

11장. Exception

1. Exception 이란?

프로그램을 작성하다 보면 수많은 에러(error)에 부딪치게 되고 에러가 발생하면 프로그램은 에러가 발생한 곳에서 멈추게 된다. 에러는 그 종류와 프로그램에 미치는 심각성이 각기 다르므로 대처하는 방법이 다를 수밖에 없다.

예외라는 것은 에러 중에서 대처할 수 있는 에러라고 말할 수 있다. 예외처리는 예외를 방지하거나 에러로 인한 프로그램 수행결과가 잘못 동작을 하는 것이 아니라, 에러를 잡고 처리하는 방법을 제공한다. 따라서 예외의 긍정적인 측면은 특정에러에 대응방법을 제공한다는 것이다.

Java 의 예외 객체는 모두 java.lang.Throwable 에서 파생된다. Error, Exception 가 있다
프로그램이 중단하게 되면 JVM은 Throwable이 자동 생성되어 중단된 원인을 찾아 Error, Exception의 객체를 생성하여 기본 예외처리를 한다. . Error 클래스의 자손이 발생하면 치명적인 오류이므로 프로그램이 종료되는 되지만 Exception 클래스는 Runtime시에 발생하는 오류로 오류 메시지 등을 내보내고 오류 발생 가능성이 있는 부분에 대해서 미리 프로그램으로 처리를 하도록 한다.

Java에서는 예외가 발생했을 때, 미리 준비되어있는 오류 메시지 (Exception in thread "main" java.lang.ArithmeticException ... 등)를 표시 프로그램을 종료한다 이것이 Java의 기본 예외 처리이다 기본예외처리가 발생되어 JVM 으로부터 Exception 을 발생 결과를 받게 되면 프로그래머는 그 발생결과를 보고 Exception handling 을 하게 된다.

다음과 같은 프로그램을 연동해 보자.

```
public class TestException {  
    public static void main(String[] args) {  
        int x = 10/0;  
    }  
}
```

[실행결과]

Exception in thread "main" java.lang.ArithmeticException: / by zero
at com.sec11.TestException.main(TestException.java:5) -->기본예외처리

위의 프로그램을 Exception handling 을 한 결과이다.

```

public class TestException {
    public static void main(String[] args) {
        try {
            int x = 10 / 0; // (1) 0으로 나누기
        } catch (ArithmeticException e) { // (2)
            System.out.println("0으로 나눌 수 없습니다.");
        }
    }
}

```

[실행결과]

0으로 나눌 수 없습니다.

[설명]

- (1) 정수를 0으로 나눈 경우 예외가 발생하고 예외 객체가 생성된 Java 실행 시스템에 전달된다.
- (2) 예외가 발생하면 Java 실행 시스템은 해당 예외 처리기를 찾는다. 해당 예외 핸들러를 찾으면, 그 예외 핸들러를 실행한다. 예외 핸들러는 println 메소드에서 "0으로 나눌 수 없습니다."를 표시하는 작업이 포함되어 있다.

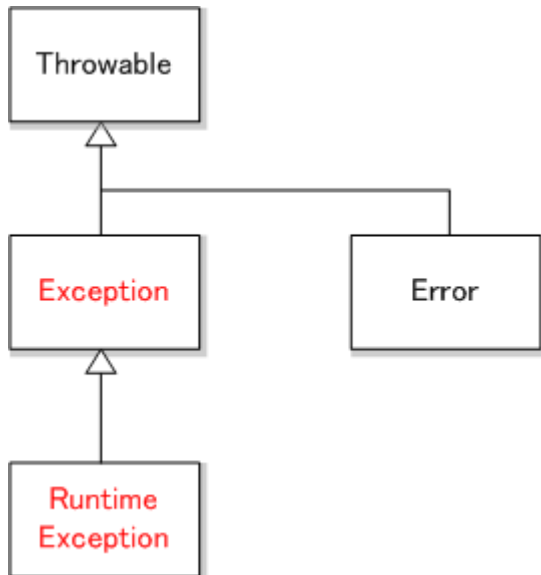
2. Exception 의 종류와 처리

>>예외클래스의 종류

Java 예외는 "Throwable"클래스를 상속한 객체를 throw 하는 것으로 발생합니다. 예외로 throw 되는 클래스는 다음의 3 종류로 분류되어 있다.

