

# 1. Java api document

❖ 1.7 API: <http://docs.oracle.com/javase/7/docs/api/>

❖ 1.8 API: <http://docs.oracle.com/javase/8/docs/api/>

❖ 예제 코드

✓ <http://www.java2s.com/Code/JavaAPI/CatalogJavaAPI.htm>

✓ <http://www.java2s.com/Tutorial/Java/CatalogJava.htm>

## 2. 예외(Exception)

- ❖ 컴파일 에러(error): .java 파일을 .class 파일로 만드는 과정에서 발생하는 오류로 .java 파일에 오류가 있거나 jvm이 인식할 수 없는 클래스 사용으로 인한 오류
- ❖ 예외(exception): 컴파일은 성공적으로 수행되어 클래스가 만들어 졌지만 실행 도중 외부 요인이나 잘못된 입력 등으로 발생하는 오류
- ❖ 논리적 에러: 컴파일 이나 런타임 시에 에러가 발생하지는 않지만 정상적으로 실행이 되었지만 의도하지 않은 결과가 나오는 경우
- ❖ 단언(assert): 개발자가 의도적으로 특정 조건을 만족했을 때 만 프로그램이 정상적으로 동작하도록 만드는 것
- ❖ 컴파일 에러가 발생하면 에러를 수정
- ❖ 컴파일 에러는 eclipse 나 IntelliJ, NetBeans와 같은 IDE를 사용하면 명확하게 표시를 해주므로 비교적 찾기 쉬움
- ❖ 런타임 에러나 논리적 에러는 프로그램 외부의 문제인지 논리적인 오류인지 여러 가지 상황을 고려해야 하므로 에러의 원인을 찾기 어려운 경우가 많음

## 2. 예외(Exception)

- ❖ 예외나 논리적인 오류의 원인을 밝혀내고 수정하는 과정이 Debugging
- ❖ 디버깅 방법
  - ✓ System.out.print 메소드를 이용해서 값을 출력해보는 방법 - 로깅(Logging)
  - ✓ IDE가 제공하는 Debugging Tool을 사용하는 방법
  - ✓ 파일이나 데이터베이스에 로그를 남기는 Logging Tool을 사용하는 방법
  - ✓ 테스트를 자동으로 수행해주는 테스트 툴을 사용하는 방법



## 2. 예외(Exception)

❖예외: 문법적으로는 이상이 없어서 컴파일 시점에는 아무런 문제가 없지만 프로그램 실행 중에 발생하는 예기치 않은 사건으로 인해 프로그램이 중단되는 런타임 에러

❖예외가 발생하는 경우

- ✓ 정수를 0으로 나누는 경우
- ✓ 배열의 첨자가 범위를 벗어나거나 음수를 사용하는 경우
- ✓ 부적절한 형 변환이 일어나는 경우
- ✓ 입출력을 위한 파일이 없는 경우 등

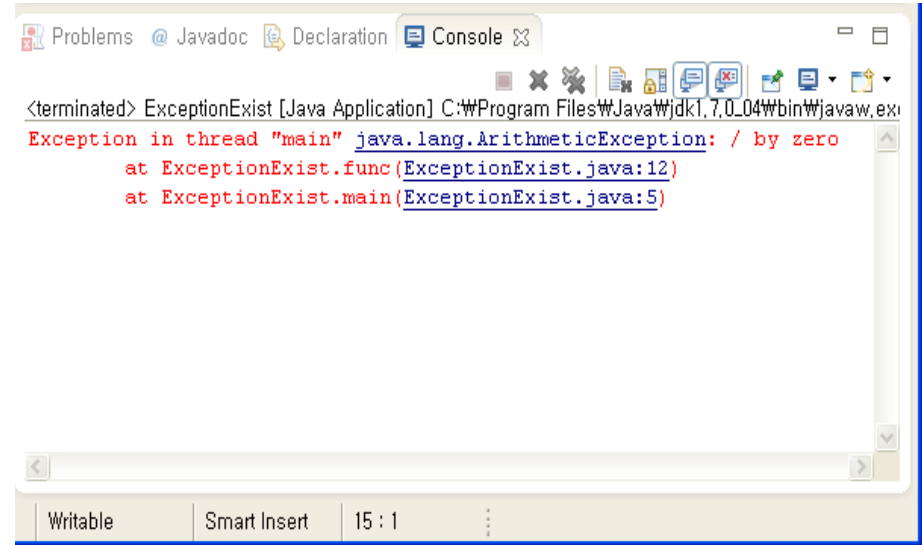
❖예외처리의 용도

- ✓ 정상 종료
- ✓ 예외내용 기록
- ✓ 예외 발생 시 무시하고 계속 실행
- ✓ 정상적인 값으로 변경해서 실행

# 실습(ExceptionExist .java)

```
public class ExceptionExist
{
    public static void main(String[] args)
    {
        func();          //메소드 호출
    }

    public static void func()
    {
        int i = 1;
        int j = 0;
        System.out.println(i/j);    // 1을 0으로 나눈다. 예외 발생
    }
}
```



