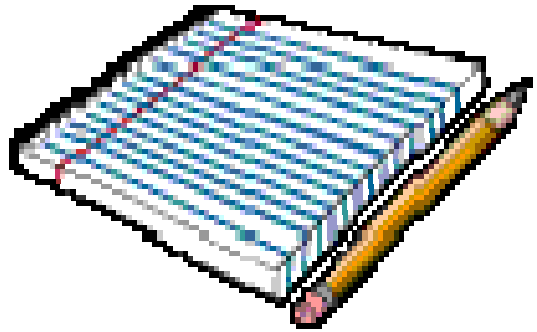
[JAVA] 1. 예외 처리

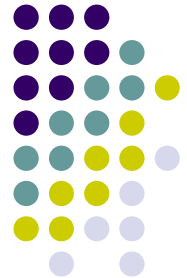
충남대학교
컴퓨터공학과



학습 내용

1. 예외처리란?
2. 예외처리 구문
3. 예외 클래스
4. 예외 던지고 받기
5. 사용자 정의 예외
6. 다형성과 예외

1. 예외 처리란?

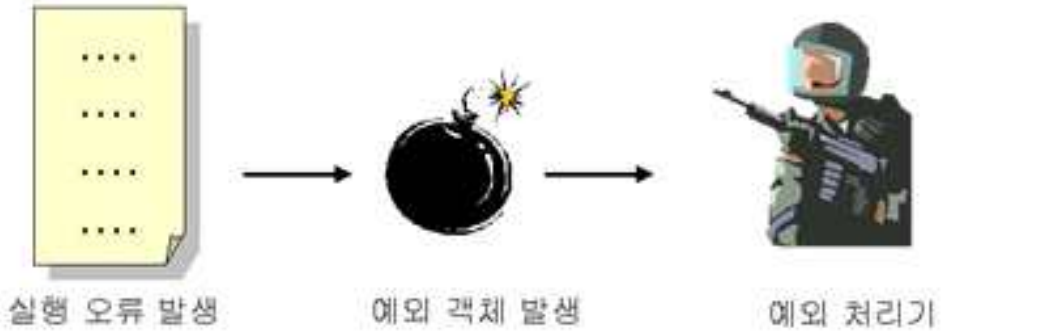


- 예외(Exception) : 예상치 못한 상황
  - 파일이 없는 경우
  - 서버가 다운되는 경우
  - 장치를 사용할 수 없는 경우
  - 나이를 입력하라고 했는데, 0보다 작은 값이 입력되는 경우
  - 나눗셈을 위한 두개의 정수를 입력받는데, 제수로 0이 입력됨
  - 주민등록번호 13자리만 입력하라고 했더니 중간에 '-'가 포함됨

- 이런 예외적인 상황을 처리하지 않으면?
  - 프로그램이 종료됨
  - 프로그램에서 예외를 감지하여 예외에 맞는 처리를 하도록 하는 것이 예외 처리(excpetion handling)임

자바에서의 예외

- 자바에서는 예외도 하나의 객체로 취급
- 메소드 실행 중에 오류가 발생하면 메소드는 예외를 객체로 만들어 호출한 곳으로 전달 – 예외 객체(exception object)
- 예외 객체에는 오류 정보, 오류의 타입과 오류 발생 시 프로그램 상태 정보 등이 포함됨



예외 발생 예

```
public class DivideByZero {
    public static void main(String[] args)
    {
        int x = 1;
        int y = 0;
        int result = x / y;      // 예외 발생!
        System.out.println("이 문장이 출력될까요?");
    }
}
```

실행되지 않음!

실행결과

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ExceptionTest.main(ExceptionTest.java:5)
```

예외 이름

2. 예외처리 구문

- 지금까지 If문을 이용한 예외처리를 해옴
  - 이는 if문이 프로그램의 주 흐름인지, 아니면 예외의 처리인지 구분이 되지 안된다는 단점

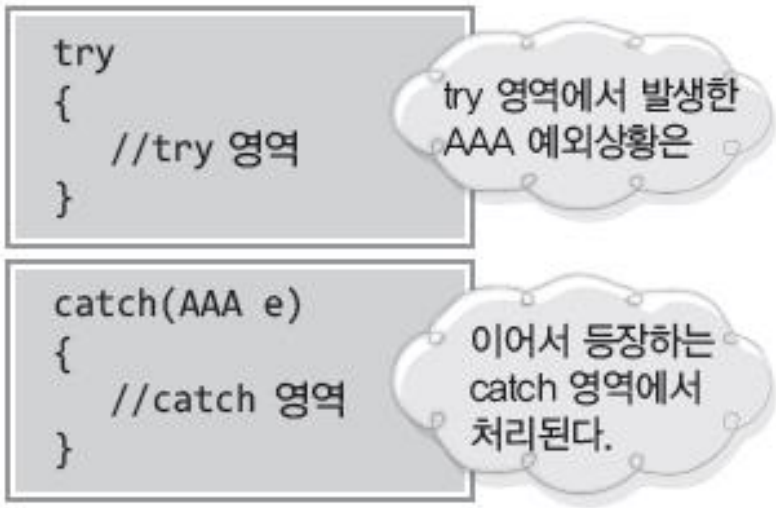
```
System.out.print("피제수 입력 : ");
int num1=keyboard.nextInt();