예외의 종류 목록		
종류	특징	상속하는 클래스
Exception	이 클래스가 throw 되는 소스는 컴파일시 예외 처리 구현이 적용된다. (검사 예외)	IOException, ClassNotFoundException 등
RuntimeException	이 클래스가 throw 되는 소스는 컴파일시 예외 처리 구현이 강제되지 않는다. (실행시 예외)	ArithmeticException, IndexOutOfBoundsException 등
Error	처리의 계속이 어려운 치명적인 경우이기 때문에 예외 처리 작성할 수 없다	VirtualMachineError 등

실제로 throw 되는 클래스는 "Throwable"클래스를 직접 상속하는 것이 아니라, 상기 3 종류 (Exception, RuntimeException, Error)의 클래스 중 하나에서 파생 있습니다. 각 클래스 계층 구조는 다음과 같습니다.



예외 처리의 대상이되는 것은 "Exception", "RuntimeException"에서 파생된 클래스 이다.

>>예외 처리 구문

예외 처리는 대략 다음과 같은 패턴으로 분류됩니다.

1. throw 된 예외를 catch 로 잡아서 처리한다.
2. throw 된 예외를 캐치하지 않고 다른곳 으로 넘긴다.
3. 명시 적으로 예외를 발생시킨다.

먼저 throw 된 예외를 catch 로 잡아서 처리하는 형식은 다음과 같습니다.

```
try {  
    예외가 발생 될 수있는 처리;  
} catch (예외 1 의 형태 변수) {  
    예외 1 을 잡을 때의 처리;  
    ※ catch 절은 예외의 수만큼 지정할 수 있습니다.  
} catch (예외 2 의 형태 변수) {  
    예외 2 를 잡을 때의 처리;  
    .  
    .  
} finally {  
    반드시 실행되는 처리;
```

```
}
```

※ try 절에서 처리 중에 예외가 발생하면 이후의 처리는 행해지 지 않고 즉시 catch 절에 처리가 전환됩니다.

예외가 발생하지 않을 경우는 try~ catch 의 catch 가 실행 되지 않는다.

```
package com.ch11;
```

```
public class TestException01 {  
    public static void main(String[] args) {  
        // "IndexOutOfBoundsException" 런타임 예외 때문에 발생 될 수 있지만  
        // 예외 처리는 적용되지 않습니다.  
        String className = "com.ch11.TestException01";  
        try {  
            // 검사 예외 "ClassNotFoundException"가 발생 될 수 있으므로 예외 처리가 강제됩니다.  
            System.out.println(Class.forName(className) + "는 존재하고  
있습니다.");  
  
            System.out.println("예외가 발생하지 않았습니다.");  
        } catch (ClassNotFoundException e) {  
            System.out.println("예외가 발생했습니다.");  
            e.printStackTrace();  
        }  
    }  
}
```

[실행결과]

class com.ch11.TestException01는 존재하고 있습니다.

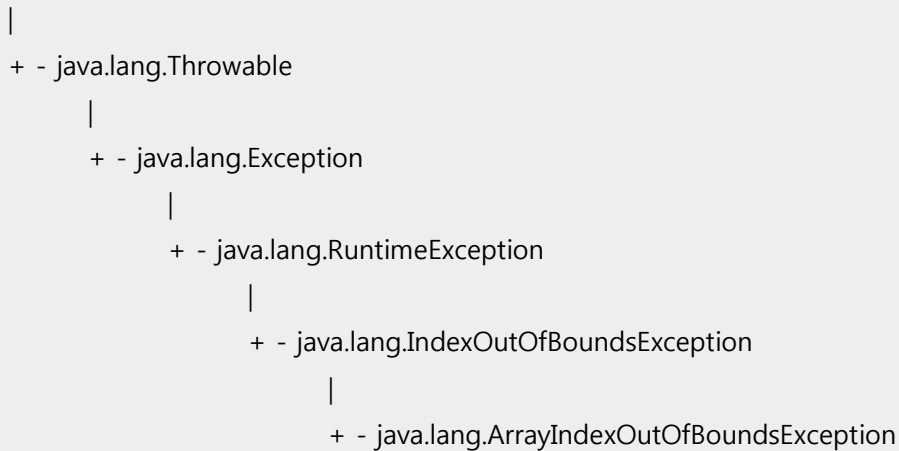
예외가 발생하지 않았습니다.