# 변수의 값 확인

- ❖ 메모리에 저장된 값을 확인하는 방법
  - ◆ System.out.print() 메소드를 이용해서 직접 값을 콘솔에 출력
  - ◆ IDE에서 제공하는 Debugging Tool을 사용
- ❖ eclipse의 Debugging Tool 사용
  - ◆ 값을 확인하고자 하는 곳에 breakpoint를 삽입
  - ◆ [Run] - [Debug]를 실행



# 실습(ExceptionExist.java)

```
1 public class ExceptionExist
2 {
3     public static void main(String[] args)
4     {
5         func();                //메소드 호출
6     }
7
8     public static void func()
9     {
10        int i = 1;
11
12        // 1을 0으로 나눈다. 예외 발생
13    }
14 }
```

Toggle Breakpoint Ctrl+Shift+B  
Disable Breakpoint Shift+Double Click  
Go to Annotation Ctrl+1  
Add Bookmark...  
Add Task...  
Show Quick Diff Ctrl+Shift+Q  
Show Line Numbers  
Folding  
Preferences...  
Breakpoint Properties... Ctrl+Double Click

javaWjdk1.8.0\_60\bin\javaw.exe (2016. 1. 12. 오전 8:35:40)  
ArithmeticException: / by zero  
ExceptionExist.java:12)

# 실습(ExceptionExist .java)

Debug - Exception/src/ExceptionExist.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Quick Access Java EE Java Debug

Debug Servers

- ExceptionExist [Java Application]
  - ExceptionExist at localhost:1435
    - Thread [main] (Suspended (breakpoint at line 11 in ExceptionExist))
      - ExceptionExist.func() line: 11
      - ExceptionExist.main(String[]) line: 5

C:\Program Files\Java\jdk1.8.0\_60\bin\javaw.exe (2016. 1. 12. 오전 8:43:05)

(x)= Variables Breakpoints

Name	Value
i	1

ObjectMain.java ExceptionExist.java

```
7
8 public static void func()
9 {
10     int i = 1;
11     int j = 0;
12     System.out.println(i/j); // 1을 0으로 나눈다. 예외 발생
13 }
14 }
15
```

Outline

- ExceptionExist
  - main(String[]) : void
  - func() : void

Console Tasks

ExceptionExist [Java Application] C:\Program Files\Java\jdk1.8.0\_60\bin\javaw.exe (2016. 1. 12. 오전 8:43:05)



# 3. 예외 관련 클래스

❖ 자바는 예외를 하나의 객체로 취급

❖ 예외 관련 클래스는 java.lang.Throwable 클래스를 상속

❖ 예외 관련 클래스의 종류

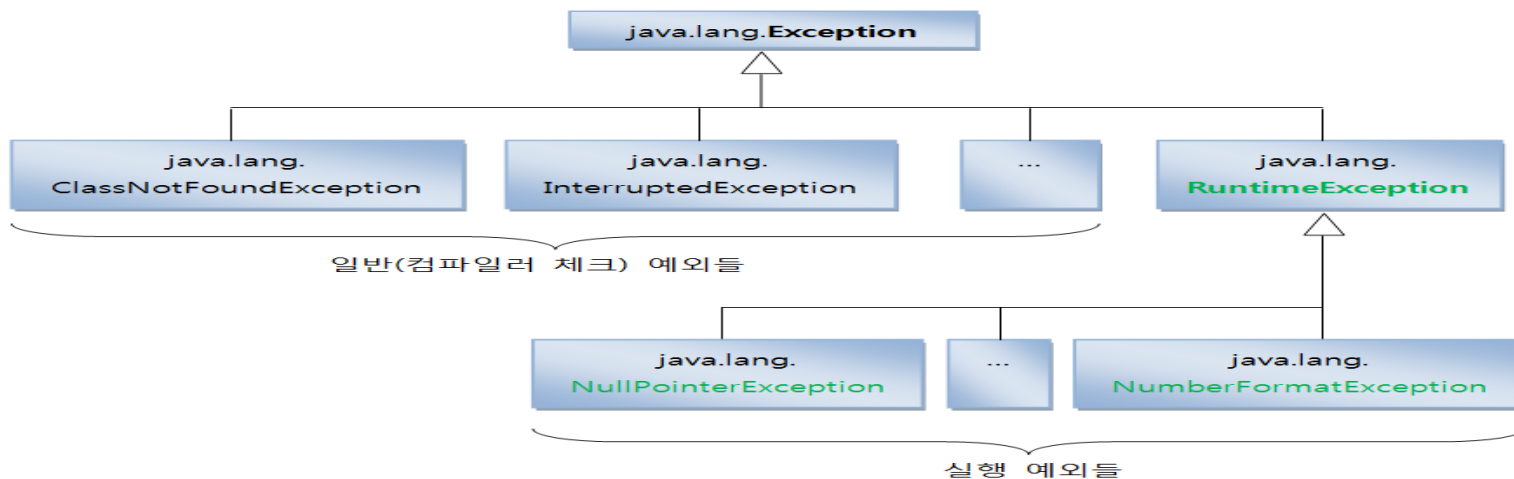
✓ 예외 관련 클래스의 계층 구조: Object -> Throwable -> Error