System.out.print("제수 입력 : ");
int num2=keyboard.nextInt();

if(num2==0)
{
    System.out.println("제수는 0이 될 수 없습니다.");
    i-=1;
    continue;
}
```

try/catch 블록

- 예외를 처리할 것임을 알려주기 위한 용도로 쓰이는 구문
  - try - 예외발생의 감지 대상을 감싸는 목적으로 사용
  - catch - 발생한 예외상황의 처리를 위한 목적으로 사용
  - try 블록과 catch 블록은 독립된 블록 – try 블록에서 정의된 변수는 catch 블록에서 사용될 수 없음



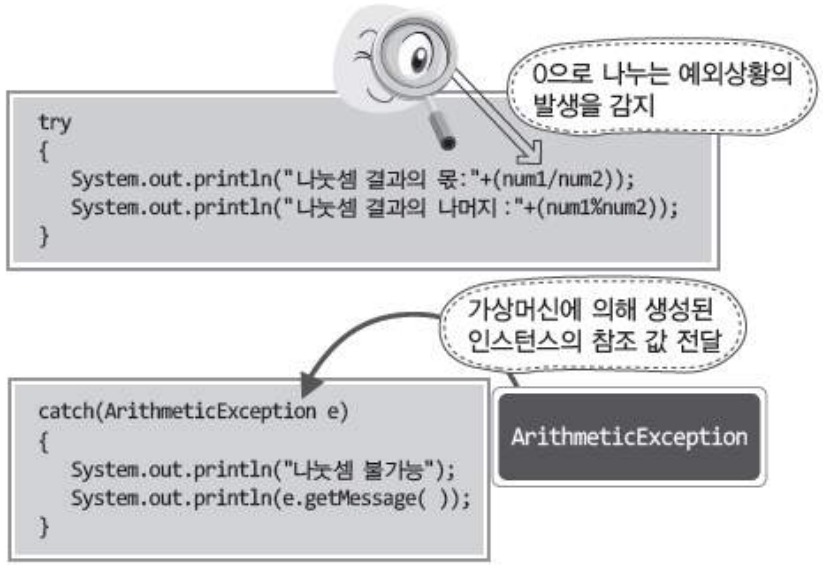
try/catch 블록의 예외처리 과정

```
try
{
   System.out.println("나눗셈 결과의 몫: "+( num1/num2 ));
   System.out.println("나눗셈 결과의 나머지: "+(num1%num2));
}
catch(ArithmeticException e)
{
   System.out.println("나눗셈 불가능");
   System.out.println(e.getMessage());
}
System.out.println("프로그램을 종료합니다.");
```

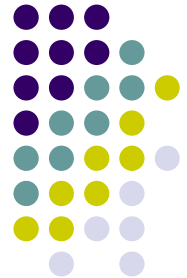
1. 예외발생

2. 참조 값 전달하면서 catch 영역실행

3. catch 영역실행 후, try~catch 다음 문장을 실행

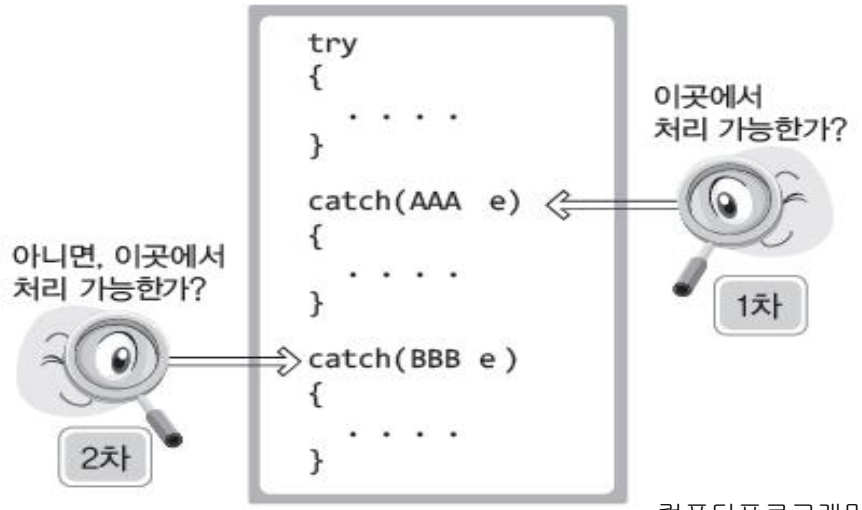


catch 블록이 여러개인 경우



- 예외의 종류에 따라 여러 개의 catch 블록 정의 가능
- 발생한 예외의 종류와 일치하는 catch 블록만 실행됨
- 일치되는 catch 블록이 없으면 발생된 예외는 처리되지 않음

```
try
{
    . . . .
}
catch(Throwable e)
{
    . . . .
}
catch(ArithmeticException e)
{
    . . . .
}
```

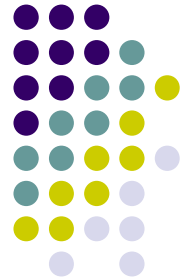


Finally 블록

- finally와 연결되어 있는 try 블록으로 일단 진입을 하면, 무조건 실행되는 영역
- 중간에 return 문을 실행하더라도 finally 블록이 실행된 다음에 메소드를 빠져나감

