정규표현식(Regular Expressions)과 J2SDK 1.4

작성자 : 김성박(삼성 SDS 멀티캠퍼스 전임강사)
email : urstory@nownuri.net
homepage : http://sunny.sarang.net

- 해당 문서는 http://sunny.sarang.net "JAVA강좌란" 혹은 http://www.javastudy.co.kr의 "칼럼"란 에서만 배포합니다. 문서는 계속 버전업 될 수 있습니다. 필자의 허락 없이 수정, 삭제, 재작성, 이동 등을 할 수 없습니다.
- 잘못된 부분을 발견하거나, 추가할 사항이 있다면 http://sunny.sarang.net의 자바 관련게시판에 글을 남겨주세요.
- 해당 문서는 이미 jdk1.4 이상이 설치되어 있는 것을 가정으로 합니다.

정규표현식과 자바.

정규표현식이란 텍스트(text)를 기술하기 위한 표현방식을 말합니다. 정규표현식은 텍스트를 탐색하거나 문자열을 조작하는데 있어서 강력한 문법을 제공하여 줍니다.

정규표현식은 주로 유닉스 시스템과 펄(perl)과 같은 언어에서 많이 사용 되어왔으며, 그 말은 유닉스 시스템과 펄은 문자열을 아주 쉽게 처리할 수 있다는 것을 의미합니다.

이러한 정규표현식이 J2SDK 1.4에 이르러서야 공식적으로 기본 Package로 제공되게 되었습니다. J2SDK 1.4이전에는 정규표현식을 이용하기 위하여는 Apache Jakarta 프로젝트의 Regexp와 같은 package를 많이 이용하였습니다.

1. 정규표현식의 문법

정규표현식이란 문자열 안에서의 일정한 패턴을 표현하기 위한 표현식을 말합니다. 이러한 표현식은 펄, 유닉스 시스템 그리고 자바에서 공통적으로 사용될 수 있습니다. 정규표현식을 자바언어에서 이용하기 전에 먼저 정규표현식을 표현하는 방법부터 자세히 알 필요가 있습니다.

문자열 안에서 hello 라는 말이 들어가 있는지, 혹은 숫자가 안에 들어가 있는지 알아보고자 할 경우에 사용됩니다. 정규표현식을 잘 사용한다는 말은, 찾고자 하는 패턴에 대한 표현을 여러 가지 특수문자들을 이용하여 잘 표기한다는 것을 의미합니다. 이러한 패턴을 표기하기 위한 특수문자에 대하여 잘 알아보도록 하겠습니다.

1-1) '.' 특수 문자.

'.' 특수 문자는 임의의 한 문자를 나타냅니다. 예를 들면, h.s 라는 패턴은 반드시 'h'로 시작하며 's'로 반드시 끝나는 단어를 의미합니다. 따라서 'his', 'hos' 등은 모두 해당 패턴에 일치하는 문자열이라고 말할 수 있습니다. 그러나'hs'와 같은 문자열은 패턴에 일치하지 않습니다. 그 이유는 '.'특수문자가 위치한 곳에는 반드시 한글자가 위치하여야 하기 때문입니다.

예)

패턴	일치하는 문자열
a.b	acb , adb , azd 등
ab.	abc , abd , abz 등
.bc	abc , dbc , zbc 등

1-2) '*' 특수문자.

'*' 특수문자는 바로 앞의 문자를 나타냅니다. '*' 는 바로 앞의 문자가 없거나 하나 이상 반복한다는 의미를 뜻합니다. 만약 'a*' 라고 쓰여있는 패턴이 있다면, 'a', 'aa', 'aaa' 등이 가능합니다. '*' 특수문자를 사용할 때 주의하여야 할 점은 패턴을 지정할 때 '*' 앞에는 한 글자 이상의 단어가 반드시 와야 한다는 것입니다.

예)

패턴	일치하는 문자열
hello*	hell , hello, helloo, hellooo 등
ab*c	abc , abbc , abbbc, abbbbc 등
*d	불가능

1-3) '+' 특수문자

'+' 특수문자는 '*'특수문자와 흡사하지만, '*'특수문자와는 달리 반드시 하나 이상의 문자가 반복을 해야 한다는 것입니다. 예를 들자면 'hello+' 라고 패턴을 지정 한다면 'hello', 'helloo' , 'hellooo' 등 '+'특수문자 바로 앞의 문자는 반드시 하나이상 반복해야 합니다.