Java 의 예외는 개체로 발생한다. 임의의 입도 예외가 객체로 코어 패키지에 정의되어 있다. 처리 가능한 예외 가장 입도의 큰 것은 모든 처리 할 수 있는 예외 객체가 슈퍼 클래스에 있는 Exception 개체이다. 이하, Exception 클래스의 서브 클래스로서 개별적인 예외에 따라 예외 클래스가 정의되어 있다

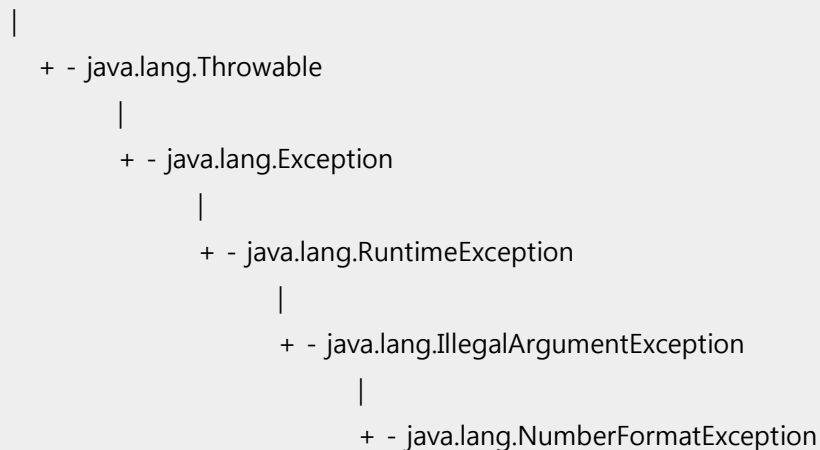
예를 들어, 배열에 잘못된 인덱스를 요청하면 Array Index Out Of Bouds Exception 클래스 형의 객체가 발생한다.

Array Index Out Of Bouds Exception

java.lang.Object



문자열을 숫자로 변환 할 때 숫자 수 없는 문자열이 주어지면
NumberFormatException 클래스 형의 객체가 발생한다.



try catch 블록에서 catch 블록 작성된 순서대로 예외 객체가 catch 된다. 따라서, 의도가 작은 것부터 순서대로 catch 블록을 작성한다. 예를 들어, 처음에 가장 의도가 큰 Exception 객체를 포착하면 나머지 catch 블록에 절대적으로 제어가 돌아 가지 않는다. 모든 예외 개체는 Exception 클래스 형에 할당 할 수 있도록 먼저 기술 한 Exception 클래스 형 catch 블록에서 처리하기 때문이다.

다음에는 NumberFormatException 은 첫 번째 catch 가 잡아서 처리한다. 다른 모든 예외는 두 번째 catch 에서 처리된다.

```
try { // 처리

} catch (java.lang.NumberFormatException e) {

// 처리
```

```

    } catch (Exception e) {

        // 처리

        // 모든 예외가 여기서 잡히는

    } finally {

        // 처리

    }

```

간단한 예외 처리 예제를 살펴보자.

ArrayIndexOutOfBoundsException

```

java.lang.Object
|
+ - java.lang.Throwable
    |
    + - java.lang.Exception
        |
        + - java.lang.RuntimeException
            |
            + - java.lang.IndexOutOfBoundsException
                |
                + - java.lang.ArrayIndexOutOfBoundsException

```

```

java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.lang.RuntimeException
        java.lang.IndexOutOfBoundsException
          java.lang.ArrayIndexOutOfBoundsException

```

이런 계층구조임

부정한 인덱스를 사용해 배열이 액세스 된 것을 나타낸다.

```

class TestExcep1 {
    public static void main (String [] args) {
        System.out.println ( "실행 시작");
        try {
            System.out.println (args [0]);
            System.out.println ( "try 블록 종료");

```

```

    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println ( "예외 :"+ e);
        System.out.println ( "인수를 하나 입력하십시오.");
    }
    System.out.println ( "실행 종료");
}
}

```

실행 시작.