-> Exception

❖ 자바 예외 처리 최상위 클래스: Throwable

❖ Error는 메모리 부족이나 스택 오버플로우와 같이 일단 발생하면 복구할 수 없는 심각한 오류를 나타내는 클래스

❖ Exception 클래스는 Error 보다는 덜 심각한 일반적인 예외를 나타내기 위한 클래스



# 3.예외 관련 클래스

## ❖예외 분류

- ✓ 예외는 일반적인 예외와 RuntimeException으로 구분
- ✓ RuntimeException – 실행 시 JVM에서 발생하는 예외로 컴파일 할 때 검사하지 않는 예외
- ✓ RuntimeException을 제외한 예외가 일반적인 예외
- ✓ 일반적인 예외는 컴파일러 체크 예외라고도 하는데 자바소스를 컴파일 할 때 예외처리 코드가 필요한지 검사를 하는데 이러한 예외는 예외처리 코드를 작성하지 않으면 에러
- ✓ RuntimeException은 컴파일 할 때 예외 처리 코드의 필요성 여부를 체크하지 않음
- ✓ RuntimeException은 컴파일러가 체크하지 않으므로 개발자의 경험에 의해서 예외 처리 코드를 삽입해야 함

# 3.예외 관련 클래스

## ❖Exception 클래스의 주요 하위 클래스들

NoSuchMethodException	메소드가 존재하지 않을 때
ClassNotFoundException	클래스가 존재하지 않을 때
CloneNotSupportedException	객체의 복제가 지원되지 않는 상황에서 복제를 시도하고자 하는 경우
IllegalAccessException	클래스에 대한 부정적인 접근
InstantiationException	추상클래스나 인터페이스로부터 객체를 생성하고자 하는 경우
InterruptedException	스레드가 인터럽트 되었을 때
RuntimeException	실행시간에 예외가 발생한 경우
IOException	입출력과 관련된 예외 처리 EOFException,FileNotFoundException,InterruptedException

# 3.예외 관련 클래스

❖ RuntimeException 클래스의 주요 하위 클래스: 프로그래머의 실수로 발생하는 예외로 소스 코드를 수정하면 해결되는 경우가 많음

ArithmeticException	0으로 나누는 등의 산술적인 예외
NegativeArraySizeException	배열의 크기를 지정할 때 음수의 사용
NullPointerException	null인 참조형 변수가 메소드나 멤버 변수에 접근하고자 하는 경우
ArrayIndexOutOfBoundsException	배열에서 인덱스 범위를 초과해서 사용하는 경우
NumberFormatException	숫자로 변경이 되지 않는 데이터를 숫자로 변경했을 때 발생
ClassCastException	객체의 타입을 변경이 불가능한 타입으로 변경하는 경우
SecurityException	보안을 이유로 메소드를 수행할 수 없을 때

# 3.예외 관련 클래스

## ❖ Throwable 클래스의 주요 멤버

- ✓ `public String getMessage()` : 예외 객체의 상세 메시지를 문자열로 리턴하는데 생성자에서 넘겨받은 문자열을 리턴
- ✓ `public void printStackTrace()`: 예외 객체 및 그 백 트레이스를 표준 에러 스트림에 출력

## 4.예외 처리

### ❖예외를 처리하는 방법

- ✓ 예외가 발생한 메소드 내에서 처리하는 방법(try, catch 절 사용)
- ✓ 예외가 발생한 메소드를 호출한 메소드에게 예외의 처리를 넘겨주는 방법(throws 절 사용)



# 실습(ExceptionProblem.java)

## ❖예외의 발생

- ✓ 예제는 정상적으로 동작하다가 마지막에 비 정상적인 메시지를 출력
- ✓ 배열의 요소가 3개인데 4번째 출력하고자 했으므로 3번째 출력문까지는 정상적으로 수행하지만 4번째 출력에서 문제가 발생하기 때문

```
public class ExceptionProblem
```

```
{
```

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

```
    {
```

```
        int ar[] = {10,20,30};
```

```
        for(int i=0 ; i <= ar.length ; i++)
```

