 2.5 finally 블록



- ◆ out.close() 코드가 중복되거나, 필요한 곳에 없음.

```
try {  
    .....  
    out.close();    // 중복된 코드  
} catch (ArrayIndexOutOfBoundsException e) {  
    System.err.println("잡기 "  
        + "ArrayIndexOutOfBoundsException: "  
        + e.getMessage());  
    out.close();    // 중복된 코드  
} catch (IOException e) {  
    System.err.println("IOException 잡기: "  
        + e.getMessage());  
    //필요한 곳  
}
```

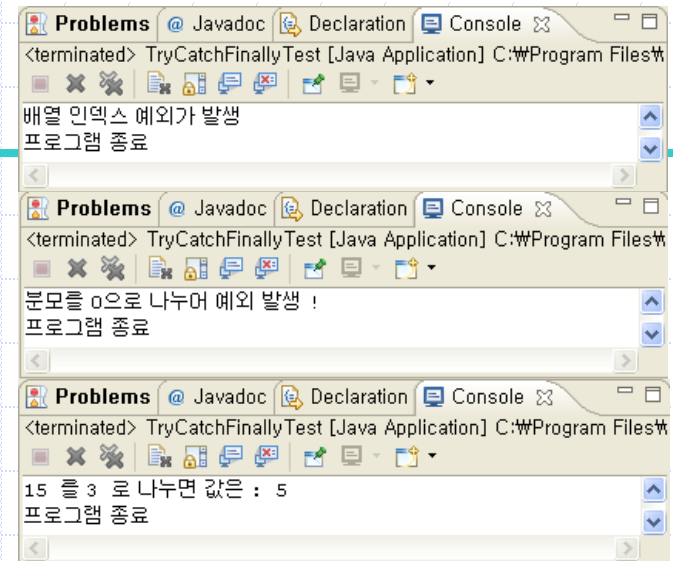


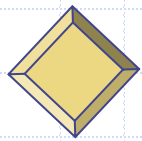
## 2.5 finally 블록

```
try {  
    .....  
} catch (ArrayIndexOutOfBoundsException e) {  
    System.err.println("잡기 "  
        + "ArrayIndexOutOfBoundsException: "  
        + e.getMessage());  
} catch (IOException e) {  
    System.err.println("IOException 잡기: " + e.getMessage());  
}finally {  
    if (out != null) { ①  
        System.out.println("PrintWriter를 닫음");  
        out.close(); ②  
    } else { ③  
        System.out.println("PrintWriter가 열려 있지 않음");  
    }  
}
```

## 2.5 finally 블록

```
public class TryCatchFinallyTest {  
    public static void main(String[] args) {  
        int x, y, sum;  
        try {  
            x = Integer.parseInt(args[0]); ①  
            y = Integer.parseInt(args[1]); ①  
            sum = x / y; ②  
            System.out.println(x + " 를 " + y + " 로 나누면 값은 : " + sum);  
        } catch (ArithmeticException e) { ③  
            System.out.println("분모를 0으로 나누어 예외 발생 !");  
        } catch (Exception e) { ④  
            System.out.println("배열 인덱스 예외가 발생");  
        } finally { ⑤  
            System.out.println("프로그램 종료");  
        }  
    }  
}
```





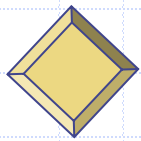
## 2.6 다중 리소스 닫기

### ◆ try-with-resources

#### ◆ 리소스(자원)와 함께 처리하는 try 문장.

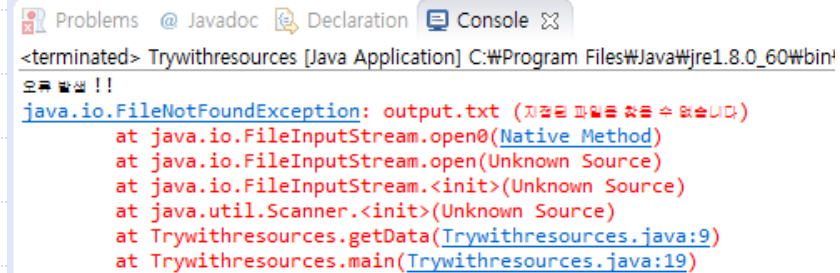
- 예외 발생 여부와 관계없이 사용했던 리소스 객체인 입출력 스트림, 서버 소켓, 채널 등의 **close()** 메소드를 자동으로 호출하여 리소스를 안전하게 닫는다.
- ◆ **AutoCloseable** 인터페이스가 추가되어 **try-with-resources**를 사용할 때에는 이 인터페이스를 구현한 클래스는 별도로 **close()** 메소드를 호출할 필요가 없다.
- ◆ 파일 및 데이터베이스와 같은 리소스를 사용하는 경우에는 반드시 예외처리를 해야 하며, 다음과 같이 **try** 문에 사용하려는 리소스를 명시할 수 있다.

```
try (리소스) {  
    ....  
}
```