예)

패턴	일치하는 문자열
hello+	hello , helloo, hellooo, helloooo 등
ab+ c	abc , abbc , abbbc , abbbbc 등
+ d	불가능

1-4) '?' 특수문자

'?' 특수문자의 바로 앞의 문자가 하나가 있거나, 없거나 하는 것을 의미합니다. 예를 들자면 'a?c' 라고 패턴을 지정한다면 'c' , 'ac' 중에서 하나를 의미하게 됩니다.

예)

패턴	일치하는 문자열
hello?	hell, hello
try?	tr, try

1-5) '^' 특수문자

'^' 특수문자는 문장의 처음을 나타냅니다. 예를 들어 '^Hello' 라고 작성된 패턴이 있다면 해당 패턴에 일치하기 위하여 문자열은 반드시 'Hello' 로 시작을 하여야 합니다.

예)

패턴	일치하는 문자열
^Hello	Hello World

^The	The pen

1-6) '$' 특수문자

'$' 특수문자는 문장의 끝을 나타냅니다. 예를 들어 'World$' 라고 작성된 패턴이 있다면 해당 패턴에 일치하기 위하여 문자열은 반드시 'World'로 끝을 맺어야 합니다.

예)

패턴	일치하는 문자열
World$	Hello PHP World
PHP$	Start PHP

1-7) '[ ]' 특수문자

대괄호 '[ ]' 특수문자의 의미는 괄호 안의 문자 중 일치하는 것을 찾고자 할 경우 사용합니다. 예를 들자면 '[abc]' 라고 패턴을 작성하게 되면 문자열에 'a' 나 'b' 혹은 'c' 등이 있어야 합니다.

예)

패턴	일치하는 문자열
[abc]	a, b, c, ab, abc 등
[a-z]	소문자가 포함된 모든 문자열
[A-Z]	대문자가 포함된 모든 문자열
[0-9]	숫자가 포함된 모든 문자열
^[a-zA-Z0-9]	숫자나 영문자로 시작되는 모든 문자열

1-8) '[ ]' 특수문자 안에서의 '^' 특수문자

대괄호 '[ ]' 특수문자 안에서 '^' 특수문자가 쓰이게 되면 '[ ]' 특수문자 안에 있는 문자와 일치하지 않는 문자열을 포함하고 있는 패턴을 의미 합니다. 즉 '[^abc]de' 라고 패턴을 작성하게 되면 'ade', 'bde', 'cde' 를 제외한 '.de' 패턴과 같은 의미가 됩니다.

예)

패턴	일치하는 문자열
[^abc]de	dde, fde, zde 등
a[^0-9]c	abc, acd, add 등

1-9) '{ }' 특수문자

중괄호 '{ }' 특수문자는 '{ }' 특수문자 앞의 문자나 문자열의 반복되는 개수를 말합니다. 예를 들자면 'hel{2}o' 라는 패턴을 작성하게 되면 'hello' 문자열이 문자열에 포함되어 있어야 됩니다.

예)

패턴	일치하는 문자열
gu{5}ggle	guuuuugle
gu{3,}ggle (u가 3개 이상)	guuuggle, guuuuggle, guuuuuggle 등
gu{2,4}ggle	guuggle, guuuggle, guuuuggle

1-10) '( )' 특수문자

소괄호 '( )' 특수문자는 '( )' 특수문자 안의 글자들을 하나의 문자로 봅니다. 예를 들어 'gu(gg){2}le' 와 같은 패턴을 작성하게 되면 'gugggggle' 문자열이 문자열에 포함되어 있어야 됩니다.

예)

패턴	일치하는 문자열
(hello){3}	hellohellohello, hey hellohellohello 등
(hello)*	NULL, hello, hellohello, hellohello 등

1-11) '|' 특수문자

패턴 안에서 OR연산을 사용할 때 사용합니다. 예를 들어 'hi|hello' 는 hi 나 hello 가 포

함되어있는 문자열을 의미합니다.