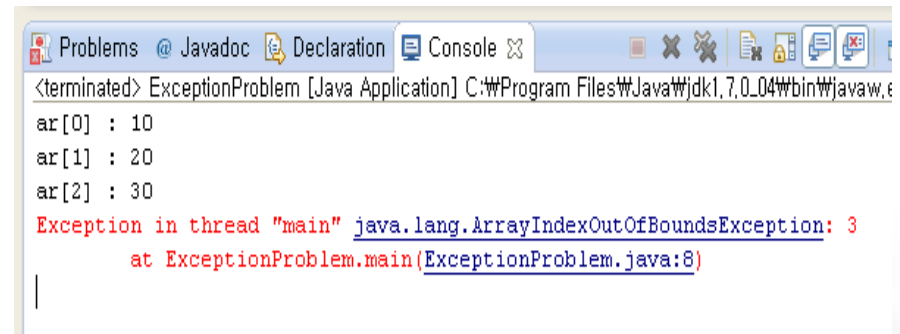
```
        {
```

```
            System.out.println("ar[" + i + "]" : " + ar[i]);
```

```
        }
```

```
    }
```

```
}
```



```
Problems @ Javadoc Declaration Console  
<terminated> ExceptionProblem [Java Application] C:\Program Files\Java\jdk1.7.0_04\bin\javaw.exe  
ar[0] : 10  
ar[1] : 20  
ar[2] : 30  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3  
    at ExceptionProblem.main(ExceptionProblem.java:8)  
|
```

## 4.예외 처리

❖예외 처리 방법 중 예외가 발생한 메소드 내에서 직접 처리하는 방법: try, catch, finally 블록 사용(finally를 사용하는 경우 catch 생략 가능) – if 문 처럼 동작

```
try {  
    ..... // try 블록: 예외가 발생할 가능성이 있는 문장을 지정  
}  
catch(예외타입1 매개변수1) {  
    ..... // 예외 처리 블록1: 예외의 종류에 따라 처리하는 처리  
}  
catch(예외타입N 매개변수N) {  
    ..... // 예외 처리 블록 N  
}  
finally{  
    ..... // finally 블록: 예외의 발생여부와 상관없이 무조건 수행되는 블록  
    // 생략가능  
}
```

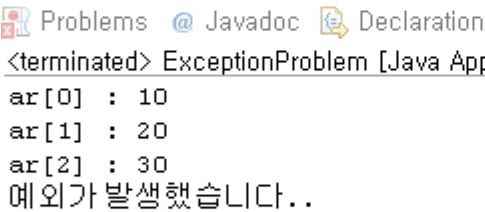


## 4.예외 처리

- ❖ try-catch 문에서 예외가 발생한 경우와 발생하지 않은 경우의 처리 흐름이 달라짐
  - ✓ try 블록 내에서 예외가 발생한 경우에는 발생한 예외를 처리할 수 있는 catch 블록이 있는지 검색한 후 일치하는 catch 블록이 있는 경우 그 블록 내의 처리를 수행하고 finally 블록이 있으면 finally 블록의 처리를 수행
  - ✓ 예외가 발생한 상태에서 일치하는 catch 블록이 없을 때는 예외는 처리되지 않음
  - ✓ 예외가 발생하지 않은 경우에는 try-catch 블록을 바로 빠져나가서 finally가 있으면 finally로 가고 없으면 빠져나감
- ❖ 여러 개의 catch를 묶어서 한꺼번에 처리할 수 있는데 여러 Exception 클래스의 상위 클래스를 이용해서 묶거나 JDK 1.7 버전 부터는 | 를 이용해서 여러 개의 Exception을 묶는 것이 가능
- ❖ 하나의 try 문을 처리하는 여러 개의 catch 블록이 있을 때 위쪽의 catch 블록에 기재된 예외 클래스가 뒤에 나오는 catch 블록의 예외 클래스보다 상위 클래스라면 뒤쪽의 코드가 unreachable code가 되므로 주의

# 실습(TryCatch.java)

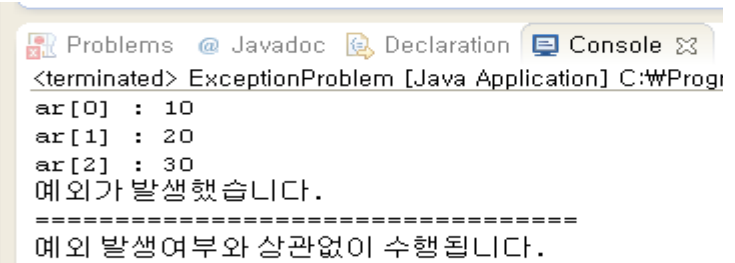
```
public class TryCatch {  
    public static void main(String[] args)  
    {  
        int[] ar = {10,20,30};  
        try  
        {  
            for(int i=0 ; i <= 3 ; i++)  
            {  
                System.out.println("ar[" + i + "]" : " + ar[i]);  
            }  
        }  
        catch (Exception e)  
        {  
            //예외 클래스 이름은 Exception이라고 해도 되고 ArrayIndexOutOfBoundsException 이  
            라고 해도 됩니다.  
            System.out.println("예외가 발생했습니다..");  
        }  
    }  
}
```



Problems @ Javadoc Declaration  
<terminated> ExceptionProblem [Java App  
ar[0] : 10  
ar[1] : 20  
ar[2] : 30  
예외가 발생했습니다..

