

Java의 정석

제 8 장

예외처리 (Exception handling)

2009. 10. 28

남궁성 강의

castello@naver.com

1. 예외처리(Exception handling)

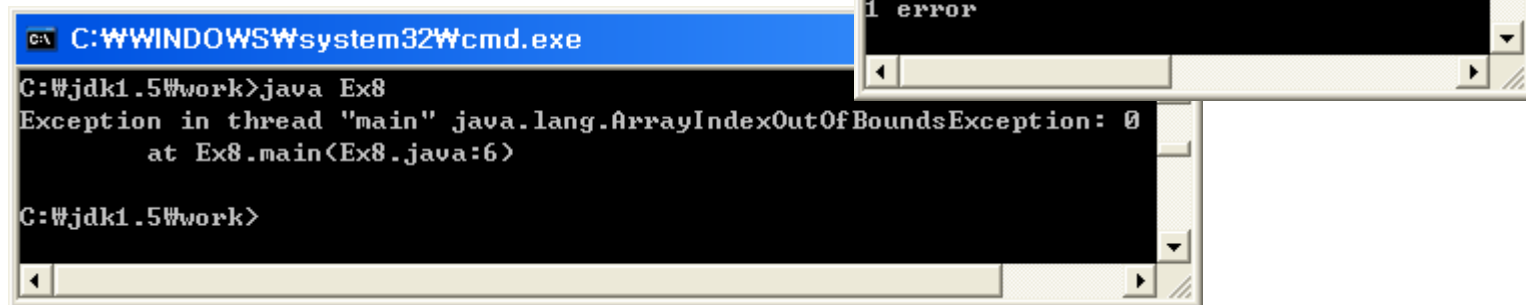
- 1.1 프로그램 오류
- 1.2 예외처리의 정의와 목적
- 1.3 예외처리구문 – try-catch
- 1.4 try-catch문에서의 흐름
- 1.5 예외 발생시키기
- 1.6 예외클래스의 계층구조
- 1.7 예외의 발생과 catch블럭
- 1.8 finally블럭
- 1.9 메서드에 예외 선언하기
- 1.10 예외 되던지기(re-throwing)
- 1.11 사용자정의 예외 만들기

1. 예외처리(Exception handling)

1.1 프로그램 오류

▶ 컴파일 에러(compile-time error)와 런타임 에러(runtime error)

- . 컴파일 에러 – 컴파일할 때 발생하는 에러
- . 런타임 에러 – 실행할 때 발생하는 에러



```
C:\WINDOWS\system32\cmd.exe
C:\Wjdk1.5\work>javac Ex8.java
Ex8.java:6: cannot find symbol
symbol  : method println(int)
location: class java.io.PrintStream
    System.out.println(i);
                  ^
1 error

C:\WINDOWS\system32\cmd.exe
C:\Wjdk1.5\work>java Ex8
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
    at Ex8.main(Ex8.java:6)

C:\Wjdk1.5\work>
```

▶ Java의 런타임 에러 – 에러(error)와 예외(exception)

에러(error) – 프로그램 코드에 의해서 수습될 수 없는 심각한 오류

예외(exception) – 프로그램 코드에 의해서 수습될 수 있는 다소 미약한 오류

1.2 예외처리의 정의와 목적

- 에러(error)는 어쩔 수 없지만, 예외(exception)는 처리해야 한다.

에러(error) - 프로그램 코드에 의해서 수습될 수 없는 심각한 오류

예외(exception) - 프로그램 코드에 의해서 수습될 수 있는 다소 미약한 오류

- 예외처리의 정의와 목적

예외처리(exception handling)의

정의 - 프로그램 실행 시 발생할 수 있는 예외의 발생에 대비한 코드를 작성하는 것

목적 - 프로그램의 비정상 종료를 막고, 정상적인 실행상태를 유지하는 것

[참고] 에러와 예외는 모두 실행 시(runtime) 발생하는 오류이다.