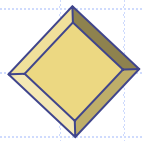


## 2.6 다중 리소스 닫기

```
import java.util.*;
import java.io.*;
public class Trywithresources {
    public void getData(String fileName) {
        try(Scanner scanner = new Scanner(new File(fileName))) { //①
            System.out.println(scanner.nextLine());
        } catch (IllegalArgumentException | FileNotFoundException |
        NullPointerException e) { //②
            System.out.println("오류 발생 !!");
            e.printStackTrace(); //③
        }
    }
    public static void main(String[] args) {
        String f = "output.txt";
        Trywithresources twr = new Trywithresources();
        twr.getData(f);
    }
}
```



```
Problems @ Javadoc Declaration Console
<terminated> Trywithresources [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\
오류 발생 !!
java.io.FileNotFoundException: output.txt (지정된 파일을 찾을 수 없습니다)
    at java.io.FileInputStream.open0(Native Method)
    at java.io.FileInputStream.open(Unknown Source)
    at java.io.FileInputStream.<init>(Unknown Source)
    at java.util.Scanner.<init>(Unknown Source)
    at Trywithresources.getData(Trywithresources.java:9)
    at Trywithresources.main(Trywithresources.java:19)
```



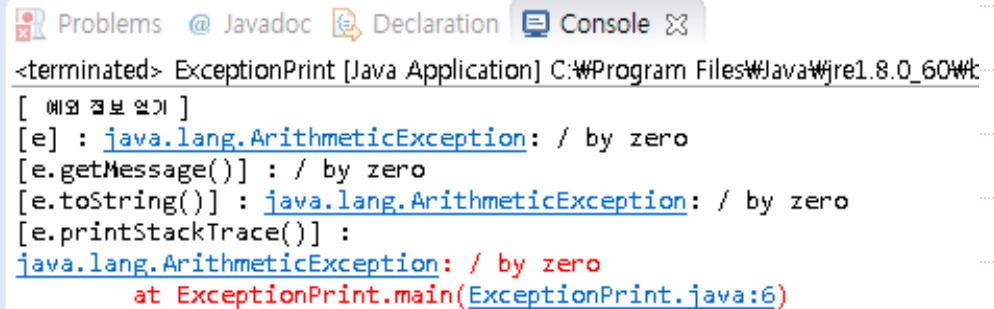
## 2.7 예외 정보 얻기

- ◆ try 블록에서 예외가 발생하면 예외 객체(e)는 catch 블록의 매개변수에서 참조하며, catch 블록의 매개변수를 이용하면 예외 객체의 정보를 알 수가 있다.
  - ◆ 예외 정보를 얻는 방법은 Exception 클래스의 메소드를 사용하는데, 가장 많이 사용되는 메소드는 toString(), getMessage(), printStackTrace()가 있다.
  - ◆ 예외 메시지를 출력하는 경우 - toString(), getMessage().
  - ◆ 예외 발생 코드를 추적해서 단계별로 오류를 보여주는 내용을 출력하는 경우 - printStackTrace().
  - ◆ 사용 방법

```
System.out.println(e.toString());  
System.out.println(e.getMessage());  
e.printStackTrace();
```

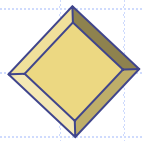
## ◆ 2.7 예외 정보 얻기

```
public class ExceptionPrint {  
    public static void main(String[] args) {  
        int div;  
        try {  
            div = 10/0; //예외 발생  
            System.out.println("결과 출력 : " + div);  
        } catch (Exception e) {  
            System.out.println("[ 예외 정보 얻기 ]");  
            System.out.println("[e] : " + e); //①  
            System.out.println("[e.getMessage()] : " + e.getMessage()); //②  
            System.out.println("[e.toString()] : " + e.toString()); //③  
            System.out.println("[e.printStackTrace()] : ");  
            e.printStackTrace(); //④  
        }  
    }  
}
```



```
Problems @ Javadoc Declaration Console X  
<terminated> ExceptionPrint [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\java.exe  
[ 예외 정보 얻기 ]  
[e] : java.lang.ArithmeticException: / by zero  
[e.getMessage()] : / by zero  
[e.toString()] : java.lang.ArithmeticException: / by zero  
[e.printStackTrace()] :  
java.lang.ArithmeticException: / by zero  
    at ExceptionPrint.main(ExceptionPrint.java:6)
```