예)

패턴	일치하는 문자열
man|woman	man , woman, manwoman , superman 등
left|right	left, right, leftright 등

1-12) 문자 클래스(character class)

'[]' 특수문자 안에서 자주 사용되는 패턴들을 미리 키워드로 정의하여 놓은 것을 문자 클래스라고 합니다. 해당 문자 클래스를 J2SDK 1.4에서는 다르게 사용을 합니다. (문자 클래스는 PHP와 같은 언어에서는 이용되지만 JAVA에서는 이용되지 않습니다. J2SDK 1.4의 예를 참고하십시요.)

패턴	문자 클래스	J2SDK 1.4의 예
[a-zA-Z] (모든 영문자)	[[:alpha:]]	₩p{Alpha}
[0-9] (숫자)	[[:digit:]]	₩p{Digit}
[a-zA-Z0-9] (영문자와 숫자)	[[:alnum:]]	₩p{Alnum}
공백	[[:space:]]	₩p{Space}

1-13) 정규표현식에서 특수문자의 사용

1-12) 까지 말한 특수문자를 정규표현식의 패턴에서 사용하려면 해당 특수문자 앞에 역 슬래시('₩')를 붙입니다. 물론 역 슬래시('₩')를 패턴에서 사용 하려면 역 슬래시를 2개 연속하여 써주어야 합니다.

2. J2SDK 1.4.x에서의 정규표현식 이용

J2SDK 1.4.x에서 정규표현식을 이용하기 위한 방법으로는 크게, java.lang.String class에 새롭게 추가된 메소드를 이용하는 방법과 java.util.regex 패키지를 이용하는 방법이 있습니다. 2가지 방법에 대하여 모두 알아보도록 하겠습니다.

2-1) String class에 새롭게 추가된 정규표현식 관련 메소드

메소드 이름	설명
boolean matches(String regex)	문자열 안에 regex정규표현식과 일치하는 패턴이 있다면 true, 그렇지 않으면 false를 반환합니다.
String replaceAll(String regex, String replacement)	문자열 안에 regex정규표현식으로 일치하는 모든 부분을 replacement문자열로 치환한 결과를 반환합니다.
String replaceFirst(String regex, String replacement)	문자열 안에 regex정규표현식으로 일치하는 첫 부분을 replacement문자열로 치환한 결과를 반환합니다.
String[] split(String regex)	문자열 안에 regex정규표현식과 일치하는 부분을 기준으로 분할하여 배열로 반환합니다.

예제1) matches메소드의 사용

ReTest1.java

```
public class ReTest1{
	public static void main(String args[]){
		String msg1 = "hello world";
		String regex1 = "hello";
		String regex2 = "hello([a-z ]*)";
		System.out.println(msg1.matches(regex1));
		System.out.println(msg1.matches(regex2));

		String msg2 = "hello world3";
		String regex3 = "hello";
		String regex4 = "hello([a-z ]*)";
		System.out.println(msg2.matches(regex3));
		System.out.println(msg2.matches(regex4));

		String msg3 = "goooooogle";
		String regex5 = "goo*gle";
		String regex6 = "google";
```

```
                System.out.println(msg3.matches(regex5));
                System.out.println(msg3.matches(regex6));
        } // end main
} // end class
```

ReTest1.java를 컴파일하고 실행하면 그림1과 같은 결과가 나오게 됩니다.