1.3 예외처리구문 – try-catch

- 예외를 처리하려면 try-catch문을 사용해야 한다.

```
try {  
    // 예외가 발생할 가능성이 있는 문장들을 넣는다.  
} catch (Exception1 e1) {  
    // Exception1이 발생했을 경우, 이를 처리하기 위한 문장을 적는다.  
} catch (Exception2 e2) {  
    // Exception2가 발생했을 경우, 이를 처리하기 위한 문장을 적는다.  
...  
} catch (ExceptionN eN) {  
    // ExceptionN이 발생했을 경우, 이를 처리하기 위한 문장을 적는다.  
}
```

[참고] if문과 달리 try블럭이나 catch블럭 내에 포함된 문장이 하나라고 해서 괄호{}를 생략할 수는 없다.

```
public static void main(String[] args)  
{  
    try {  
        try {    } catch (Exception e) {  
            //...  
        }  
        } catch (Exception e) {  
            try {    } catch (Exception e) { // 컴파일 에러 발생 !!!  
                //...  
            }  
        } // try-catch의 끝  
    } // main메서드의 끝
```

1.4 try-catch문에서의 흐름

- ▶ try블럭 내에서 예외가 발생한 경우,
 1. 발생한 예외와 일치하는 catch블럭이 있는지 확인한다.
 2. 일치하는 catch블럭을 찾게 되면, 그 catch블럭 내의 문장들을 수행하고 전체 try-catch문을 빠져나가서 그 다음 문장을 계속해서 수행한다. 만일 일치하는 catch블럭을 찾지 못하면, 예외는 처리되지 못한다.
- ▶ try블럭 내에서 예외가 발생하지 않은 경우,
 1. catch블럭을 거치지 않고 전체 try-catch문을 빠져나가서 수행을 계속한다.

```
class ExceptionEx4 {  
    public static void main(String args[]) {  
        System.out.println(1);  
        System.out.println(2);  
  
        try {  
            System.out.println(3);  
            System.out.println(4);  
        } catch (Exception e) {  
            System.out.println(5);  
        } // try-catch의 끝  
        System.out.println(6);  
    } // main메서드의 끝  
}
```

[실행결과]

```
1  
2  
3  
4  
6
```

```
class ExceptionEx5 {  
    public static void main(String args[]) {  
        System.out.println(1);  
        System.out.println(2);  
        try {  
            System.out.println(3);  
            System.out.println(0/0);  
            System.out.println(4);  
        } catch (ArithmeticException ae) {  
            System.out.println(5);  
        } // try-catch의 끝  
        System.out.println(6);  
    } // main메서드의 끝  
}
```

[실행결과]

```
1  
2  
3  
5  
6
```

1.5 예외 발생시키기

1. 먼저, 연산자 `new`를 이용해서 발생시키려는 예외 클래스의 객체를 만든 다음

```
Exception e = new Exception("고의로 발생시켰음");
```

2. 키워드 `throw`를 이용해서 예외를 발생시킨다.

```
throw e;
```

[예제8-6]/ch8/ExceptionEx6.java

```
class ExceptionEx6
{
    public static void main(String args[])
    {
        try {
            Exception e = new Exception("고의로 발생시켰음.");
            throw e; // 예외를 발생시킴
            // throw new Exception("고의로 발생시켰음.");

        } catch (Exception e) {
            System.out.println("에러 메시지 : " + e.getMessage());
            e.printStackTrace();
        }
        System.out.println("프로그램이 정상 종료되었습니다.");
    }
}
```

위의 두 줄을 한 줄로
줄여 쓸 수 있다.

[실행결과]

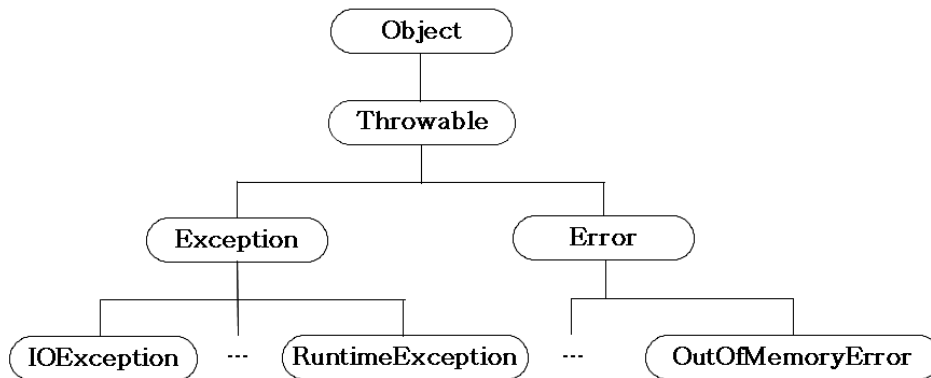
```
에러 메시지 : 고의로 발생시켰음.
java.lang.Exception: 고의로 발생시켰음.
    at ExceptionEx6.main(ExceptionEx6.java:6)
프로그램이 정상 종료되었음.
```