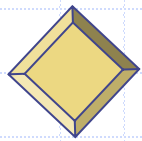


## 2.8 프로그램의 흐름

◆ 예외 발생 여부, 예외 발생 종류에 따라 프로그램의 흐름의 변화 확인.

- 3가지 경우에 모두 공통으로 수행되는 제일 앞에 있는 코드 부분

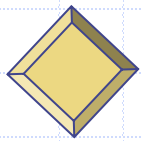
```
import java.io.*;
import java.util.Vector;
public class VectorOfIntegers {
    private Vector vec;
    private static final int initSize = 20;
    public VectorOfIntegers () {
        vec = new Vector(initSize);
        for (int i = 0; i < initSize; i++) {
            vec.addElement(new Integer(i));
        }
    }
    . . . . .
```



## 2.8 프로그램의 흐름

예외가 발생하지 않는  
경우의 프로그램의 흐름

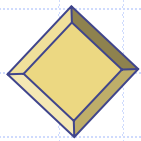
```
public void printVector() {  
    PrintWriter out = null;  
    try {  
        System.out.println("try 블록 진입");  
        out = new PrintWriter(  
            new FileWriter("Vector.txt"));  
  
        for (int i = 0; i < initSize; i++)  
            out.println( i + "번째 원소값은 "  
                + vec.elementAt(i));  
  
    } catch (ArrayIndexOutOfBoundsException e) {  
        System.err.println("잡기 "  
            + "ArrayIndexOutOfBoundsException: "  
            + e.getMessage());  
    }  
  
    catch (IOException e) {  
        System.err.println("IOException 잡기: "  
            + e.getMessage());  
    }  
  
    finally {  
        if (out != null) {  
            System.out.println("PrintWriter를 닫음");  
            out.close();  
        }  
        else {  
            System.out.println("PrintWriter가 열려 있지 않음");  
        }  
    }  
}
```



## 2.8 프로그램의 흐름

IOException 예외가  
발생하는 경우의  
프로그램의 흐름

```
public void printVector() {  
    PrintWriter out = null;  
    try {  
        System.out.println("try 블록 진입");  
        out = new PrintWriter(  
            new FileWriter("Vector.txt")); //try 블록 빠져나감  
  
        for (int i = 0; i < initSize; i++)  
            out.println( i + "번째 원소값은 "  
                + vec.elementAt(i)/" 출구 2*");  
    }  
  
    catch (ArrayIndexOutOfBoundsException e) {  
        System.err.println("잡기 "  
            + "ArrayIndexOutOfBoundsException: "  
            + e.getMessage());  
    }  
  
    catch (IOException e) {  
        System.err.println("IOException 잡기: "  
            + e.getMessage());  
    }  
  
    finally {  
        if (out != null) {  
            System.out.println("PrintWriter를 닫음");  
            out.close();  
        }  
        else {  
            System.out.println("PrintWriter가 열려 있지 않음");  
        }  
    }  
}
```



## 2.8 프로그램의 흐름

ArrayIndexOutOfBoundsException  
예외가 발생하는  
경우의 프로그램 흐름

```
public void printVector() {
    PrintWriter out = null;
    try {
        System.out.println("try 블록 진입");
        out = new PrintWriter(
            new FileWriter("Vector.txt"));

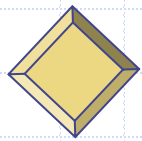
        for (int i = 0; i < initSize; i++)
            out.println( i + "번째 원소값은 "
                + vec.elementAt(i)/try 블록 빠져나감*/);

    } catch (ArrayIndexOutOfBoundsException e) {
        System.err.println("잡기 "
            + "ArrayIndexOutOfBoundsException: "
            + e.getMessage());

    } catch (IOException e) {
        System.err.println("IOException 잡기: "
            + e.getMessage());

    }

    finally {
        if (out != null) {
            System.out.println("PrintWriter를 닫음");
            out.close();
        }
        else {
            System.out.println("PrintWriter가 열려 있지 않음");
        }
    }
}
```