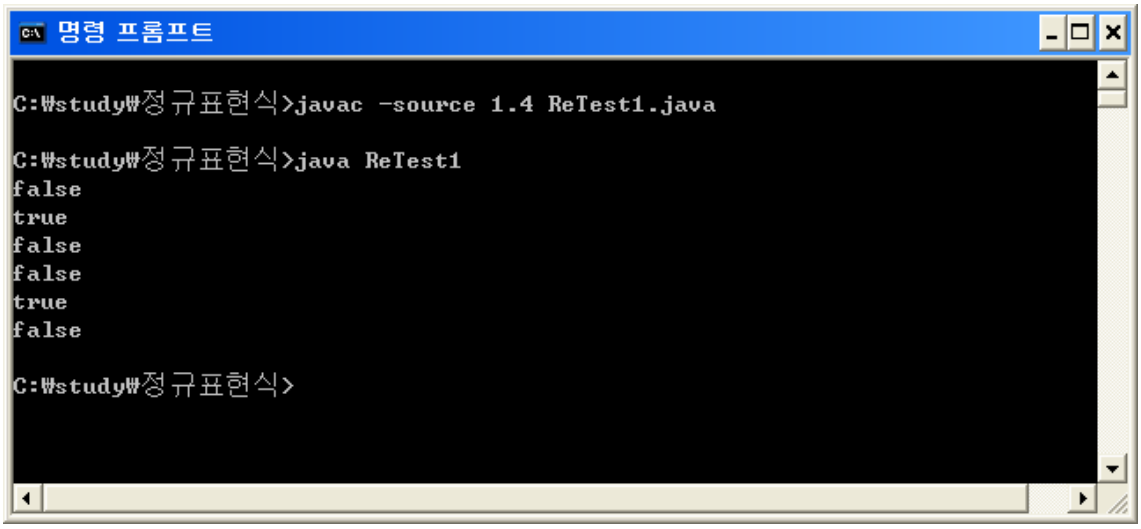

그림 1 ReTest1.java의 컴파일과 실

ReTest1.java에서 사용된 matches메소드는 J2SDK1.4.x이상에서만 사용할 수 있기 때문에 컴파일 시에는 반드시 "-source 1.4"옵션을 줘야 합니다. 소스를 한 줄 한 줄 설명하도록 하겠습니다.

```
String msg1 = "hello world";
String regex1 = "hello";
String regex2 = "hello([a-z ]*)";
System.out.println(msg1.matches(regex1));
System.out.println(msg1.matches(regex2));
```

msg1 String형 변수에 "hello world"문자열을 지정한 후, String class가 가지고 있는 matches메소드의 인자로 regex1과 regex2를 넣어서 결과를 확인 해보면, 첫 번째 것은 false가 두 번째 것은 true가 나옵니다. regex1의 패턴은 hello라는 정확한 패턴을 찾는 것이기 때문에 false이며 regex2의 패턴은 hello로 시작되며, 그 후의 문자열은 소문자와 공백들로 이루어져 있다라는 의미이기 때문입니다.

```
String msg2 = "hello world3";
String regex3 = "hello";
String regex4 = "hello([a-z ]*)";
System.out.println(msg2.matches(regex3));
System.out.println(msg2.matches(regex4));
```

msg2 String형 변수에 “hello world3”이라는 문자열을 지정한 후, matches메소드에 인자로 regex3와 regex4에 지정된 패턴을 인자로 넣어서 일치되는지 확인하고 있습니다. 이 경우, “hello”로 정의 된 패턴은 “hello”와 정확히 일치하여야 하기 때문에 false가 나오며, “hello([a-z]*)”로 정의된 패턴은 msg2안에 들어있는 문자열 3을 지정하지 않았기 때문에 false가 반환됩니다.

```
String msg3 = "gooooooogle";
String regex5 = "goo*gle";
String regex6 = "google";
System.out.println(msg3.matches(regex5));
System.out.println(msg3.matches(regex6));
```

msg3 String형 변수에 "gooooooogle"이라는 문자열을 지정한 후 패턴이 일치하는지 보고 있습니다. regex5에 지정된 패턴 “goo*gle”은 goo로 시작하고 gle로 끝나는 모든 문자열을 지칭하는 것이기 때문에 true가 반환되며, regex6에 지정된 패턴 “google”은 정확하게 “google”과 일치해야 함으로 false가 반환되게 됩니다.

예제2) replaceAll 메소드의 사용

ReTest2.java

```
public class ReTest2{
        public static void main(String args[]){
                String msg = "gooooooogle goooogle gooogle goooogle";
                String regex = "goo*gle";
                System.out.println(msg.replaceAll(regex, "google"));
        } // end main
} // end class
```

ReTest2.java를 컴파일 하고 실행하면 다음과 같은 결과가 나옵니다.

그림 2 ReTest2.java의 컴파일과 실행

ReTest2.java의 내용을 한 줄 한 줄 설명하도록 하겠습니다.

```
String msg = "gooooooogle goooogle gooogle goooogle";
String regex = "goo*gle";
System.out.println(msg.replaceAll(regex, "google"));
```

String형 변수 msg에 문자열 "gooooooogle goooogle gooogle goooogle"을 지정한 후, replaceAll메서드를 이용하여 regex패턴과 일치하는 내용을 “google”이라는 문자열로 치환한 후 결과를 출력하도록 합니다. 이때 regex패턴 “goo*gle”은 “goo”로 시작하고 “gle”로 끝나는 부분을 의미함으로 “gooooooogle” 문자열도 “google”로, “goooogle”문자열도 “google”로 “google”문자열도 “google”로 치환하게 됩니다.