1.6 예외 클래스의 계층구조(1/2)

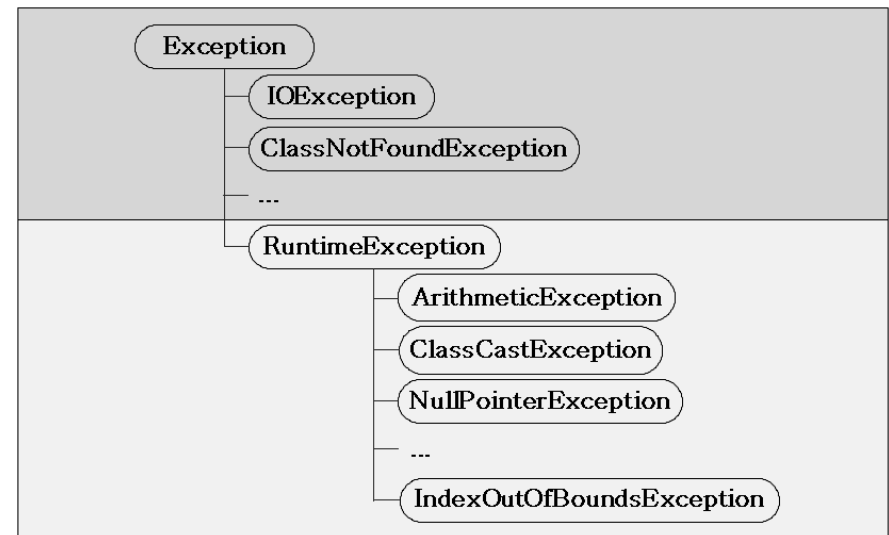
- 예외 클래스는 크게 두 그룹으로 나뉜다.

RuntimeException 클래스들 - 프로그래머의 실수로 발생하는 예외 ← 예외처리 필수

Exception 클래스들 - 사용자의 실수와 같은 외적인 요인에 의해 발생하는 예외 ← 예외처리 선택



[그림8-1] 예외클래스 계층도



[그림8-2] Exception클래스와 RuntimeException클래스 중심의 상속계층도

1.6 예외 클래스의 계층구조(2/2)

RuntimeException 클래스들 - 프로그래머의 실수로 발생하는 예외 ← 예외처리 필수

Exception 클래스들 - 사용자의 실수와 같은 외적인 요인에 의해 발생하는 예외 ← 예외처리 선택

[예제8-7]/ch8/ExceptionEx7.java

```
class ExceptionEx7
{
    public static void main(String[] args)
    {
        throw new Exception(); // Exception을 강제로 발생시킨다.
    }
}
```

[예제8-8]/ch8/ExceptionEx8.java

```
class ExceptionEx8 {
    public static void main(String[] args)
    {
        try {
            throw new Exception();
        } catch (Exception e) {
            System.out.println("Exception이 발생했습니다.");
        }
    } // main메서드의 끝
}
```

[예제8-9]/ch8/ExceptionEx9.java

```
class ExceptionEx9 {
    public static void main(String[] args)
    {
        throw new RuntimeException(); // RuntimeException을 강제로 발생시킨다.
    }
}
```

1.7 예외의 발생과 catch블럭(1/2)

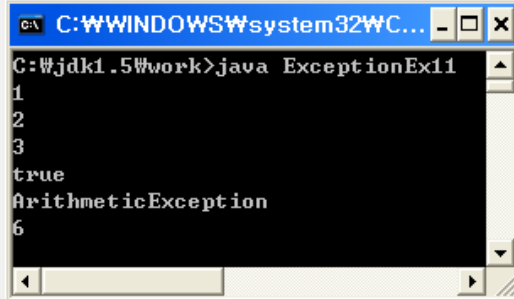
- try블럭에서 예외가 발생하면, 발생한 예외를 처리할 catch블럭을 찾는다.
- 첫번째 catch블럭부터 순서대로 찾아 내려가며, 일치하는 catch블럭이 없으면 예외는 처리되지 않는다.
- 예외의 최고 조상인 Exception을 처리하는 catch블럭은 모든 종류의 예외를 처리할 수 있다.(반드시 마지막 catch블럭이어야 한다.)