```
try
{
    int result=num1/num2;
    System.out.println("나눗셈 결과는 "+result);
    return true;
}
catch(ArithmeticException e)
{
    System.out.println(e.getMessage());
    return false;
}
finally
{
    System.out.println("finally 영역 실행");
}
```

try/catch와 관련된 규칙



1. try 없이 catch나 finally만 쓸 수는 없음

```
void go() {
    Foo f = new Foo();
    f.foof();
    catch(FooException ex) { }
}
```

2. try와 catch 사이에 코드를 집어넣을 수 없음

```
try {
    x.doStuff();
}
int y = 43;
} catch(Exception ex) { }
```

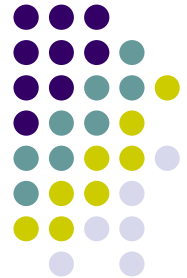
try/catch와 관련된 규칙(cont')

3. try 뒤에는 반드시 catch나 finally가 있어야 함

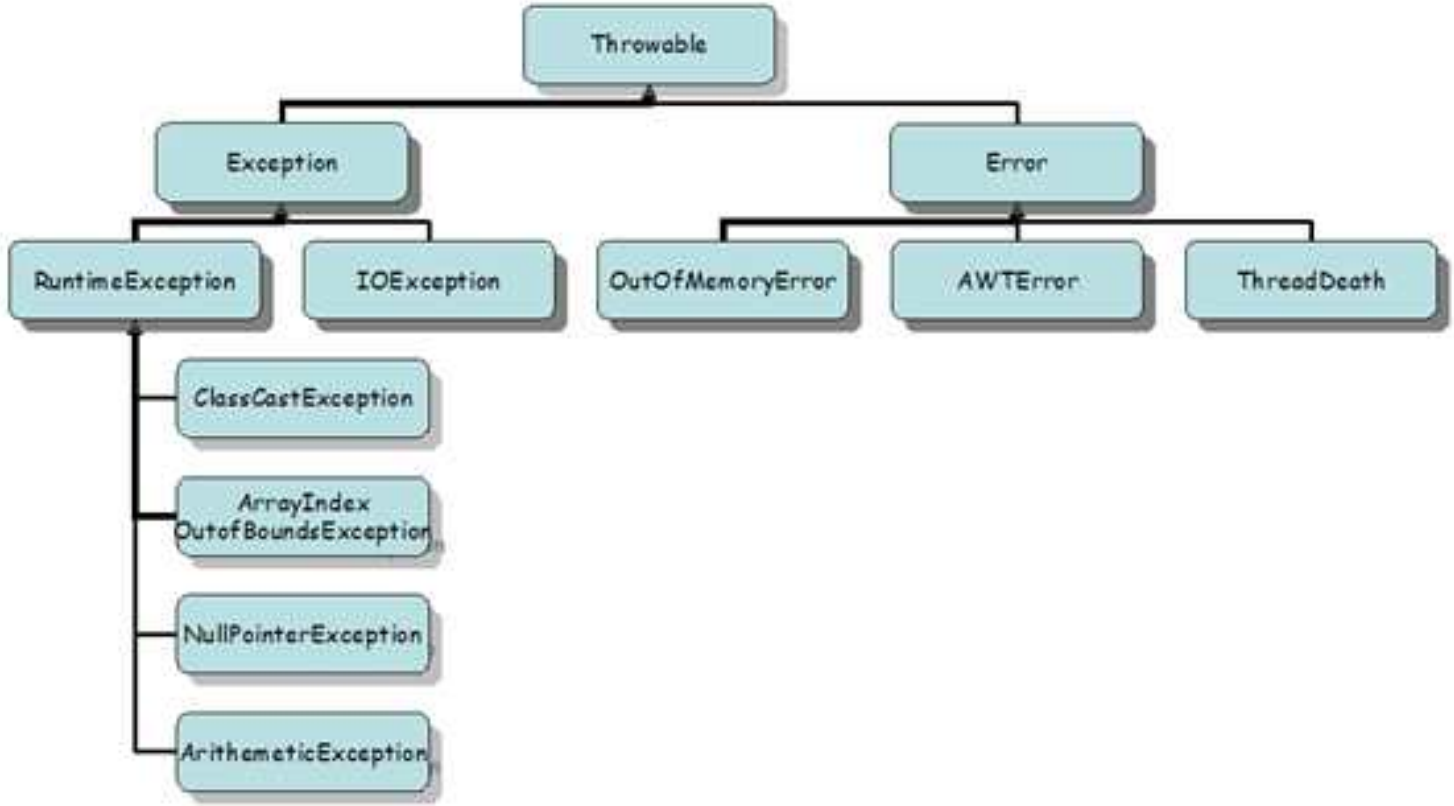
```
try {
    x.doStuff();
} finally { }
```

catch 블록이 없어도 finally가 있으면 문법적으로 문제가 없음. 하지만 try만 달랑 있는 것은 안됨

3. 예외(Exception) 클래스



- 예외 관련 클래스 계층도



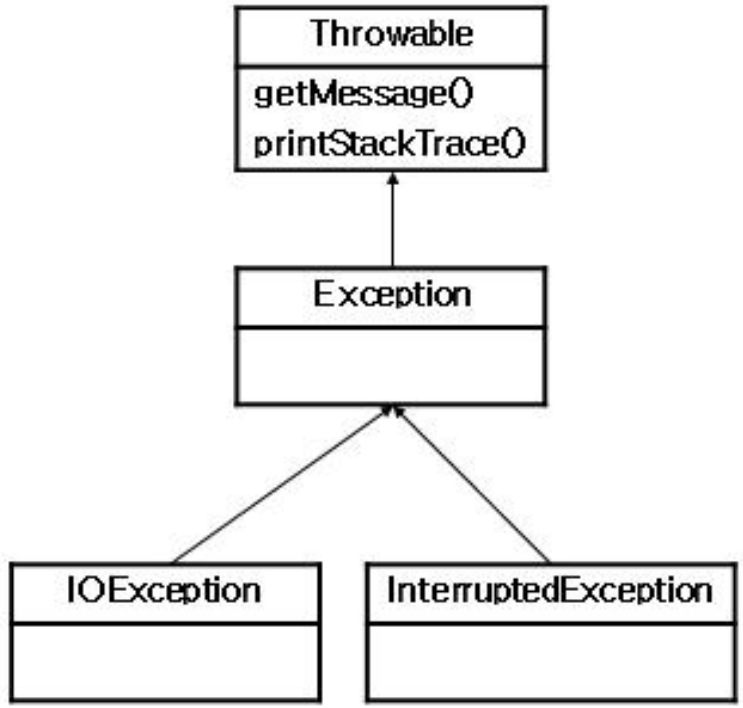
Error 클래스

- 시스템에서 발생하는 심각한 에러로 치명적인 내부 오류나 JVM내의 자원의 고갈을 나타냄
- Error 클래스는 try~catch로 처리가 불가능한 예외로 Error가 발생하면 프로그램이 종료됨
- Error 클래스를 상속하는 예외 클래스는 프로그램 내에서 해결이 불가능한 치명적인 예외 상황을 알리는 예외 클래스의 정의에 사용됨
- 우리의 관심사는 Error 클래스가 아닌, Exception 클래스

Exception 클래스

- 프로그램 실행 시에 흔히 발생할 수 있는 일반적인 예외

