

Java



Study Point :

- Package의 정의와 활용을 배운다.
- JDK5.0에 추가된 Static import를 배운다.
- Exception의 정의와 필요성을 배우고 User가 직접 Exception을 작성하면서 얻게 되는 효율성을 알아본다.
- Assertion의 역할에 대해서 배운다.

package :

- package?

- ➡ Java에서 package는 서로 관련 있는 class와 interface를 하나의 단위로 묶는 것을 의미, 일종의 Library다.

- package Declaration Process

- ➡ Package Declaration은 Comment을 제외하고 반드시 Source file의 첫 줄에 와야 한다,

```
package package path name;
```

- 예제들을 C:\kostaEx\PackageTest라는 folder에 또는 독자가 기억할 만한 곳에 저장한다.

package :

```
01 package myPack.p1;
```

```
02 public class MyPackOne{
```

```
03
```

```
04     public void one(){
```

```
05         System.out.println("MyPackOne class의 one method");
```

```
06     }
```

```
07 }
```

```
01 package myPack.p1;
```

```
02 public class MyPackTwo{
```

```
03
```

```
04     public void two(){
```

```
05         System.out.println("MyPackTwo class의 two method");
```

```
06     }
```

```
07 }
```

package :

- 두 개의 file이 저장된 곳으로 cmd창을 연다. 다음과 같이 compile을 수행하고 표를 참조하여 Option에 대한 설명을 확인해본다.

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Michael>cd C:\JavaEx\PackTest

C:\JavaEx\PackTest>javac -d . MyPackOne.java
C:\JavaEx\PackTest>javac -d . MyPackTwo.java
C:\JavaEx\PackTest>_
```

※ 패키지 컴파일 시 옵션

옵션	설명
javac [옵션] [작업위치] [소스파일명]	-d: 디렉토리(패키지)를 의미하며 컴파일되어 생기는 클래스 파일이 저장될 위치를 말하는 것이다. 즉, 디렉토리(패키지) 작업을 의미하는 것이며 디렉토리(패키지)를 만들어야 한다면 만들라는 뜻이다.
	.: 디렉토리(패키지) 작업은 바로 현재 디렉토리에서 하라는 뜻이다.

package :

Package using process.

- ▶ package에 있는 특정한 class를 사용하려면 [import문]을 사용한다. import는 수입이란 의미가 되므로 현재 Object에서 원하는 다른 Object를 가져다 사용할 때 사용. Java interpreter는 import하는 package path의 class들은 CLASSPATH환경변수에 지정된 path에서 검색한다. package문 아래에 오는 것이 기본 방법이며 다음과 같다.

```
import [package path.class name]; 또는 import  
[package path.*];
```

```
01 import myPack.p1.MyPackOne;  
02 import myPack.p1.MyPackTwo;  
03 class MyPackTest {  
04  
05     public static void main(String[] args) {  
06         MyPackOne myOne = new MyPackOne();  
07         myOne.one();  
08         MyPackTwo myTwo = new MyPackTwo();  
09         myTwo.two();  
10     }  
11 }
```

package :

❖ static import문

- JDK5.0이전에서는 Constant 또는 static으로 선언된 것들을 사용하려면 해당 [class name,Constant] 등으로 접근했다. 하지만 JDK5.0에서부터는 static import를 사용하여 보다 쉽고 빠르게 static Constant 또는 method등을 호출한다. 이 때문에 작성된 Source분석이 조금 어려워졌다는 단점도 있다.