[예제8-11]/ch8/ExceptionEx11.java

```
class ExceptionEx11 {  
    public static void main(String args[]) {  
        System.out.println(1);  
        System.out.println(2);  
        try {  
            System.out.println(3);  
            System.out.println(0/0);  
            System.out.println(4);  
        } catch (ArithmeticException ae) {  
            if (ae instanceof ArithmeticException)  
                System.out.println("true");  
            System.out.println("ArithmeticException");  
        } catch (Exception e) {  
            System.out.println("Exception");  
        }  
        // try-catch의 끝  
        System.out.println(6);  
    }  
    // main메서드의 끝  
}
```

0으로 나눠서
ArithmeticException을
발생시킨다.

ArithmeticException을
제외한 모든 예외가 처리된
다.



```
C:\WINDOWS\system32\cmd.exe  
C:\Wjdk1.5\work>java ExceptionEx11  
1  
2  
3  
true  
ArithmeticException  
6
```

1.7 예외의 발생과 catch블럭(2/2)

- 발생한 예외 객체를 catch블럭의 참조변수로 접근할 수 있다.

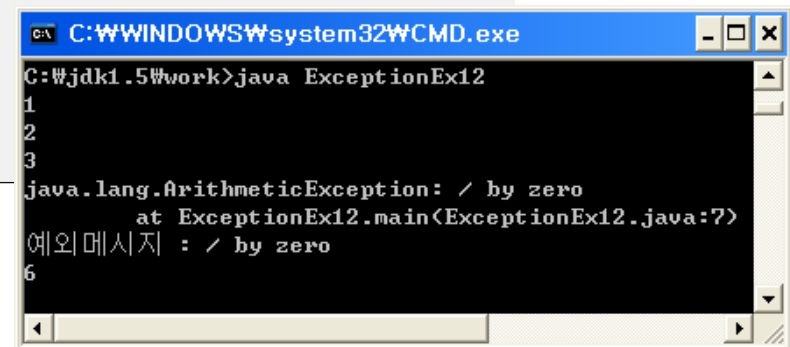
printStackTrace() - 예외발생 당시의 호출스택(Call Stack)에 있었던 메서드의 정보와 예외 메시지를 화면에 출력한다.

getMessage() - 발생한 예외클래스의 인스턴스에 저장된 메시지를 얻을 수 있다.

[예제8-12]/ch8/ExceptionEx12.java

```
class ExceptionEx12 {  
    public static void main(String args[]) {  
        System.out.println(1);  
        System.out.println(2);  
        try {  
            System.out.println(3);  
            System.out.println(0/0); // 예외발생!!  
            System.out.println(4);    // 실행되지 않는다.  
        } catch (ArithmeticException ae) {  
            ae.printStackTrace();  
            System.out.println("예외메시지 : " + ae.getMessage());  
        } // try-catch의 끝  
        System.out.println(6);  
    } // main메서드의 끝  
}
```

참조변수 ae를 통해, 생성된 ArithmeticException 인스턴스에 접근할 수 있다.



```
C:\WINDOWS\system32\cmd.exe  
C:\jdk1.5\work>java ExceptionEx12  
1  
2  
3  
java.lang.ArithmeticException: / by zero  
    at ExceptionEx12.main(ExceptionEx12.java:7)  
예외메시지 : / by zero  
6
```

1.8 finally블럭

- 예외의 발생여부와 관계없이 실행되어야 하는 코드를 넣는다.
- 선택적으로 사용할 수 있으며, try-catch-finally의 순서로 구성된다.
- 예외 발생시, try → catch → finally의 순서로 실행되고
예외 미발생시, try → finally의 순서로 실행된다.
- try 또는 catch블럭에서 return문을 만나도 finally블럭은 수행된다.

```
try {  
    // 예외가 발생할 가능성이 있는 문장들을 넣는다.  
} catch (Exception1 e1) {  
    // 예외처리를 위한 문장을 적는다.  
} finally {  
    // 예외의 발생여부에 관계없이 항상 수행되어야하는 문장들을 넣는다.  
    // finally블럭은 try-catch문의 맨 마지막에 위치해야한다.  
}
```