# 실습(TryCatchFinally.java)

```
public class TryCatchFinally {  
    public static void main(String[] args)  
    {  
        int[] ar = {10,20,30};  
        try{  
            for(int i=0 ; i <= 3 ; i++){  
                System.out.println("ar[" + i + "]" : " + ar[i]);  
            }  
        }  
        catch (Exception e)  
        {  
            System.out.println("예외가 발생했습니다.");  
        }  
        finally  
        {  
            System.out.println("=====");  
            System.out.println("예외 발생여부와 상관없이 수행됩니다.");  
        }  
    }  
}
```



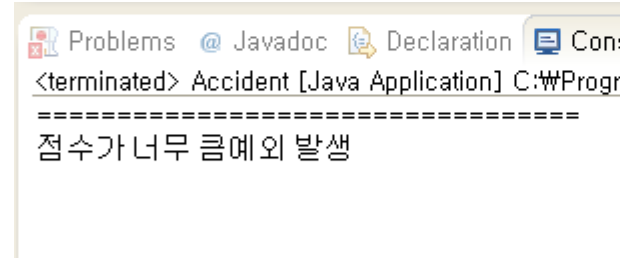
```
Problems @ Javadoc Declaration Console  
<terminated> ExceptionProblem [Java Application] C:\WPProgr  
ar[0] : 10  
ar[1] : 20  
ar[2] : 30  
예외가 발생했습니다.  
=====  
예외 발생여부와 상관없이 수행됩니다.
```

## 5. 예외의 인위적 발생

- ❖ 프로그램에서 강제로 예외를 발생시킬 수 있음
- ❖ 예외를 강제로 발생시키기 위해서는 throw 문 사용
  - throw 예외객체이름;
  - 또는
  - throw new 예외클래스(매개변수);

# 실습(Accident.java)

```
public class Accident
{
    public static void main(String args[])
    {
        try
        {
            int jumsu = Integer.parseInt(args[0]);
            if(jumsu>100)
            {
                throw new NumberFormatException("점수가 너무 큼");
                // 예외의 인위적 발생
            }
        }
        catch(NumberFormatException e)
        {
            System.out.println("=====");
            System.out.println(e.getMessage() + "예외 발생");
        }
    }
}
```



## 6.호출한 메소드에 예외 전달

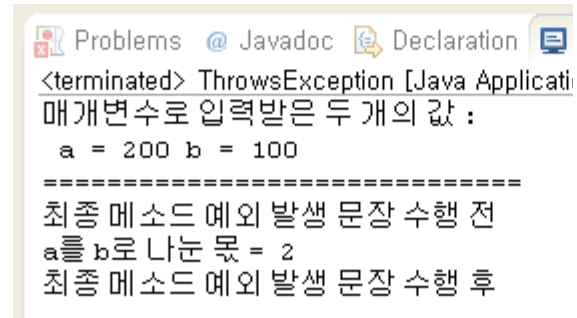
- ❖예외를 직접 처리하지 않고 자신을 호출한 메소드에게 예외를 넘겨주는 방법(throws 절 사용)
- ❖현재 메소드에서 예외처리를 하기가 어렵거나 호출되는 여러 개의 메소드에서 발생하는 예외가 동일한 경우 현재 영역을 호출한 곳에서 예외를 대신 처리해 달라며 예외 객체를 양도 하는 것
- ❖메소드 선언 시 다음과 같은 방법으로 지정합니다.

`public 결과형 메소드명(매개변수) throws 예외클래스`

- ❖main 메소드에서 예외를 throws 하게 되면 프로그램의 모든 영역에서 예외를 직접 처리하지 않아도 되지만 권장하지는 않음

# 실습(ThrowsException .java)

```
public class ThrowsException {  
    static int a, b;  
  
    public static void main(String args[])  
    {  
        try {  
            a = Integer.parseInt("12a");  
            b = Integer.parseInt("0");  
            method1();  
        }  
        catch(ArithmeticException e) {  
            System.out.println("ArithmeticException 처리 루틴 : ");  
            System.out.println(e + " 예외 발생");  
        }  
        catch(NumberFormatException e){  
            System.out.println("NumberFormatException 처리 루틴 : ");  
            System.out.println(e + " 예외 발생");  
        }  
    }  
}
```



Problems @ Javadoc Declaration

<terminated> ThrowsException [Java Applicati  
매개변수로 입력받은 두 개의 값 :  
a = 200 b = 100  
=====