예외 : java.lang.ArrayIndexOutOfBoundsException

인수를 하나 입력하십시오.

실행 종료.

인수의 배열의 개수가 올바르지 않기 때문에 일어난 예외입니다. 이 경우 논리에서 쉽게 해결할 수 있다.

```

if (args.length != 1) {
    System.out.println ( "인수를 하나 입력하십시오.");
    return;
}

```

배열 args[]의 요소 수가 1 이 아닌 경우에 true 가 되어, 블록이 실행된다.

NumberFormatException 을 처리한 경우

```

java.lang.Object
|
+ - java.lang.Throwable
    |
    + - java.lang.Exception
        |
        + - java.lang.RuntimeException
            |
            + - java.lang.IllegalArgumentException
                |
                + - java.lang.NumberFormatException

```

문자열을 수치 형으로 변환 하려고 했을 때, 캐릭터 라인의 형식이 올바르지 않은 경우에 발생된다.

package com.ch11;

public class TestException02 {

```

public static void main(String[] args) {
    System.out.println ( "실행 시작");
    try {
        int i;
        i = Integer.parseInt(args [0]);
        // 인수가 없거나 정수로 고치지 않으면 예외가 throw 됨
        System.out.println (i * 100);
        System.out.println ( "try 블록 종료");
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println ( "예외 :"+ e);
        System.out.println ( "인수를 하나 입력하십시오.");
    } catch (NumberFormatException e) {
        System.out.println ( "예외 :"+ e);
        System.out.println ( "인수를 정수로 입력하십시오.");
    }
    System.out.println ( "실행 종료");
}
}

```

[실행결과]

실행 시작

예외 :[java.lang.ArrayIndexOutOfBoundsException](#): 0

인수를 하나 입력하십시오.

실행 종료

ArithmeticException 을 처리한 경우

```

java.lang.Object
|
+ - java.lang.Throwable
    |
    + - java.lang.Exception
        |
        + - java.lang.RuntimeException
            |
            + - java.lang.ArithmeticException

```


산술 계산으로 예외적 조건이 발생했을 경우에 캐치된다. 예를 들어, 정수를 「제로로 나누려고 하면」 클래스의 인스턴스가 throw 된다..

```
public class TestException03 {
    public static void main(String[] args) {
        System.out.println("실행 시작");
        try {
            int i, j = 100;
            System.out.println("j : " + j);
            i = Integer.parseInt(args[0]);
            System.out.println("j / i : " + j / i);
            // 0으로 나누면 예외가 발생
            System.out.println("try 블록 종료");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("예외 : " + e);
            System.out.println("인수를 하나 입력하십시오.");
        } catch (NumberFormatException e) {
            System.out.println("예외 : " + e);
            System.out.println("인수를 정수로 입력하십시오.");
        } catch (ArithmeticException e) {
            System.out.println("예외 : " + e);
            System.out.println("인수를 0이 아닌 정수로 입력하십시오.");
        }
        System.out.println("실행 종료");
    }
}
```

인수가 주어지지 않는 경우 첫 번째 catch 의 블록으로 이동하고 . 인수는 주어진 하지만, 정수로 변환 할 수 없는 경우, 두 번째 catch 블록으로 이동해서 처리된다. . 인수가 주어 정수로 변환 할 수 있지만, 그것이 0 인 경우는 세 번째 catch 블록으로 이동되어 처리된다.

```
실행 시작.
j : 100
예외 : java.lang.ArithmeticException : / by zero
인수를 0 이 아닌 정수로 입력하십시오.
실행 종료.
실행 시작. ----->-5 을 입력한 경우
j : 100
```

```
j / i : -20  
try 블록 종료  
실행 종료.
```

Exception

발생하는 예외를 모두 캐치하려면 Exception 형으로 잡을 수 있습니다.

모든 예외 클래스는 Exception 클래스를 슈퍼 클래스로 가진다. 따라서 모든 예외 클래스를 Exception 형식으로 catch 할 수 있다.

```
public class TestException04 {  
    public static void main(String[] args) {  
        String str = null;  
        try {  
            System.out.println(str.length());  
        } catch (Exception e) {  
            System.out.println("예외 : " + e);  
        }  
    }  
}
```

이 예제에서는 null 값이 할당된 문자열 객체에 length() 메소드를 요구하고 있다. 문자열이 할당되어 있으면 문자의 개수를 돌려줍니다만, null 값의 경우는 예외 NullPointerException 이 발생한다.