```
try {
    // 위험한 일 처리
} catch (Exception ex) {
    // 문제 해결
}
```



- 주요 메소드
  - getMessage()
    - 예외가 발생한 원인정보를 문자열 형태로 반환
  - printStackTrace()
    - 예외 발생 당시 스택의 내용을 출력

```
java.lang.NullPointerException
        at MyClass.mash(MyClass.java:9)
        at MyClass.crunch(MyClass.java:6)
        at MyClass.main(MyClass.java:3)
```

RuntimeException 및 IOException

- RuntimeException
  - 프로그램 실행 중에 어느 때라도 발생할 수 있는 예외로 Error를 상속하는 예외 클래스만큼 치명적인 예외상황의 표현에 사용되지 않는다.
  - RuntimeException계열의 예외 클래스들은 try~catch문을 통해서 처리하기도 하나, Error 클래스를 상속하는 예외 클래스와 마찬가지로, try~catch문 또는 throws 절을 명시하지 않아도 된다.
  - 이들이 명시하는 예외의 상황은 프로그램의 종료로 이어지는 것이 자연스러운 경우가 대부분이기 때문이다.

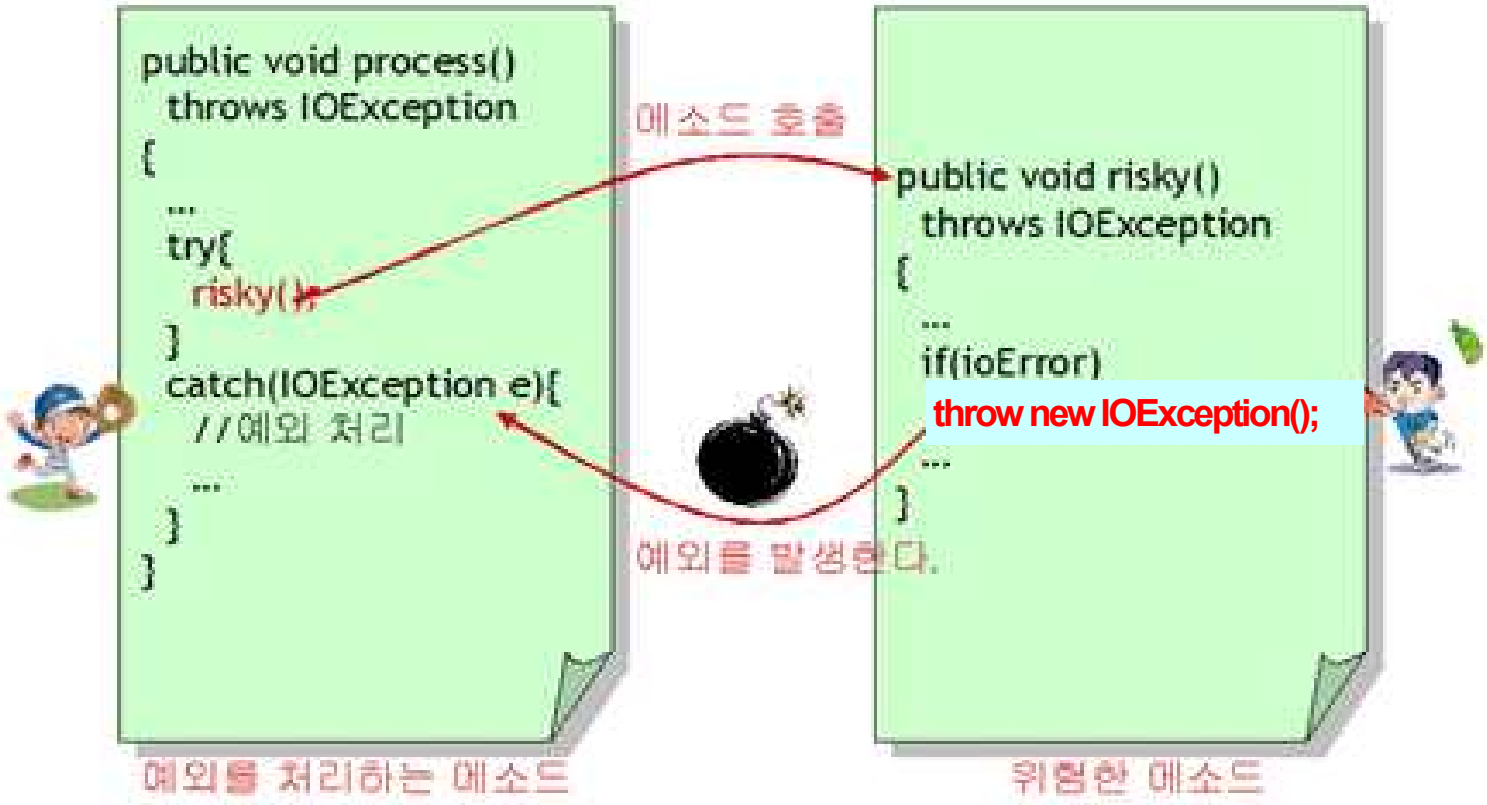
- IOException
  - 입출력과 관련된 예외

RunTimeException 서브 클래스들

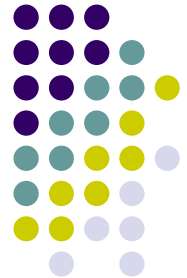
예외	설명
ArithmeticException	어떤 수를 0으로 나눌 때 발생한다.
NullPointerException	널 객체를 참조할 때 발생한다.