```
import static [package path,class name.*];
```

```
import static [package path,class name.Constant field name];
```

- 다음은 간단하게 Math class의 random()를 사용하여 난수를 받아 출력하는 예제이다,

```
01 import static java.lang.Math.*;
02 import static java.lang.System.out;
03 class StaticImpTest {
04
05     public static void main(String[] args) {
06         int i = (int)(random()*26+65);
07         out.println((char)i);
08     }
09 }
```

exception :

❁ Exception

- ➡ 지하철을 탈때까지만 해도 날씨가 좋았는데 도착하여 지하철역에서 나오려니 비가내리고 있다던가 또는 책을 보다가 종이에 손을 베었다던가 하는 뜻하지 않은 일들을 당하게된다. 이렇게 예상하지 못한 일들을 'exception' 라 하고 대비하고 준비하는 것이 'exception handling' 다.

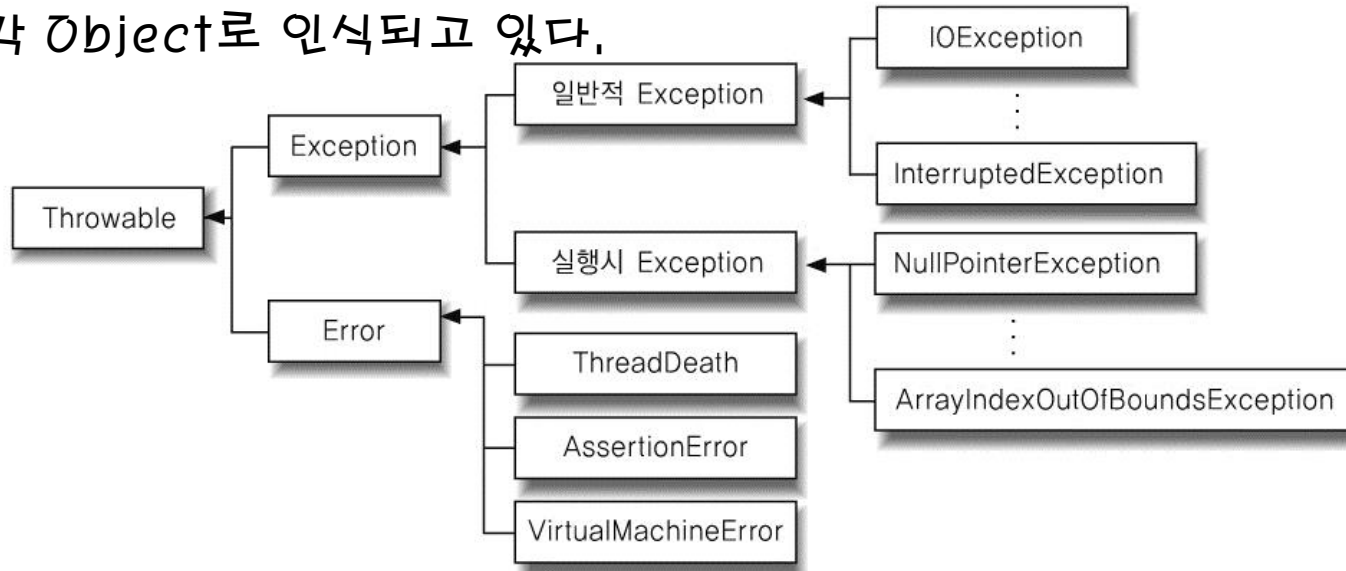
❖ Exception handling 대한 필요성과 이해

- Java에서 program의 실행하는 도중에 exception이 발생하면 발생한 그 시점에서 program이 바로 종료가 된다. Exception이 발생 했을 때 program을 종료시키는 것이 바른 판단일 수도 있다. 하지만 가벼운 exception이거나 예상을 하고 있었던 exception라면 program을 종료시키는 것이 조금은 가혹 (?) 하다, 그래서 'exception handling' 라는 mechanism이 제안되었고 exception handling를 통해 우선 program의 비 정상적인 종료를 막고 발생한 exception에 대한 처리로 정상적인 program을 계속 진행할 수 있도록 하는 것이 exception handling의 필요성이라 할 수 있다.

exception :

❖ exception의 종류

- Java에서 발생하는 모든 exception은 다음과 같은 구조를 이루면서 각각 Object로 인식되고 있다.



※ 오류의 구분

오류구분	설명
예외(Exception)	가벼운 오류이며 프로그램적으로 처리한다.
오류(Error)	치명적인 오류이며 JVM에 의존하여 처리한다.

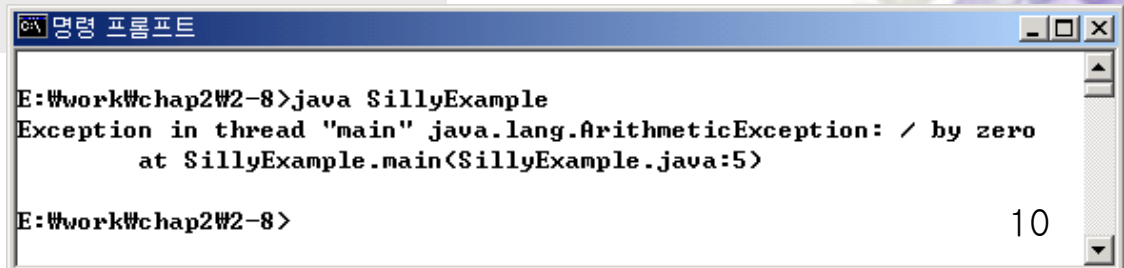
예외처리 - Exception

08. 익셉션 처리에 사용되는 try 문

익셉션이란?

- 익셉션(exception, 예외) : 자바에서 에러를 지칭하는 용어
- [예제 2-48] 익셉션을 발생하는 프로그램의 예