[실행결과]

예외 : [java.lang.NullPointerException](#)

이 코드에서는 NullPointerException 예외가 발생하고 있지만 모든 예외는 Exception 에서 파생 있기 때문에 자동 형 변환되어 처리된다.

Exception 형은 모든 예외 오브젝트와 대입 호환이 가능하며, 마지막으로 기술 할 필요가 있다. Exception 형을 처음으로 기술 해 버리면 모든 예외 객체가 캐치되어 버리므로, 후속 catch 블록에 절대적으로 제어가 넘어 가지 않는다. 방금 전의 ArithmeticException 의 캐치의 예를 들면, Exception 은 마지막으로 설정되어야 한다.

```

public class TestException05 {

    public static void main(String[] args) {

        System.out.println ( "실행 시작");
        try {
            int i, j = 100;
            System.out.println ( "j :"+ j);
            i = Integer.parseInt(args [0]);
            System.out.println ( "j / i :"+ j / i);
            // 0으로 나누면 예외가 throw가 된다.
            System.out.println ( "try 블록 종료");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println ( "예외 :"+ e);
            System.out.println ( "인수를 하나 입력하십시오.");
        } catch (NumberFormatException e) {
            System.out.println ( "예외 :"+ e);
            System.out.println ( "인수를 정수로 입력하십시오.");
        } catch (ArithmeticException e) {
            System.out.println ( "예외 :"+ e);
            System.out.println ( "인수를 0이 아닌 정수로 입력하십시오.");
        } catch (Exception e) {
            System.out.println ( "어떤 예외가 발생했습니다.");
            e.printStackTrace ();
        }
        System.out.println ( "실행 종료");
    }
}

```

두번째 throw 된 예외를 캐치하지 않고 그대로 throw 하는 경우는 "throws"를 사용하면 예외 처리를하지 않고 발생한 예외를 그대로 던질 수 있는 경우이다.

즉 예외처리가 발생하게 되는 메소드를 호출하는 쪽으로 예외처리를 위임하는 것이다.

```

수식 반환 값 메서드 (서명) throws 예외 클래스 목록 {
    처리
}
생성자 (서명) throws 예외 클래스 목록 {
    초기화 처리
}

```

throws 뒤에 설명 throws 목록은 쉼표로 구분 여러 예외 클래스를 작성할 수 있다
 이렇게 명시 해두면 해당 메소드 이용자에게 알기 쉬울 뿐만 아니라 컴파일시 발생 된 예외가
 호출자 캐치되어 있는지 확인합니다. 캐치되어 있지 않으면 컴파일 오류가 발생한다. 구성 요소의
 개발자가 이용자에 대해 예외 처리를 강제 할 수 있기 때문에, 예상외의 예외가 발생하는 것을
 미연에 방지 할 수 있다.

예외 처리에는 두 가지이고 , try {} catch () {} 에서 자체적으로 처리할지 호출자에게 처리를
 위임하거나의 차이라고 말할 수 있다.

다음 코드를 보면 static int cal (int i, int j) throws ArithmeticException {} 메소드가 throws라는
 키워드를 이용해서 현재 메소드에서 발생하는 예외를 try~ catch로 처리하지 않고 호출하는
 쪽으로 위임해서 메인 메소드에서 처리하는 것을 볼 수 있다.

```
package com.ch11;
```

```
public class TestThrows {
```

```
    public static void main(String[] args) {
```

```
        System.out.println("실행 시작");
```

```
        int i = 100, j = 0, k = 0;
```

```
        try {
```

```
            k = cal(i, j);
```

```
            System.out.println(k);
```

```
        } catch (ArithmeticException e) {
```

```
            System.out.println("예외 :" + e);
```

```
        }
```

```
        System.out.println("실행 종료");
```

```
    }
```

```
    static int cal(int i, int j) throws ArithmeticException {
```

```
        return i / j;
    }
}
```

[실행결과]