IncompatibleClassChangeException	클래스 내부의 변수의 선언이 static에서 non-static 또는 반대로 변경되었는데 다른 클래스가 이 변수를 참조하는 경우
ClassCastException	적절치 못하게 클래스를 형변환하는 경우
NegativeArraySizeException	배열의 크기가 음수값인 경우
OutOfMemoryException	사용 가능한 메모리가 없는 경우
NoClassDefFoundException	원하는 클래스를 찾지 못하였을 경우
IncompatibleTypeException	인터페이스의 인스턴스를 생성하려고 시도하는 경우
ArrayIndexOutOfBoundsException	배열을 참조하는 인덱스가 잘못된 경우
UnstatisfiedLinkException	자바가 아닌 다른 프로그램 언어로 작성된 메소드를 연결하는데 실패했을 경우

4. 예외 던지고 받기

- throw를 이용하여 예외객체 던지기



throw와 throws 키워드



- throw : 예외 객체를 생성하여 던질 때 사용
- throws : 어떤 메소드가 발생시킬 수 있는 예외를 나열할 때 사용

```
void allocateMemory() throws AllocateMemoryFailException
{
    ...
    if(memory == null)
        throw new AllocateMemoryFailException();
    ...
}
```

예외 클래스가 일치하여야 함

예외와 메소드

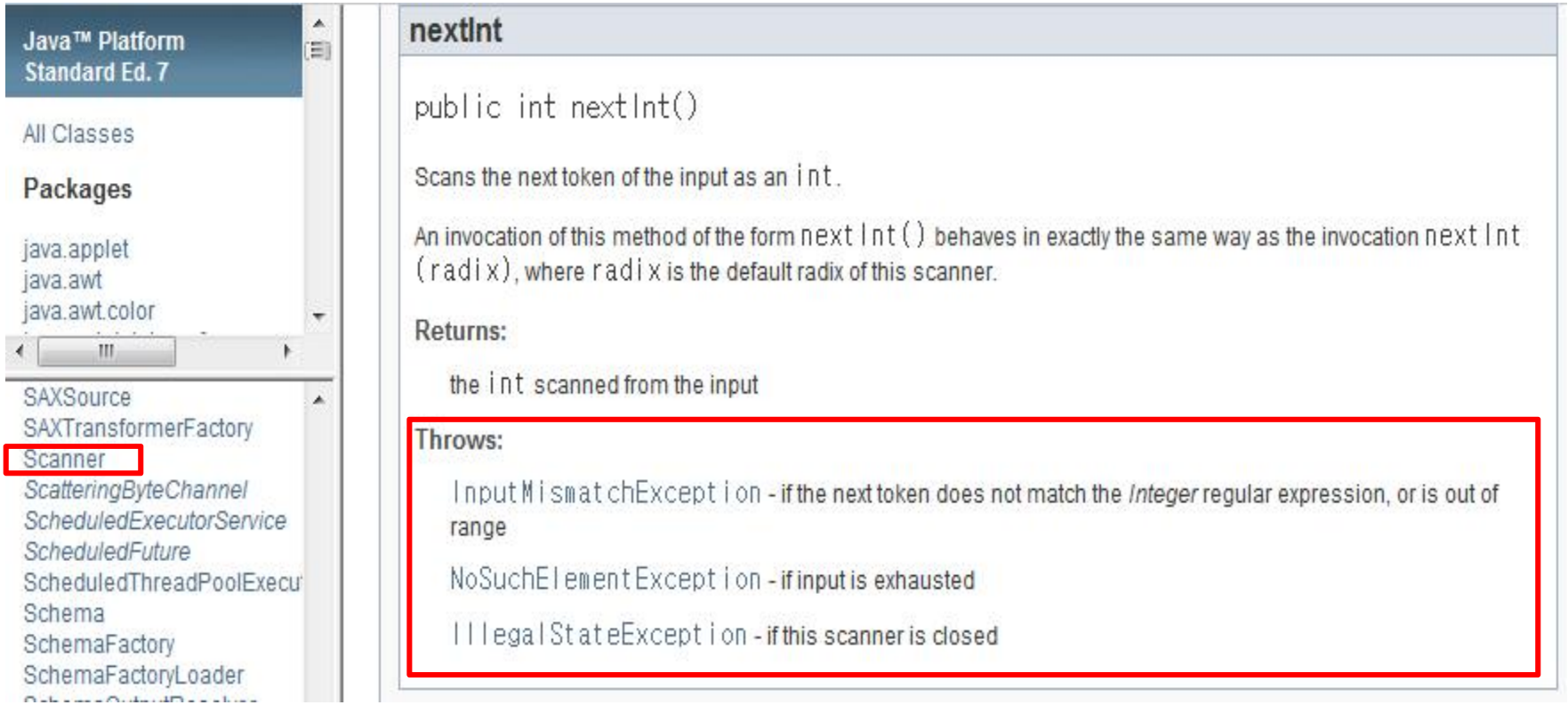
- 예외를 발생시킬 수 있는 메소드는 자신을 호출한 메소드에게 예외가 발생하였다는 것을 알려줘야 호출 메소드가 예외를 처리할 수 있는 코드를 준비할 수 있음
- 예외 발생 메소드 정의