```
1  class SillyExample {  
2      public static void main(String args[]) {  
3          int a = 3, b = 0;  
4          int result;  
5          result = a / b;  
6          System.out.println(result);  
7          System.out.println("Done.");  
8      }  
9  }
```



```
명령 프롬프트  
E:\work\chap2\2-8>java SillyExample  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
    at SillyExample.main(SillyExample.java:5)  
E:\work\chap2\2-8>
```

예외처리 - Exception

08. 익셉션 처리에 사용되는 try 문

try 문

- try 문 : 익셉션 처리에 사용되는 명령문
- 기본 형식

```
try
    try블록
catch (익셉션타입 익셉션변수) } ----- catch 절(catch clause). 반복 가능
    catch블록
finally
    finally블록 } ----- finally 절(finally clause). 생략 가능
```

- try 블록, catch 블록, finally 블록은 모두 중괄호로 둘러싸인 블록이어야 함

예외처리 - Exception

08. 익셉션 처리에 사용되는 try 문

try 문

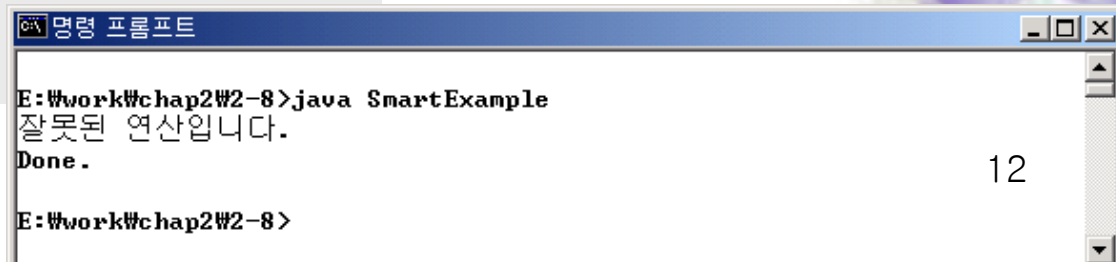
- [예제 2-49] try 문의 사용 예

```
1  class SmartExample {  
2      public static void main(String args[]) {  
3          int a = 3, b = 0;  
4          int result;  
5          try {  
6              result = a / b;  
7              System.out.println(result);  
8          }  
9          catch (java.lang.ArithmeticException e) {  
10             System.out.println("잘못된 연산입니다.");  
11         }  
12         finally {  
13             System.out.println("Done.");  
14         }  
15     }  
16 }
```

이 부분을 실행하다가
익셉션이 발생하면

이 부분을 실행합니다.

이 부분은 익셉션 발생 유무와
상관없이 실행됩니다.