실행 시작

예외 : [java.lang.ArithmeticException](#): / by zero

실행 종료

세번째 명시 적으로 예외를 발생시키는 경우로 "throw 문"을 사용하여 예외를 명시 적으로 발생시킬 수 있는 구문이다. 형식은 다음과 같다.

throw 예외 객체;

예외도 클래스이므로 인스턴스화하고 개체를 만들 수 있다. 이렇게 생성 된 예외 객체는 던져 온다.

```
public class TestThrows01 {

    public static void main(String[] args) {
        try {
            System.out.println("실행 시작");
            throw new IllegalAccessException();
        } catch (IllegalAccessException e) {
            System.out.println("예외 : " + e);
            System.out.println("실행 종료");
        }
    }
}
```

[실행결과]

실행 시작

예외 : [java.lang.IllegalAccessException](#)

실행 종료

여기에서는 표준 클래스 라이브러리에 포함 된 예외를 인스턴스화했지만 Exception 클래스를 상속하여 자신도 자신의 예외 클래스를 정의하고 인스턴스화 할 수 있다. 이러한 직접 만든 예외 클래스 형 오브젝트는 자동으로 발생하지 않기 때문에, throw 문을 사용하여 명시 적으로 예외를 발생시킬 수 있어야 한다.

3. 사용자 Exception

적절한 예외 클래스를 상속하여 사용자 정의 예외 클래스를 만들 수 있다. 예외 클래스는 java.lang.Throwable 의 서브 클래스이며, throw 로 예외를 발생 시키고 메소드의 throws 목록에 추가 할 수 있다. 또한 수퍼 클래스에 선택한 예외 클래스는 해당 클래스에 정의 된 특수 메서드를 사용할 수 있다.

이 때, RuntimeException 을 상속하면 catch 하거나 메소드의 throws 목록에 지정할 필요가 없다. 그러나 명시적인 예외의 throw 는 반환 값이나 인수와 같은 메소드 인터페이스의 일부 이기에 의미있는 것이라고 말할 수 있다. 일반적으로 Exception 클래스에서 상속하여 상위의 제어를 잡는다.

구체적인 예를 만든 후 거기에 맞추어 예외 클래스를 정의 해 보자
다음 예제는 사용자 이름의 배열을 유지하는 클래스이다. 인스턴스화 때 생성자에 사용자 이름의 배열을 준다. 특정 사용자를 나타내는 요소의 인덱스를 해당 사용자의 ID 로 ID 에서 사용자 이름을 사용자 이름에서 ID 를 얻을 수 있다.

```
public class My_UserList {  
    private String[] users;  
  
    // 사용자의 배열을 할당  
    void setUsers(String[] args) {  
        // 배열의 참조 값 할당  
        users = args;  
    }  
  
    // UID에서 사용자 이름을 검색  
    String getUser(int uid) {  
        return users[uid];  
    }  
  
    // 사용자 이름에서 UID를 취득
```

```

int getUser (String user) {
    for (int i = 0; i < users.length; i++) {
        if (user.equals (users [i])) {
            return i;
        }
    }
    // 정수 값을 돌려 줄 필요가 있기 때문에,
    // 여기에서는 기본값은 0으로했다.
    return 0;
}
}

```

이 클래스는 문자열의 배열을 유지하고 메소드는 그 게터 (getXXX ())와 세터 (setXXX ())가 정의되어 있다. 이 클래스를 사용 컨트롤 클래스를 다음과 같이 정의 해 보자.

```

public class My_UserTest {
    public static void main(String[] args) {
        // 인스턴스화
        My_UserList obj = new My_UserList();
        // 사용자 배열 세트
        obj.setUsers(args);
        System.out.print("dominica->");
        System.out.println(obj.getUser("dominica"));
        System.out.print("0 ->");
        System.out.println(obj.getUser(0));
    }
}

```

위에서 언급 한 UserList 클래스는이 클래스와 같은 파일에 기술해도 상관하지 않고, 나누어도 상관 없다, 같은 디렉토리에 존재하고 있어야 한다.

실행후 커맨드 창에 root team01 team02 dominica system admin 이라고 입력 후 실행한다.