```
int method(int n) throws a, b
{
        ...
}
```

  - 메소드 method가 a와 b라는 예외를 발생시킬 수 있음을 나타냄
  - 메소드안에서 발생되는 모든 예외를 나열해야 하는 것은 아님
  - Error와 RuntimeException은 열거될 필요가 없다.

자바 API와 예외

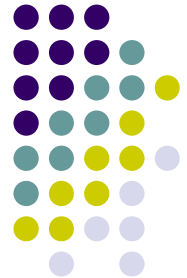
- 자바 API 클래스의 메소드 설명에 발생가능한 예외가 표시
  - 메소드를 호출할 때, 이 예외를 처리하는 코드를 작성해야 함



메소드에서 발생한 예외 처리 방법

- 메소드를 호출한 곳에서 처리하는 2가지 방법
  - 예외를 자신이 직접 try/catch로 처리하는 방법
  - 예외를 상위 메소드로 전달하는 방법

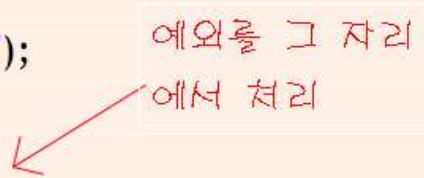
예외를 자신이 직접 처리



```
import java.io.IOException;

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:");
                try {
                        System.in.read(buf);
                } catch (IOException e) {
                        System.out.println(e.getMessage());
                        e.printStackTrace();
                }
                return new String(buf);
        }
}
```



예외를 상위 메소드로 전달

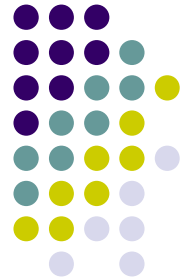
```
import java.io.IOException;

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);
        }
}
```

예외를 상위 메소드로 전달

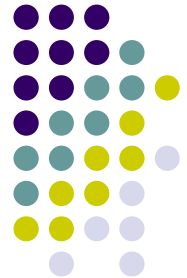
여러 개의 예외를 주고 받는 방법



```
class NetworkFile {
        public void saveFile( ) throws NetworkException, DiskException
        {
                // 너트워크에서 데이터를 받아서 파일에 저장한다.
        }
}
```