1.8 finally블럭 - 예제

[예제8-15]/ch8/FinallyTest.java

```
class FinallyTest {
    public static void main(String args[]) {
        try {
            startInstall();      // 프로그램 설치에 필요한 준비를 한다.
            copyFiles();         // 파일들을 복사한다.
            deleteTempFiles();   // 프로그램 설치에 사용된 임시파일들을 삭제한다.
        } catch (Exception e) {
            e.printStackTrace();
            deleteTempFiles();   // 프로그램 설치에 사용된 임시파일들을 삭제한다.
        } // try-catch의 끝
    } // main의 끝

    static void startInstall() {
        /* 프로그램 설치에 필요한 준비를 하는 코드를 적는다.*/
    }

    static void copyFiles() { /* 파일들을 복사하는 코드를 적는다. */ }
    static void deleteTempFiles() { /* 임시파일들을 삭제하는 코드를 적는다.*/ }
}

try {
    startInstall();
    copyFiles();
} catch (Exception e) {
    e.printStackTrace();
} finally {
    deleteTempFiles();
} // try-catch의 끝
```

1.9 메서드에 예외 선언하기

- 예외를 처리하는 또 다른 방법
- 사실은 예외를 처리하는 것이 아니라, 호출한 메서드로 전달해주는 것
- 호출한 메서드에서 예외처리를 해야만 할 때 사용

```
void method() throws Exception1, Exception2, ... ExceptionN {
    // 메서드의 내용
}
```

[참고] 예외를 발생시키는 키워드 throw와 예외를 메서드에 선언할 때 쓰이는 throws를 잘 구별하자.

```
public final void wait()
    throws InterruptedException
```

Causes current thread to wait until another thread invokes the `notify()` method for this object. In other words, this method implements the call `wait(0)`.

Throws:

[IllegalMonitorStateException](#) - if the current thread is not the owner of the object.
[InterruptedException](#) - if another thread interrupted the current thread while it was waiting for a notification. The `interrupted` status is cleared when this exception is thrown.

See Also:

[notify\(\)](#), [notifyAll\(\)](#)

java.lang

Class IllegalMonitorStateException

[java.lang.Object](#)

↳ [java.lang.Throwable](#)

↳ [java.lang.Exception](#)

↳ [java.lang.RuntimeException](#)

↳ [java.lang.IllegalMonitorStateException](#)

java.lang

Class InterruptedException

[java.lang.Object](#)

↳ [java.lang.Throwable](#)

↳ [java.lang.Exception](#)

↳ [java.lang.InterruptedException](#)

1.9 메서드에 예외 선언하기 – 예제1

[예제8-18]/ch8/ExceptionEx18.java

```

class ExceptionEx18 {
    public static void main(String[] args) throws Exception {
        method1();    // 같은 클래스내의 static멤버이므로 객체생성없이 직접 호출가능.
    }    // main메서드의 끝

    static void method1() throws Exception {
        method2();
    }    // method1의 끝

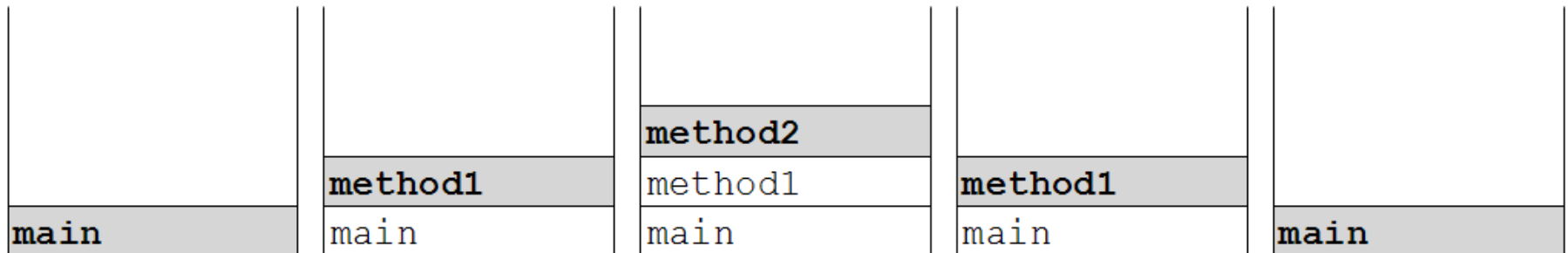
    static void method2() throws Exception {
        throw new Exception();
    }    // method2의 끝
}

```

```

C:\WINDOWS\system32\cmd.exe
G:\jdk1.5\work>java ExceptionEx18
Exception in thread "main" java.lang.Exception
    at ExceptionEx18.method2(ExceptionEx18.java:11)
    at ExceptionEx18.method1(ExceptionEx18.java:7)
    at ExceptionEx18.main(ExceptionEx18.java:3)

```



- 예외가 발생했을 때, 모두 3개의 메서드(main, method1, method2)가 호출스택에 있었으며,
- 예외가 발생한 곳은 제일 윗줄에 있는 method2()라는 것과
- main메서드가 method1()를, 그리고 method1()은 method2()를 호출했다는 것을 알 수 있다.