최종 메소드 예외 발생 문장 수행 전  
a를 b로 나눈 몫 = 2  
최종 메소드 예외 발생 문장 수행 후

# 실습(ThrowsException.java)

```
        catch(Exception e)
        {
            System.out.println("나머지 모든 예외 처리 루틴 : ");
            System.out.println(e + " 예외 발생");
        }
    }

    public static void method1() throws NumberFormatException {
        System.out.println("매개변수로 입력받은 두 개의 값 :");
        System.out.println(" a = " + a + " b = " + b);
        System.out.println("=====");
        method2();
    }

    public static void method2() throws ArithmeticException {
        System.out.println("최종 메소드 예외 발생 문장 수행 전");
        System.out.println("a를 b로 나눈 몫 = " + (a / b));
        System.out.println("최종 메소드 예외 발생 문장 수행 후");
    }
}
```



# 7. 사용자 정의 예외 클래스

- ❖ 사용자가 새로운 예외 클래스를 정의하여 사용할 수 있음
- ❖ 예외에 대한 정보를 변경하려고 할 때 사용자가 직접 작성한 후 예외를 강제로 발생시켜 원하는 결과를 얻는데 목적이 있음
- ❖ 사용자 정의 Exception을 작성하기 위해서는 Exception으로부터 상속을 받는 것이 유용
- ❖ Throwable로부터 상속받지 않는 이유는 Error 클래스 때문
- ❖ 입출력에 관련된 예외 만을 작성하기 위해 IOException으로부터 상속을 받는 것도 가능

```
class UserException1 extends Exception
{
    // 사용자 정의 예외는 Exception 클래스로부터 상속
    public UserException1(String message)
    {
        // 생성자 메소드
        super(message);
        // 상위 클래스인 Exception 클래스의 생성자를 호출하여
        // 예외 객체 생성
    }
}
```

# 실습(UserException.java)

```
import java.util.*;
```

```
// 사용자 정의 예외는 Exception 클래스로부터 상속
class UserException1 extends Exception {
    private static final long serialVersionUID = 1L;

    public UserException1(String message) {
        super(message);
    }
}
```

```
class UserException2 extends Exception {

    private static final long serialVersionUID = 1L;

    public UserException2(String message) {
        super(message);
    }
}
```

# 실습(UserException.java)

```
public class UserException  
{
```

```
    public static void main(String args
```

```
        Scanner in = new Scanner(System.in);
```

```
        try
```

```
        {
```

```
            System.out.print("하나의 숫자를 입력하세요=>");
```

```
            int jumsu = in.nextInt();
```

```
            if(jumsu < 0)
```

```
            {
```

```
                throw new UserException1("점수가 너무 작음");
```

```
            }
```

```
            else if (jumsu > 100)
```

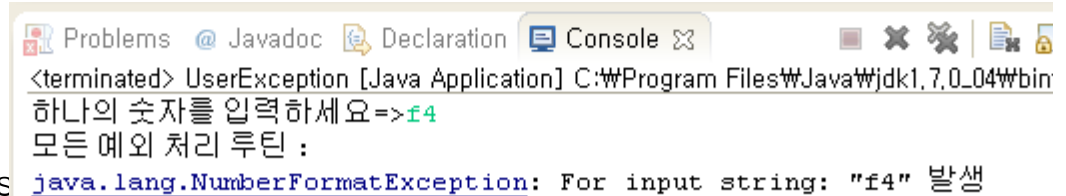
```
            {
```

```
                throw new UserException2("점수가 너무 큼");
```

```
            }
```

```
            System.out.println("정상적인 점수 입력");
```

```
        }
```



# 실습(UserException.java)

```
catch(UserException1 e) {  
    System.out.println("UserException1 처리 루틴 : ");  
    System.out.println(e + " 발생");  
}  
catch(UserException2 e) {  
    System.out.println("UserException2 처리 루틴 : ");  
    System.out.println(e + " 발생");  
}  
catch(Exception e) {  
    System.out.println("모든 예외 처리 루틴 : ");  
    System.out.println(e + " 발생");  
}  
}  
}
```

# 8.try-with-resources

❖try(자원 생성) 구문을 이용하게 되면 자원 생성 위치에서 생성한 객체는 예외 발생 여부에 상관 없이 자원 해제가 자동으로 이루어 지는데 자원은 AutoCloseable 인터페이스를 implements 한 클래스 만 가능

❖아래처럼 작성하면 fis나 fos는 close 메소드를 호출하지 않아도 자동으로 닫는 메소드인 close()를 호출

```
try (FileInputStream fis = new FileInputStream(srcFile);
    FileOutputStream fos = new FileOutputStream(dstFile))
{
    int b;
    while ((b = fis.read()) != -1)
        fos.write(b);
}
```