```
public class Test {
        public void process() {
                NetworkFile f = new NetworkFile();
                try {
                        f.saveFile();
                }
                catch(NetworkException e){
                        // 네트워크 오류 처리 코드
                }
                catch(DiskException e){
                        // 디스크 오류 처리 코드
                }
        }
}
```

5. 사용자 정의 예외



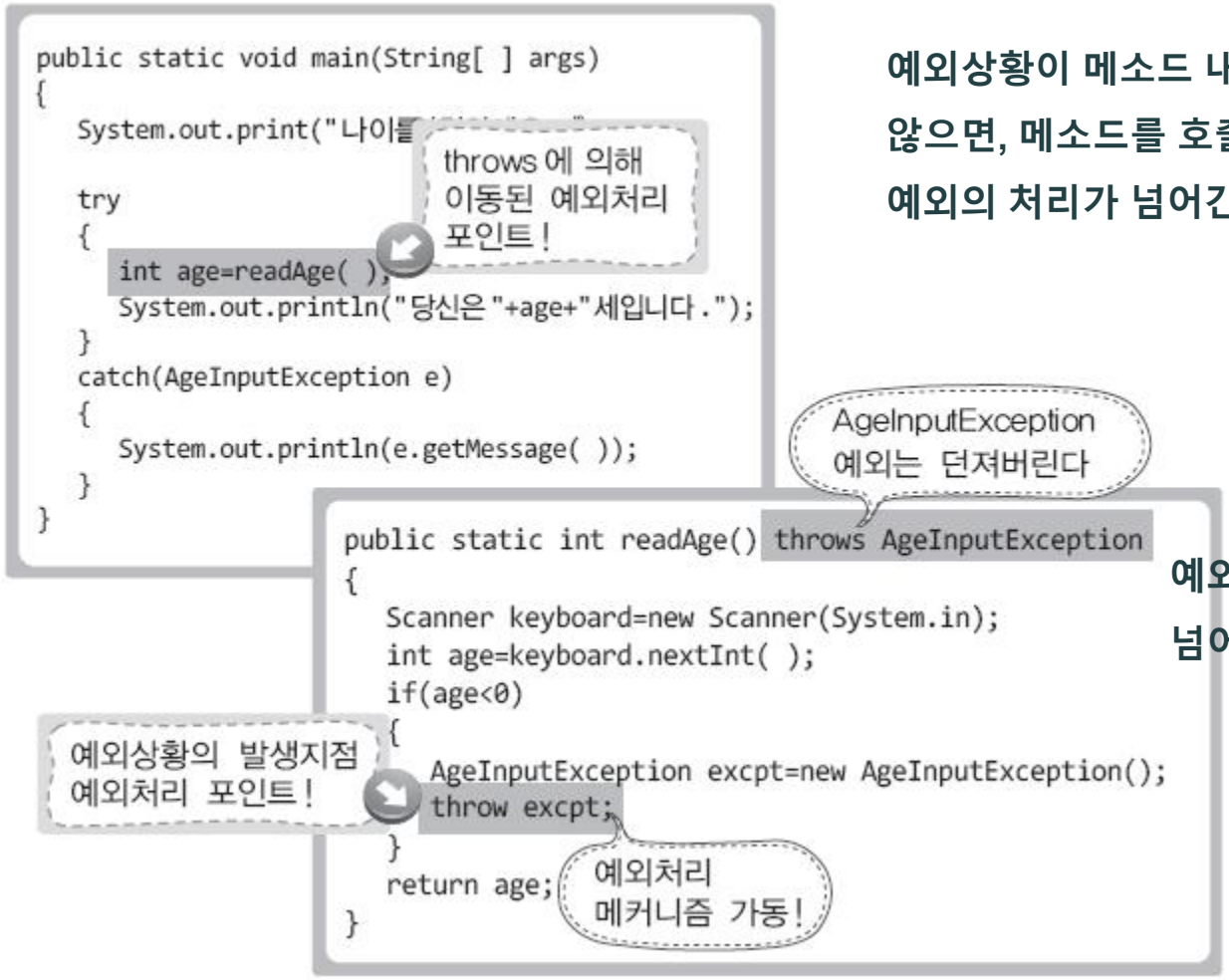
- 프로그램의 성격에 따라 프로그래머가 직접 상황에 맞는 예외 클래스를 정의해야 할 때가 있다.

  - 나이를 입력하라고 했더니, -20살을 입력했다.
  - 이름 정보를 입력하라고 했더니, 나이 정보를 입력했다.

- Exception의 서브클래스로 새로운 예외 클래스 정의

```
class AgeInputException extends Exception
{
    public AgeInputException()
    {
        super("유효하지 않은 나이가 입력되었습니다.");
    }
}
```

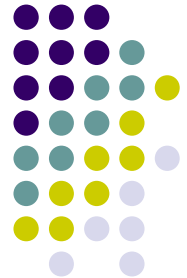
사용자 정의 예외 처리 예



예외상황이 메소드 내에서 처리되지 않으면, 메소드를 호출한 영역으로 예외의 처리가 넘어간다!

예외가 처리되지 않고 넘어감을 명시해야 한다.

사용자 정의 예외 처리 예(2)



```
class DivideByZeroException     extends ArithmeticException {
    public DivideByZeroException()
    {
        super( "0으로 나눌수는 없음." );
    }
}
 public class ExceptionTest {
    public static void main(String[] args)
    {
        double result;
        try {
            result = quotient(1,0);
        }
        catch ( DivideByZeroException e ) {
            System.out.println(e.toString());
        }
    }

    public static double quotient( int n, int d )
        throws DivideByZeroException
    {
        if ( d == 0 )
             throw new DivideByZeroException();
        return ( double ) n/ d;
     }
}
```

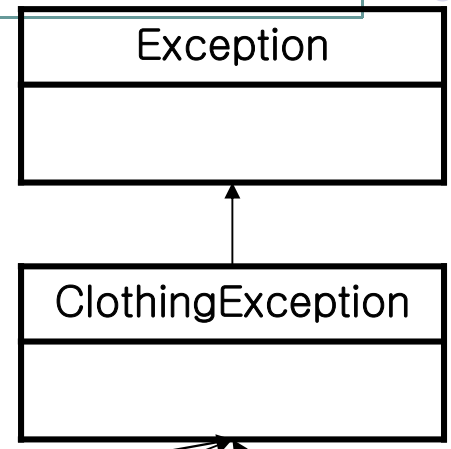
실행결과