```
E:\work\chap2\2-8>java SmartExample  
잘못된 연산입니다.  
Done.  
E:\work\chap2\2-8>
```

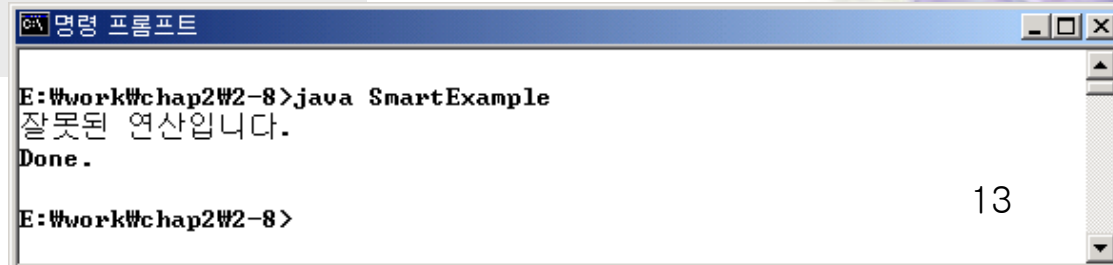
예외처리 - Exception

08. 익셉션 처리에 사용되는 try 문

try 문

- [예제 2-50] finally 블록이 없는 try 문의 예

```
1  class SmartExample {
2      public static void main(String args[]) {
3          int a = 3, b = 0;
4          int result;
5          try {
6              result = a / b;
7              System.out.println(result);
8          }
9          catch (java.lang.ArithmeticException e) {
10             System.out.println("잘못된 연산입니다.");
11          }
12         System.out.println("Done.");
13     }
14 }
```



```
명령 프롬프트
E:\work\chap2\2-8>java SmartExample
잘못된 연산입니다.
Done.
E:\work\chap2\2-8>
```

예외처리 - Exception

08. 익셉션 처리에 사용되는 try 문

- [예제 2-51] 두 종류의 익셉션을 처리하는 try 문의 예

```
1  class TryExample {
2      public static void main(String args[]) {
3          int divisor[] = { 5, 4, 3, 2, 1, 0 };
4          for (int cnt = 0; cnt < 10; cnt++) {
5              try {
6                  int share = 100 / divisor[cnt];
7                  System.out.println(share);
8              }
9              catch (java.lang.ArithmeticException e) {
10                 System.out.println("잘못된 연산입니다.");
11             }
12             catch (java.lang.ArrayIndexOutOfBoundsException e) {
13                 System.out.println("잘못된 인덱스입니다.");
14             }
15         }
16         System.out.println("Done.");
17     }
18 }
```

이 명령문은 0으로 나눌 때와 인덱스를 잘못 썼을 때
익셉션을 발생할 수 있습니다.

0으로 나눌 때의 익셉션을 처리합니다.

인덱스를 잘못 썼을 때의 익셉션을
처리합니다.

```
명령 프롬프트
E:\work\chap2\2-8>java TryExample
20
25
33
50
100
잘못된 연산입니다.
잘못된 인덱스입니다.
잘못된 인덱스입니다.
잘못된 인덱스입니다.
잘못된 인덱스입니다.
잘못된 인덱스입니다.
Done.

E:\work\chap2\2-8>
```

exception :

- Nothing exception handling ex

```
01 import static java.lang.System.out;
02 class ExceptionEx1 {
03
04     public static void main(String[] args) {
05         int[] var = {10,200,30};
06         for(int i=0 ; i <= 3 ; i++)
07             out.println("var["+i+"] : "+var[i]);
08
09         out.println("프로그램 끝!");
10     }
11 }
```

exception :

- 앞의 예제가 Array의 범위를 벗어나
ArrayIndexOutOfBoundsException이 발생했다.
- try~catch문을 사용하여 'exception handling' 를 수행하자.