## 9. 예외처리 권장 사항

- ❖ 가능하면 예외처리 구문을 사용하지 않고 논리적으로 해결하고 특히 반복문 내에서의 예외처리 구문을 피하는 것을 권장
- ❖ 가능하면 표준 예외 클래스를 이용해서 예외를 던지는 것을 권장
- ❖ catch 처리 구문에서 여러 종류의 예외를 한꺼번에 처리하고자 하는 경우에는 매개변수를 Throwable 보다는 Exception으로 처리하는 것을 권장

# 10.ASSERTION

- ❖ ASSERTION이란 특정 지점에서의 특정 조건을 검사하여 강제로 예외를 발생시켜 프로그램을 중단시키는 것
- ❖ 어느 특정 메소드의 인자 값은 100이하이어야 한다면 이 조건을 검사해서 이런 경우에만 프로그램이 실행될 수 있도록 해주는 것이 Assertion
- ❖ Assertion은 JDK1.6 버전 이하에서만 정상 수행되고 실행 할 때 -ea 옵션을 설정해야 함

## ❖ 형식

`assert [boolean식];`

`assert [boolean식]: “메시지”;`

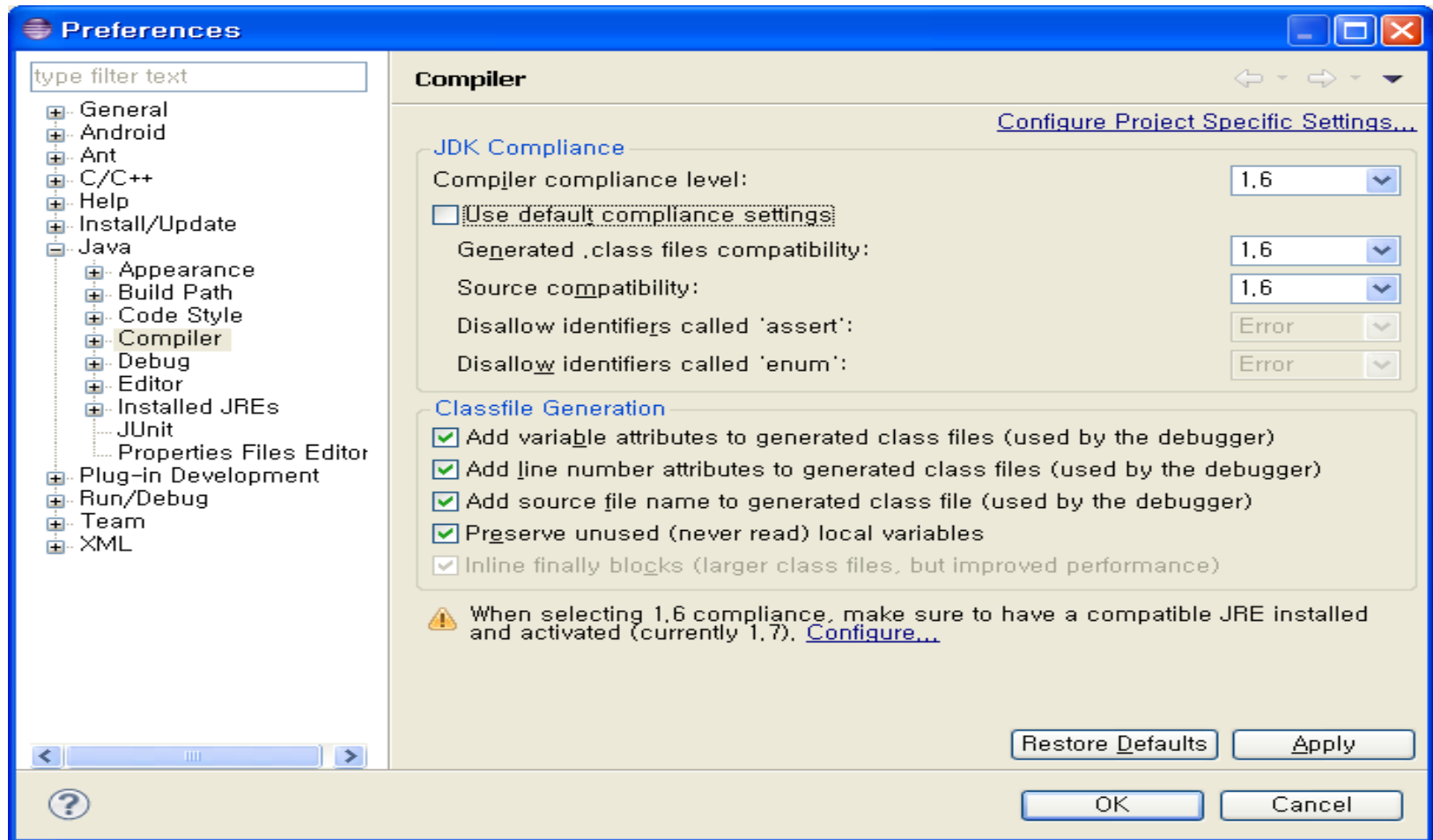
`assert var > 10; 10보다 큰 값이어야 함`

`assert var < 10 : 10보다 작은 값이어야 함`

`assert str != null : “str에 null값이 들어오면 안됨!”;`