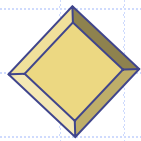


### 3. 메소드에서 던지는 예외 알리기 (throws)

- ◆ 메소드 내에서 발생한 예외를 그 메소드에서 처리하지 않고 호출 스택의 하위에 있는 다른 메소드들이 처리하게 하는 방법
- ◆ 예외의 간접처리(throws 문)
  - 즉, printVector를 호출한 메소드들이 처리
  - catch 블록을 작성하지 않고, 메소드 선언 부분에 자신이 던지는 예외를 알려야 한다.

예외처리를  
하지않음

```
public void printVector() {  
    PrintWriter out = new PrintWriter(new FileWriter("Vector.txt"));  
    for (int i = 0; i < initSize; i++) {  
        out.println( i +"번째 원소값은 " + vec.elementAt(i));  
    }  
    out.close();  
}
```



### 3. 메소드에서 던지는 예외 알리기

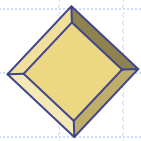
#### ◆ printVector가 2개의 예외를 던지는 방법

- 메소드 선언 부분에 throws 문을 추가
- throws 구문의 사용법은 throws 키워드 다음에 예외들을 “,”로 구분하여 나열

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

**ArrayIndexOutOfBoundsException** 예외는 비  
검사 예외이기 때문에  
**throws** 문에 포함시키지  
않아도 됨

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

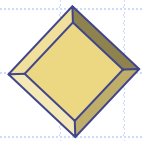


### 3. 메소드에서 던지는 예외 알리기

```
public class ThrowsExam {  
    public static void calc2() throws ArithmeticException{  
        int num = 5/0;  
    }  
    public static void calc1() throws ArithmeticException{  
        calc2();  
    }  
    public static void main(String[] args) {  
        System.out.println("메소드에서 던지는 예외알리기");  
        try{  
            calc1();  
        }catch(ArithmeticException e) {  
            System.out.println(e.getMessage());  
        }finally {  
            System.out.println("메소드에서 던지는 예외알리기 프로그램 종료");  
        }  
    }  
}
```

Problems @ Javadoc Declaration

<terminated> ThrowsExam [Java Application] C:\  
/ by zero  
메소드에서 던지는 예외알리기 종료



## 4. 예외 던지기(throw)

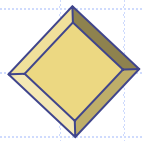


### throw 문

- 어떤 상황에서 일부러 예외를 발생시키고 싶을 경우에 사용
- 사용 예) throw anExceptionObject;
- popUp() 메소드

```
public Object popUp() throws EmptyStackException {  
    Object obj;  
    if (size == 0) {  
        throw new EmptyStackException(); //size가 0 이면 예외 발생  
    }  
    obj = objectAt(size - 1);  
    setObjectAt(size - 1, null);  
    size--;  
    return obj;  
}
```





## 4. 예외 던지기(throw)

```
public class ThrowExm {
    static int pnum= 70;
    static int inum=40;
    public static void calc(int num) throws Exception {
        if( num < pnum ) {
            throw new Exception( "num 값인 " + num + " 이 " + pnum + " 보
다 작다.");
        }
    }
    public static void main(String[] args) {
        System.out.println("예외 던지기");
        try{
            calc(inum);
        }catch(Exception e){
            System.out.println(e.getMessage());
        }finally{
            System.out.println("예외 던지기 프로그램 종료");
        }
    }
}
```

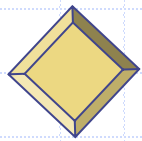
Problems @ Javadoc Decla

<terminated> ThrowExam [Java Appli

[ 예외 던지기 ]

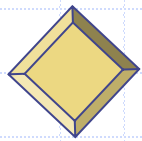
num 값인 40 이 70 보다 작다.

예외 던지기 프로그램 종료 !!