[실행결과]

dominica->3

0 ->root

>>사용자 정의 예외 만들기

위 프로그램을 발생할 수 있는 다음과 같은 실행 오류를 생각해본다.

- ① 사용자가 없으면 기본값 "0"을 반환하는 경우
- ② "uid"의 적정 값이 할당되지 않을 경우
- ③ 사용자 목록이 존재하지 않는 경우가 고려되지 않은 경우

이러한 문제를 해결하기 위해, 여기에 사용자 예외 클래스를 정의 해 보자

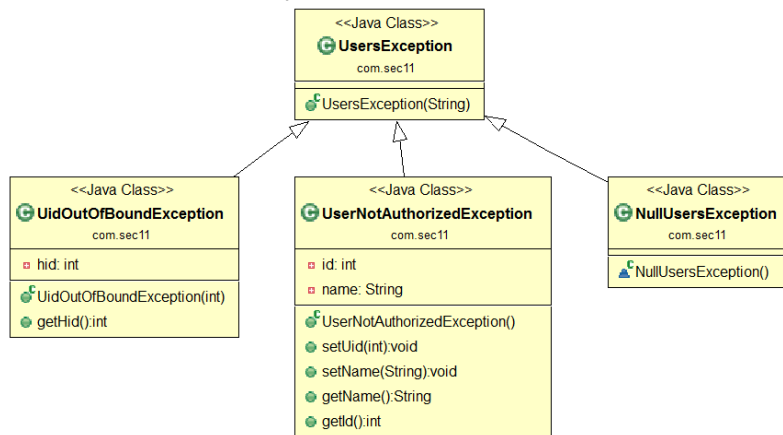
UserNotAuthorizedException : 사용자를 찾을 수 없는 경우에 발생

UidOutOfBoundException : 지정된 UID가 적절한 범위가 아닌 경우에 발생

NullUsersException : 사용자가 설정되기 전에 사용자가 요청 된 경우에 발생

이처럼 자신의 정의 예외를 만드는 경우는 끝에 Exception 하는 것이 명명 관례 (naming convention)이다.

.이 예외 클래스는 Exception 클래스의 파생클래스로 만든다.



```
public class UsersException extends Exception {
    public UsersException (String str) {
        super (str);
    }
}

class UserNotAuthorizedException extends UsersException {
    // UID
    private int id;
    // 사용자 이름
    private String name;
    // 생성자
    public UserNotAuthorizedException () {
        super ( "이 사용자는 인증되지 않습니다.");
    }
}
```



```







    }
    public void setUid (int i) {
        id = i;
    }
    public void setName (String user) {
        name = user;
    }
    public String getName () {
        return name;
    }
    public int getId () {
        return id;
    }
}

class UidOutOfBoundException extends UsersException {
    // UID의 상한
    private int hid;
    // 생성자 (인수는 UID 제한)
    public UidOutOfBoundException (int i) {
        super ( "UID의 범위는"+0 + "에서"+ i + "입니다.");
        hid = i;
    }
    public int getHid () {
        return hid;
    }
}

class NullUsersException extends UsersException {
    // 생성자
    NullUsersException () {
        super ( "사용자가 정의되어 있지 않습니다.");
    }
}

```

사용자 예외를 다음과 같이, UserList 클래스를 이러한 예외를 throw하도록 편집한다.

<<Java Class>>	
	UserList
com.sec11	
users: String[]	
	UserList()
	setUsers(String[]):void
	getUsers():String[]
	getUser(int):String
	getUser(String):int

```
public class UserList {
    private String [] users;
    void setUsers (String [] args) {
        users = args;
    }
    String [] getUsers () throws NullUsersException {
        if (users.equals (null)) {
            throw new NullUsersException ();
        }
        return users;
    }
    String getUser (int uid) throws UsersException {
        if (users.equals (null)) {
            throw new NullUsersException ();
        } else if (uid < 0 || uid >= users.length) {
            UidOutOfBoundException excep
                = new UidOutOfBoundException (users.length - 1);
            throw excep;
        }
        return users [uid];
    }
    int getUser (String user) throws UsersException {
        int uid = 0;
        if (users.length == 0) {
            throw new NullUsersException ();
        }
    }
}
```

```

    } else {
        for (int i = 0; i < users.length; i++) {
            if (user.equals (users [i])) {
                uid = i;
                break;
            } else if (i == users.length - 1) {
                UserNotAuthorizedException excep
                    = new UserNotAuthorizedException ();
                excep.setName (user);
                throw excep;
            }
        }
    }
    return uid;
}
}
}