```
try{  
    // exception 이 발생 가능한 문장들;  
}catch(예상되는_exception Object 변수명){  
    // 해당 exception가 발생했을 시 수행할 문장들;  
}
```

- try brace에 정의되는 문장은 exception이 발생 가능한 문장들을 기재하는 곳이니 주의한다. try brace는 반드시 하나이상의 catch brace 아니면 finally brace가 같이 따라 줘야 한다. 앞의 [ExceptionEx1]예제를 수정해 보자!

exception :

```
01 import static java.lang.System.out;
02 class ExceptionEx2 {
03
04     public static void main(String[] args) {
05         int[] var = {10,200,30};
06         for(int i=0 ; i <= 3 ; i++){
07             try{
08                 out.println("var["+i+"] : "+var[i]);
09             }catch(ArrayIndexOutOfBoundsException ae){
10                 out.println( "Array Overflow.");
11             }
12         }//for의 끝
13
14         out.println( "program end!");
15     }
16 }
```

exception :

- try ~ catch문에서의 주의 사항

아무리 try ~ catch문으로 'exception 처리' 를 했다 해도 모든 것이 해결되는 것은 아니다, 다음 예문을 살펴 보자!

```
07 try{
08     out.println((i+1)+"번째");
09     out.println("var["+i+"] : "+var[i]);
10     out.println("~~~~~"); //수행을 못할 수도 있음!
11 }catch(ArrayIndexOutOfBoundsException ae){
12     out.println( "Array Overflow");
13 }
```

- 위의 8번 Line과 10번 Line의 문장을 추가한 후 program을 다시 compile하고 실행.

총 반복문이 4번을 반복하게 되는데 4번째 반복 수행을 8번 행의 출력으로 알 수 있지만 exception이 발생하는 9번 행을 만나면서 10번 행을 수행하지 못하고 바로 catch scope을 수행함을 알 수 있다.

exception :

- multiple catch문

multiple catch문은 하나의 try문 내에 여러 개의 exception이 발생 가능할 때 사용.

```
try{  
    // exception가 발생 가능한 문장들;  
} catch(예상되는_exception Object1 변수명){  
    // 해당 exception가 발생했을 시 수행할 문장들;  
} catch(예상되는_exception Object2 변수명){  
    // 해당 exception가 발생했을 시 수행할 문장들;  
} catch(예상되는_exception Object3 변수명){  
    // 해당 exception가 발생했을 시 수행할 문장들;  
}
```

exception :

```
01 import static java.lang.System.out;
02 class ExceptionEx3 {
03
04     public static void main(String[] args) {
05         int var = 50;
06         try{
07             int data = Integer.parseInt(args[0]);
08
09             out.println(var/data);
10         }catch(NumberFormatException ne){
11             out.println("숫자가 아닙니다.");
12         }catch(ArithmeticException ae){
13             out.println("0으로 나눌 순 없죠?");
14         }
15         out.println("프로그램 종료!");
16     }
17 }
```

exception :

- multiple catch문의 주의 사항

일반적 exception에서 가장 parent class가 exception이다, 그러므로 가장 아래쪽에 정의 해야 한다, 이유는 exception은 parent class가 모든 exception를 가지고 있으므로 가장 위에 정의를 하게 되면 모든 exception를 처리하게 되므로 두 번째 catch문부터는 절대로 비교 수행할 수 없게 된다,

```
try{
    수행문1;
    수행문2;
    수행문3;
    ...
    수행문n;
}catch(예상되는_예외객체1 변수명){
    ...;
}catch(예상되는_예외객체2 변수명){
    ...;
}catch(예상되는_예외객체3 변수명){
    ...;
}
```



exception :

- throws keyword

exception를 처리하기 보다는 발생한 exception Object를 양도하는 것이다, 즉, 현재 method에서 exception처리를 하기가 조금 어려운 상태일 때 현재 영역을 호출해준 곳으로 발생한 exception Object를 대신 처리해 달라며 양도하는 것이다, 사용법은 다음의 구성과같이 throws라는 keyword를 통해 method를 선언하는 것이다,

[Access Modifier] [Return Type] [method name](args1, ...argsn) throws exception class1,...exception classn{}