1.9 메서드에 예외 선언하기 – 예제2

[예제8-19]/ch8/ExceptionEx19.java

```

class ExceptionEx19 {
    public static void main(String[] args) {
        method1();    // 같은 클래스내의 static멤버이므로 객체생성없이 직접 호출가능.
    }    // main메서드의 끝

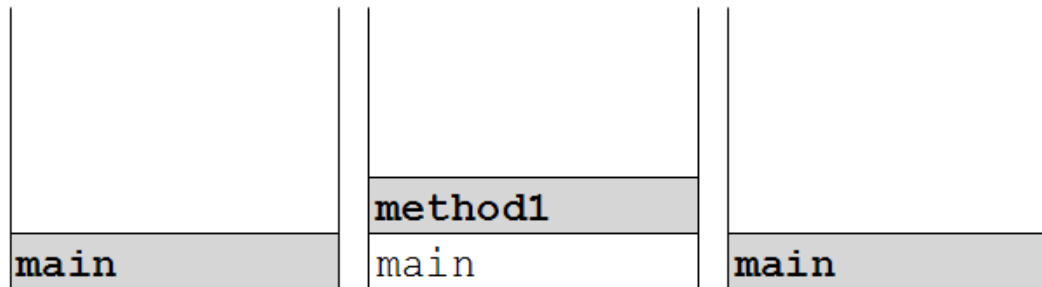
    static void method1() {
        try {
            throw new Exception();
        } catch (Exception e) {
            System.out.println("method1메서드에서 예외가 처리되었습니다.");
            e.printStackTrace();
        }
    }    // method1의 끝
}

```

```

C:\WINDOWS\system32\cmd.exe
C:\Wjdk1.5\work>java ExceptionEx19
method1메서드에서 예외가 처리되었습니다.
java.lang.Exception
    at ExceptionEx19.method1(ExceptionEx19.java:8)
    at ExceptionEx19.main(ExceptionEx19.java:3)

```



1.9 메서드에 예외 선언하기 – 예제3

[예제8-21]/ch8/ExceptionEx21.java

```
import java.io.*;

class ExceptionEx21 {
    public static void main(String[] args) {
        // command line에서 입력받은 값을 이름으로 갖는 파일을 생성한다.
        File f = createFile(args[0]);
        System.out.println( f.getName() + " 파일이 성공적으로 생성되었습니다. ");
    } // main메서드의 끝

    static File createFile(String fileName) {
        try {
            if (fileName==null || fileName.equals(""))
                throw new Exception("파일이름이 유효하지 않습니다.");
        } catch (Exception e) {
            // fileName이 부적절한 경우, 파일 이름을 '제목없음.txt'로 한다.
            fileName = "제목없음.txt";
        } finally {
            File f = new File(fileName); // File클래스의 객체를 만든다.
            createNewFile(f);           // 생성된 객체를 이용해서 파일을 생성한다.
            return f;
        }
    } // createFile메서드의 끝

    static void createNewFile(File f) {
        try {
            f.createNewFile(); // 파일을 생성한다.
        } catch (Exception e) { }
    } // createNewFile메서드의 끝
} // 클래스의 끝
```

[실행결과]

C:\jdk1.5\work>java ExceptionEx21 "test.txt"
test.txt 파일이 성공적으로 생성되었습니다.

C:\jdk1.5\work>java ExceptionEx21 ""
제목없음.txt 파일이 성공적으로 생성되었습니다.