## 5. 예외 만들기

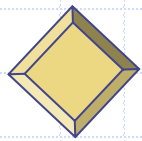
- ◆ 개발자들이 자신의 메소드에서 발생할 수 있는 문제 (예외)를 위한 **사용자 정의 예외 클래스**를 만드는 방법
  - 앞의 모든 예외 클래스들은 자바에서 제공하는 예외임.
  - 사용자 정의 예외 클래스는 Exception 클래스로부터 상속받는다.
  - 패키지 개발자는 다른 개발자들이 사용할 수 있도록 패키지 전체에서 발생할 수 있는 예외 클래스들을 체계적으로 작성하는 것이 좋다.
  - 개발자가 작성한 메소드에서 자바가 제공하는 예외뿐 아니라 사용자 정의 예외 클래스에서 생성된 예외 객체를 던질 수 있다.



## 5.1 사용자 정의 예외 만들기

- ◆ 사용자 정의 예외 클래스는 Exception 클래스나 서브클래스로부터 상속 받음.
- ◆ 사용자 정의 예외 클래스에 2개의 생성자 정의.
  - 매개변수가 없는 생성자
  - 발생한 문제를 설명하는 스트링 객체를 매개변수 한 개로 갖는 생성자
- ◆ Ex) 사용자 정의 예외 클래스인 XxxException 클래스.
  - XxxException은 Exception의 서브클래스로 정의하였으며 2개의 생성자를 갖는다.

```
public class XxxException extends [ Exception | RuntimeException ] {  
    public XxxException( ) { }  
    public XxxException(String msg) {  
        super(msg); //super class의 생성자를 호출한다  
    }  
}
```



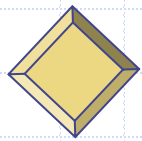
## 5.2 사용자 정의 예외 던지기

- ◆ 사용자 정의 예외를 던지려면 먼저 예외 클래스에서 new 연산자로 예외 객체를 생성.
- ◆ XxxException 클래스에서 예외 객체를 만들어 던지는 코드이다. 2개의 다른 생성자를 사용하여 예외 객체를 생성.

```
(1) XxxException me = new XxxException(); //첫 번째 생성자 사용
    throw me;
(2) //두 번째 생성자 사용
    XxxException me = new XxxException("Exception : wrong format");
    throw me;
```

- ◆ 예외를 던지는 코드를 포함하는 메소드는 메소드 선언부에 throws 문 포함.
  - Exception 클래스로부터 상속받는 예외 객체를 던지려면 반드시 메소드 선언부에 throws 문을 사용.

```
public void myMethod() throws XxxException {
    .....
    XxxException me = new XxxException();
    throw me;
    .....
}
```



# 사용자 정의 예외 예제-1

```
class MyException extends Exception { //사용자 정의 예외 클래스
```

```
    public MyException() { }
```

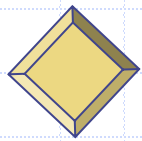
```
    public MyException(String msg) {  
        super(msg);  
    }  
}
```

```
-----  
public class MyExceptionTest {
```

```
    public void myMethod() throws MyException {  
        MyException me = new MyException("Exception : wrong format");  
        throw me;  
    }
```

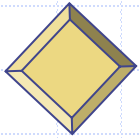
```
    public static void main(String[] args) {  
        MyExceptionTest mE = new MyExceptionTest();  
        try {  
            mE.myMethod();  
        } catch (MyException me) {  
            me.printStackTrace();  
        }  
    }  
}
```

```
Problems @ Javadoc Declaration Console  
<terminated> MyExceptionTest [Java Application] C:\Program Files\Java\jre-1  
MyException: Exception : wrong format  
    at MyExceptionTest.myMethod(MyExceptionTest.java:4)  
    at MyExceptionTest.main(MyExceptionTest.java:11)
```



## 6. 예외를 처리하는 3가지 방법

- 예외가 발생한 메소드에서 처리
  - try, catch 문으로 처리
- 호출 스택의 메소드로 전달하기
  - 메소드 선언부에 throws문 사용
- 예외가 발생한 메소드에서 처리하고 전달하기
  - try, catch와 throws문 모두 사용



## 예외가 발생한 메소드에서 처리하고 전달하기

```
public void responsibleExceptionMethod() throws IOException { //④
    MessageReader mr=new MessageReader();
    try { //①
        mr.loadHeader();
    }catch(IOException e) {
        ..... //IOException 처리하기 //②
        throw e; //③
    }
}
```

# ◆ 실습문제



◆ 1, 2, 3번 실습



# ◆ 연습문제



◆ 1)

◆ 2)

◆ 3)

◆ 4)번 과제