```
04      public void setData(String n) throws NumberFormatException{
05          if(n.length() >= 1){
06              String str = n.substring(0,1);
07              printData(str);
08          }
09      }
```

- [throws]를 사용하여 발생한 exception Object의 양도는 것은 어디까지나 양도이지 exception에 대한 처리는 아니다, 양도를 받은 곳에서도 다시 양도가 가능하지만 언젠가는 try~catch문으로 해결을 해야 Program의 진행을 계속 유지 할 수 있음!

exception :

❖ finally의 필요성

- exception가 발생하든 발생하지 않든 무조건 수행하는 부분이 바로 finally영역이다, 이것은 뒤에서 Database처리나 File처리를 한다면 꼭 필요한 부분이다, 이유는 Database를 열었다거나 또는 File을 열었다면 꼭 닫아주고 Program이 종료되어야 하기 때문이다,
- 구성,

```
try{  
    // exception가 발생 가능한 문장들;  
} catch(예상되는_exception객체1 변수명){  
    // 해당 exception가 발생했을 시 수행할 문장들;  
} finally{  
    // exception발생 여부와 상관없이 수행할 문장들;  
}
```

exception :

```
01 import static java.lang.System.out;
02 class FinallyEx1 {
03
04     public static void main(String[] args) {
05         int[] var = {10,200,30};
06         for(int i=0 ; i <= 3 ; i++) {
07             try{
08                 out.println((i+1)+"번째");
09                 out.println("var["+i+"] : "+var[i]);
10                 out.println("~~~~~");
11             }catch(ArrayIndexOutOfBoundsException ae){
12                 out.println("배열을 넘었습니다.");
13                 return;
14             }finally{
15                 out.println("!!!!!! Finally !!!!!!! 무조건 실행됨");
16             }
17         } //for의 끝
18
19         out.println("프로그램 끝!");
20     }
21 }
```


exception :

❖ User Definition exception

- User Definition Exception이 필요한 이유는 표준exception가 발생할 때 exception에 대한 정보를 변경하거나 정보를 수정하고자 한다면 User가 직접 작성하여 보안된 exception를 발생시켜 원하는 결과를 얻는데 있다,
- User Definition Exception을 작성하기 위해서는 Throwable을 받지 않고 그 하위에 있으면서 보다 많은 기능들로 확장되어 있는 Exception으로부터 Inheritance 받는 것이 유용하다, 물론 I/O에 관련된 exception를 작성하기 위해 IOException으로부터 Inheritance 받는 것이 보편적인 것이다,

exception :

```
01 public class UserException extends Exception{
02
03     private int port = 772;
04     public UserException(String msg){
05         super(msg);
06     }
07     public UserException(String msg, int port){
08         super(msg);
09         this.port = port;
10     }
11     public int getPort(){
12         return port;
13     }
14 }
```

exception :