```

UsersList 클래스의 private 이 아닌 인터페이스 부분을 수정했기 때문에, 이 클래스를 인스턴스화하는 컨트롤 클래스 분도 수정한다.

```

public class UserTest {
    public static void main(String[] args) {
        UserList obj = new UserList ();
        obj.setUsers (args);
        try {
            System.out.print ( "dominica ->");
            System.out.println (obj.getUser ( "dominica"));
            System.out.print ( "10 ->");
            System.out.println (obj.getUser (10));
        } catch (UsersException e) {
            System.out.println ( "");
            System.out.println (e);
            return;
        }
    }
}

```

}

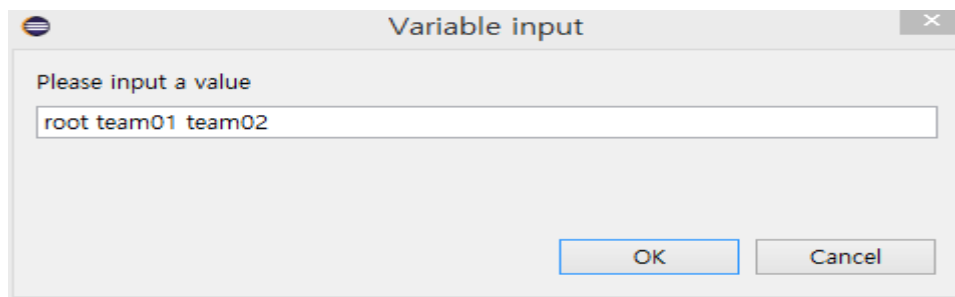
컴파일하고 실행한다. 명령 줄 인수가 사용자 이름의 배열이다. 인수를 주지 않고 실행하면 실행하면 사용자가 정의되어 있지 않기 때문에 예외 `NullUsersException` 가 발행한다.

[실행결과]

dominica ->

[com.ch11.NullUsersException](#): 사용자가 정의되어 있지 않습니다.

인수를 주어서 해당 사용자가 존재해야 `UserNotAuthorizedException` 가 발생하는 것입니다. 실행시 커맨드 창에 `root team01 team02` 라고 입력 후 실행



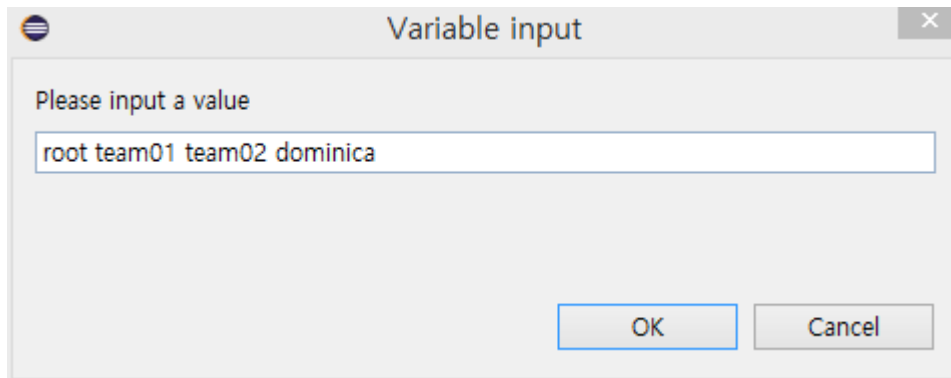
[실행결과]

dominica ->

[com.ch11.UserNotAuthorizedException](#): 이 사용자는 인증되지 않습니다.

또한 배열의 인덱스를 UID 로 사용자 이름을 입력 할 수 있어야 하지만, UID 가 배열의 요소 수를 초과하거나 음수이거나하면 `UidOutOfBoundException` 가 발생한다.

root team01 team02 dominica 를 입력 후 실행



[실행결과]

dominica ->3

10 ->

[com.ch11.UidOutOfBoundsException](#): UID 의 범위는 0 에서 3 입니다.