# 10.ASSERTION

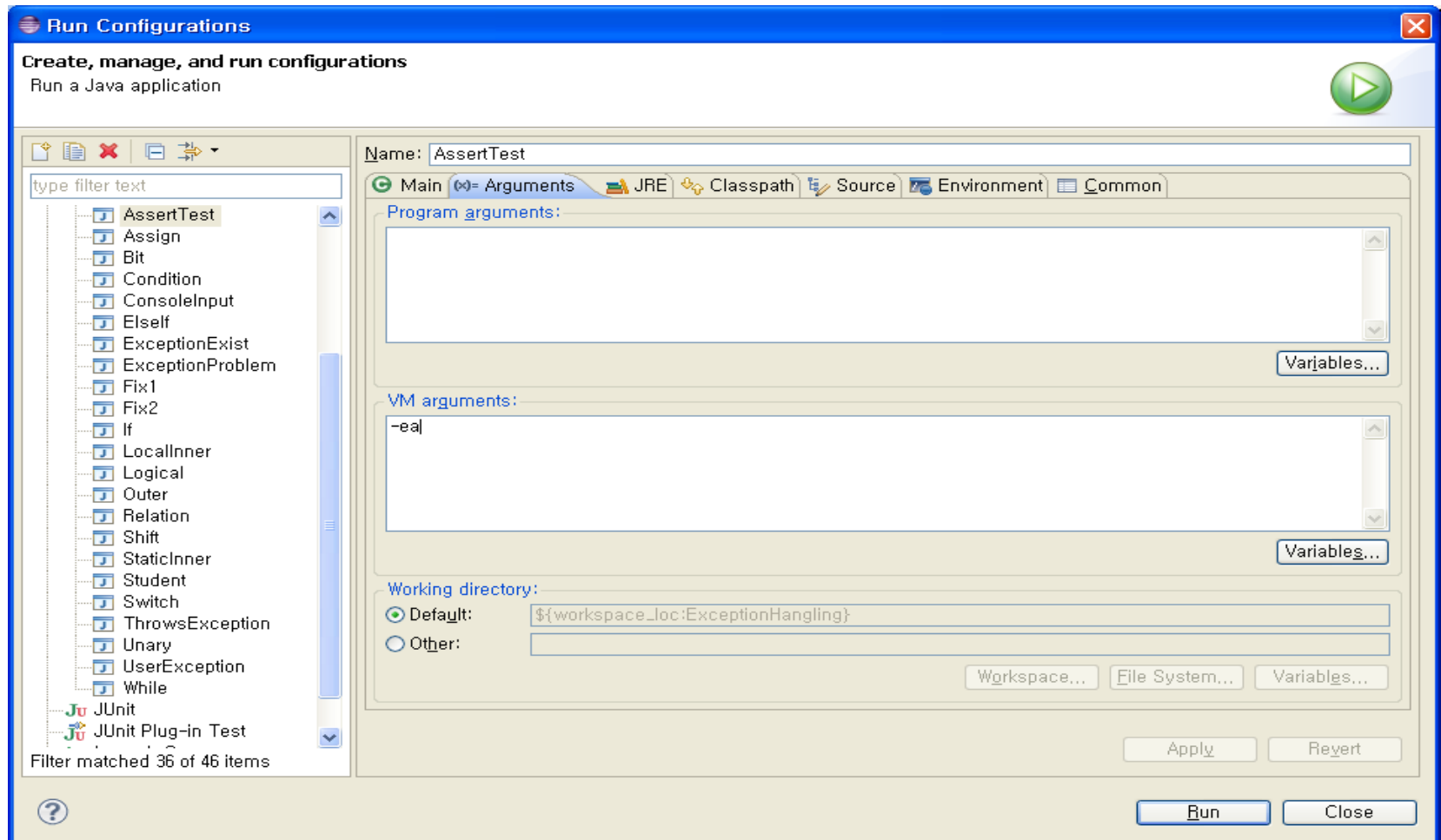
## ❖ 1.6 버전 컴파일





# 10.ASSERTION

## ❖ 실행 시 옵션



# 실습(AssertTest.java)

```
import java.util.*;

public class AssertTest {
    public static void main(String args[]) {
        int a;
        Scanner in = new Scanner(System.in);
        try{
            System.out.print("점수를 입력하세요:");
            a = in.nextInt();
            assert (a < 100 && a >= 0): "올바르지 못한 점수를 입력하셨습니다.";
            System.out.println("올바른 점수를 입력하셨습니다");
        }
        catch(Exception e)
        {
            System.out.println("예외 발생");
        }
        finally{
            in.close();
        }
    }
}
```