C:\jdk1.5\work>dir *.txt

드라이브 c에 레이블이 없습니다
볼륨 일련 번호 251C-08DD
디렉터리 C:\jdk1.5\work

제목없음 TXT 0 03-01-24 0:47 제목없음.txt
TEST TXT 0 03-01-24 0:47 test.txt

1.9 메서드에 예외 선언하기 – 예제4

[예제8-22]/ch8/ExceptionEx22.java

```
import java.io.*;

class ExceptionEx22 {
    public static void main(String[] args)
    {
        try {
            File f = createFile(args[0]);
            System.out.println( f.getName()+"파일이 성공적으로 생성되었습니다.");
        } catch (Exception e) {
            System.out.println(e.getMessage()+" 다시 입력하십시오.");
        }
    } // main메서드의 끝

    static File createFile(String fileName) throws Exception {
        if (fileName==null || fileName.equals(""))
            throw new Exception("파일 이름이 유효하지 않습니다.");
        File f = new File(fileName); // File클래스의 객체를 만든다.
        // File객체의 createNewFile메서드를 이용해서 실제 파일을 생성한다.
        f.createNewFile();
        return f; // 생성된 객체의 참조를 반환한다.
    } // createFile메서드의 끝
} // 클래스의 끝
```

[실행결과]

```
C:\jdk1.5\work>java ExceptionEx22 test2.txt
test2.txt파일이 성공적으로 생성되었습니다.
```

```
C:\jdk1.5\work>java ExceptionEx22 ""
파일 이름이 유효하지 않습니다. 다시 입력해 주시기 바랍니다.
```

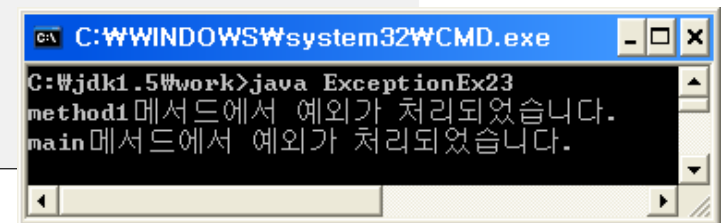
1.10 예외 되던지기(re-throwing)

- 예외를 처리한 후에 다시 예외를 생성해서 호출한 메서드로 전달하는 것
- 예외가 발생한 메서드와 호출한 메서드, 양쪽에서 예외를 처리해야 하는 경우에 사용.

[예제8-23]/ch8/ExceptionEx23.java

```
class ExceptionEx23 {
    public static void main(String[] args)
    {
        try {
            method1();
        } catch (Exception e) {
            System.out.println("main메서드에서 예외가 처리되었습니다.");
        }
    } // main메서드의 끝

    static void method1() throws Exception {
        try {
            throw new Exception();
        } catch (Exception e) {
            System.out.println("method1메서드에서 예외가 처리되었습니다.");
            throw e; // 다시 예외를 발생시킨다.
        }
    } // method1메서드의 끝
}
```



```
C:\W\WINDOWS\system32\CMD.exe
C:\W\jdk1.5\work>java ExceptionEx23
method1메서드에서 예외가 처리되었습니다.
main메서드에서 예외가 처리되었습니다.
```

1.11 사용자정의 예외 만들기

- 기존의 예외 클래스를 상속받아서 새로운 예외 클래스를 정의할 수 있다.

```
class MyException extends Exception {  
    MyException(String msg) { // 문자열을 매개변수로 받는 생성자  
        super(msg); // 조상인 Exception 클래스의 생성자를 호출한다.  
    }  
}
```

- 에러코드를 저장할 수 있게 ERR_CODE와 getErrCode()를 멤버로 추가

```
class MyException extends Exception {  
    // 에러 코드 값을 저장하기 위한 필드를 추가 했다.  
    private final int ERR_CODE;  
  
    MyException(String msg, int errCode) { // 생성자.  
        super(msg);  
        ERR_CODE = errCode;  
    }  
  
    MyException(String msg) { // 생성자.  
        this(msg, 100); // ERR_CODE를 100 (기본값) 으로 초기화한다.  
    }  
  
    public int getErrCode() { // 에러 코드를 얻을 수 있는 메서드도 추가했다.  
        return ERR_CODE; // 이 메서드는 주로 getMessage()와 함께 사용될 것이다.  
    }  
}
```

감사합니다.

더 많은 동영상강좌를 아래의 사이트에서 구하실 수 있습니다.

<http://www.javachobo.com>

이 동영상강좌는 비상업적 용도일 경우에 한해서 저자의 허가없이 배포하실 수 있습니다.
그러나 일부 무단전제 및 변경은 금지합니다.

관련문의 : 남궁성 castello@naver.com