```
01 class UserExceptionTest {
02
03     public void test1(String[] n) throws UserException{
04         System.out.println("Test1");
05         if(n.length < 1)
06             throw new UserException("아무것도 없다네"); // 강제 exception 발생
07         else
08             throw new UserException("최종 예선",703); // 강제 exception 발생
09     }
10     public static void main(String[] args) {
11         UserExceptionTest ut = new UserExceptionTest();
12         try{
13             ut.test1(args);
14         }catch(UserException ue){
15             // System.out.println(ue.getMessage());
16             ue.printStackTrace();
17         }
18     }
19 }
```

Assertion :

❖ Assertion

- Assertion? Programmer 자신이 전개하고 있는 Code 내용에서 Programmer가 생각하고 있는 움직임과 그리고 특정 지점에서의 Program상의 설정 값들이 일치하고 있는지를 검사할 수 있도록 하는 것이 바로 Assertion이다, 예로 어느 특정 method의 Argument 값은 10이상이어야 한다는 Programmer의 확고함이 있다고 하자! 이럴 때 Assertion을 사용하여 Programmer가 주장하는 확고함을 조건으로 명시하고 그 조건을 만족할 때만 Code가 실행할 수 있도록 하는 것이 Assertion이다,
- Assertion과 exception(Exception)의 차이점은?
Exception은 특정한 Code에서 exception가 발생하므로 일어나는 비정상적인 Program 종료와 같은 1차적인 손실을 막고 exception에 대한 처리로 인해 Program의 신뢰성을 높이는 것, 하지만 Assertion은 어떤 결과를 위해 특정 Code나 변수의 값을 Programmer가 예상하는 값이어야 하는 것을 검증하는 것에 차이가 있다,

Assertion :

❖ Assertion의 문법

- Assertion의 문법은 다음 두 가지로 나눈다.
 - `assert [boolean식];`
 - `[boolean식]`은 항상 `true` 아니면 `false`인 `boolean`형의 결과를 가지는 표현식이 되어야 한다.
만약 `boolean식`의 결과가 `false`일 경우에는 `AssertionError`가 발생하여 수행이 정상적으로 진행되지 않는다.
 - `assert [boolean식] : [표현식];`
 - `[boolean식]`의 결과값이 `false`일 경우에 `[표현식]`을 수행한다. 여기에는 일반 값일 수도 있고 특정 `method`를 호출하여 받은 `return` 값일 수도 있다. `[표현식]`에는 문자열로 변환이 가능한 값이 무조건 와야 한다.

Assertion :

● 간단한 예문

- ▶ `assert var > 10;`
- ▶ `assert var < 10 : "10보다 작은 값이어야 함!" ;`
- ▶ `assert str.equals("");`
- ▶ `assert !str.equals("");`
- ▶ `assert str != null : "str에 null값이 들어오면 안됨!" ;`

● 컴파일 및 실행 법

- ▶ Java Compiler는 'Assertion' 기능을 수행하지 않으면서 Compile을 하게 되어 있다, 그러므로 Compiler에게 'Assertion' 기능을 부여하면서 Compile을 수행하도록 하기 위해서는 다음과 같은 옵션을 주어야 한다.
 - `java -source 1.5 [File name]`
- ▶ 실행 할 때도 마찬가지로 다음과 같이 옵션을 주면서 실행해야 한다.
 - `java -ea [class name]`

-ea : Enable Assertions라고 해서 Assertion기능을 사용 가능하게 하는 Option이다,
-da : Disable Assertions라고 해서 ea의 반대 Option이다,

Assertion :

- 2단에서 9단까지만 입력 받아 처리 하도록 범위를 설정하는 부분에 Assertion 기능을 부여 하는 구구단 예제를 구현.

```
01 import static java.lang.System.out;
02 class AssertTest2 {
03     public void gugu(int dan){
04         assert dan > 1 && dan < 10 : "2~9단중 하나를 입력하세요";
05         out.println(dan+"단");
06         out.println("-----");
07         StringBuffer sb = new StringBuffer();
08         for(int i=0;i<9;i++){
09             sb.delete(0,sb.length());
10             sb.append(dan);
11             sb.append("*");
12             sb.append(i+1);
13             sb.append("=");
14             sb.append(dan*(i+1));
15             out.println(sb.toString());
16         }
17     }
18     public static void main(String[] args){
```

Assertion :

```
19
20     AssertTest2 at = new AssertTest2();
21     try{
22         int dan = Integer.parseInt(args[0]);
23         at.gugu(dan);
24     }catch(Exception e){
25         e.printStackTrace();
26     }
27 }
28 }
```

- 이렇게 해서 'Assertion' 에 대해 공부해 보았고 이는 exception적인 것들을 하나씩 Object화 하여 그것을 처리하게 하는 'exception처리'와는 다소 차이가 있음을 알았다. Assertion은 Programmer의 확고함을 검증하고 그 사실을 인정하여 Program의 신뢰성을 높였지만 사실 약간의 번거로움이 있어 사용자들로 하여금 이점은 인정하지만 외면당하는 경향이 있다.

영

어

하

시

다

^^!