```
DivideByZeroException: 0으로 나눌수는 없음.
```

6. 다형성과 예외

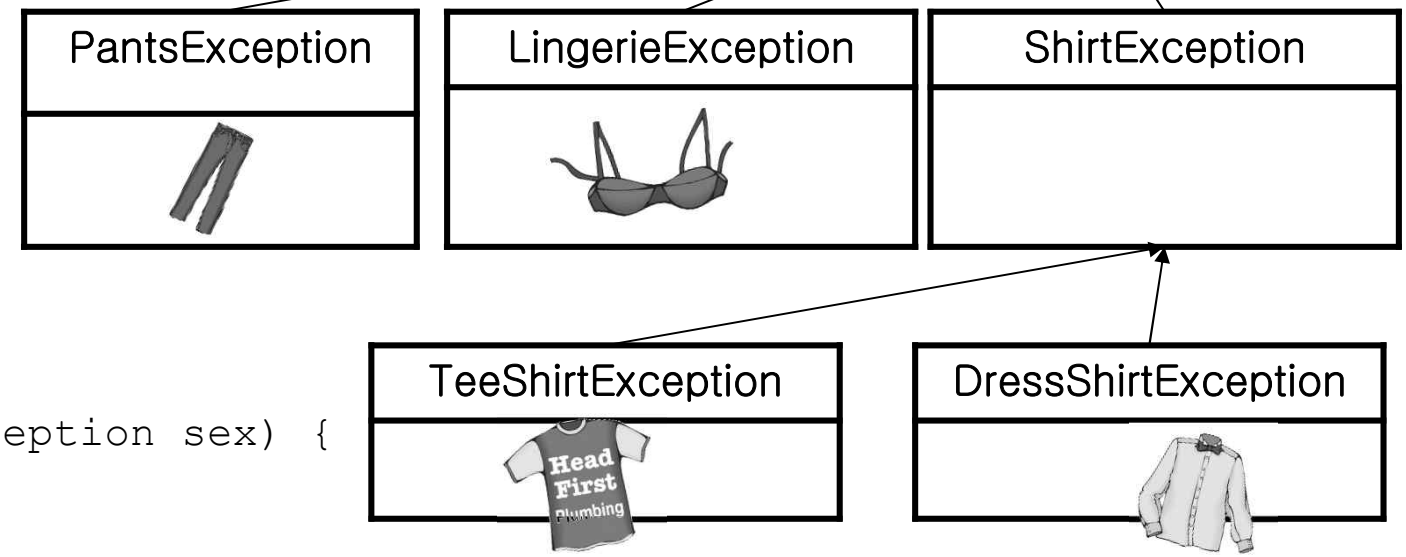
```
public void doLaundry throws ClothingException {
```

메소드에서 여러가지 예외를 던질 때, 던지고자 하는 예외의 수퍼클래스 유형을 이용하여 예외를 선언할 수 있다.



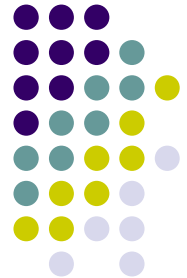

```
} catch (ClothingException cex) {
```

던져지는 예외를 수퍼클래스 유형을 써서 예외를 잡을 수 있다.




```
} catch (ShirtException sex) {
```

예외를 잡을 때는 작은것→큰것



```
try {
    laundry.doLaundry();
} catch(TeeShirtException tex) {

} catch(LingerieException lex) {

} catch(ClothingException cex) {

}
```

여러 개의 catch 블록을 쓸 때는 상/하위클래스 관계를 잘 따져봐야 합니다.

다형성과 예외 예

- 다음과 같은 예외 클래스 상속 계층 가정

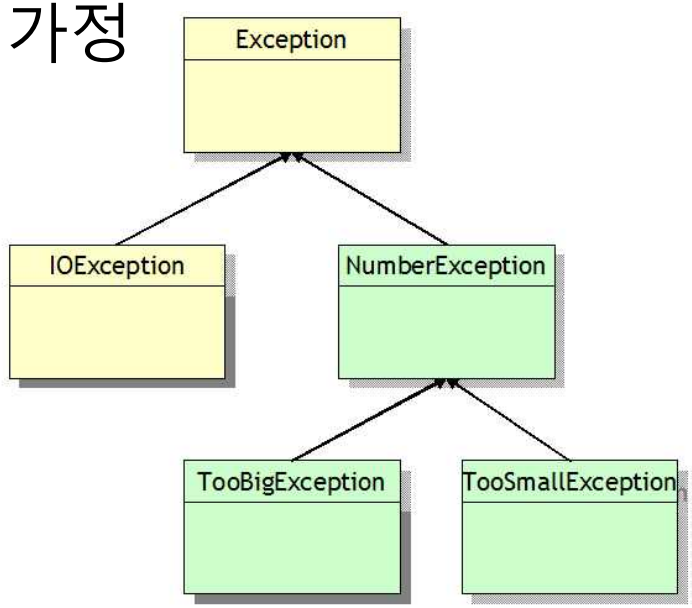


그림 15.5 오류 클래스 상속 계층도

```
try {
    getInput();
}
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) {
    //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!
}
```

Catch 블록 작성 시에는 범위가 작은 것부터 큰 것 순서로 작성해야 함