예제3) replaceFirst 메소드의 사용

ReTest3.java

```
public class ReTest3{
        public static void main(String args[]){
                String msg = "gooooooogle goooogle gooogle goooogle";
                String regex = "goo*gle";
                System.out.println(msg.replaceFirst(regex, "google"));
        } // end main
} // end class
```

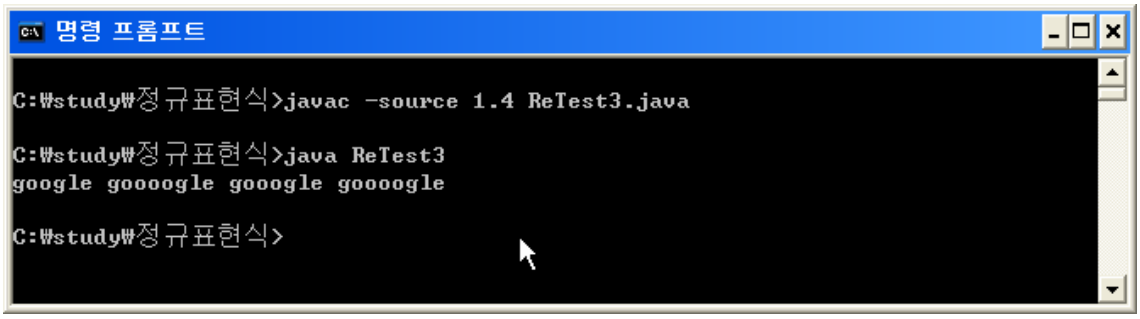

그림 3 ReTest3.java의 컴파일과 실행

replaceFirst메소드는 replaceAll 메소드의 사용방법과 흡사하지만, 패턴과 일치하는 첫 번째 문자열만 치환을 한다는 것입니다. 즉 소스를 분석하면 다음과 같습니다.

String형 변수 msg에 문자열 "gooooooogle goooogle gooogle goooogle"을 지정한 후, replaceFist메서드를 이용하여 regex패턴과 일치하는 첫번째 문자열을 “google”이라는 문자열로 치환한 후 결과를 출력하도록 합니다. 이때 regex패턴 “goo*gle”은 “goo”로 시작하고 “gle”로 끝나는 부분을 의미함으로 “gooooooogle” 문자열이 “google”로 치환하게 됩니다

예제4) split메소드의 사용

ReTest4.java

```
public class ReTest4{
        public static void main(String args[]){
                String msg = "one--two**three";
                String regex = "--|₩₩*₩₩*";
                String[] ss = msg.split(regex);
                for(int i = 0; i < ss.length; i++){
                        System.out.println(ss[i]);
                }
        } // end main
} // end class
```

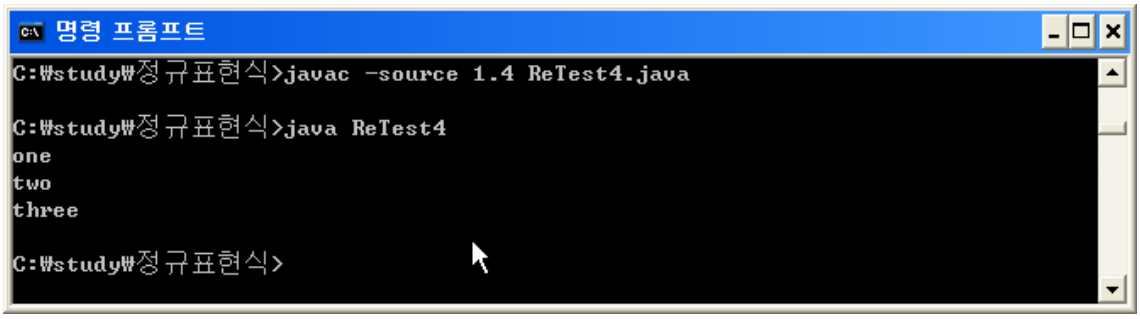

그림 4 ReTest4.java의 컴파일과 실행

ReTest4.java를 한 줄 한 줄 설명하도록 하겠습니다.

```
String msg = "one--two**three";
String regex = "--|₩₩*₩₩*";
String[] ss = msg.split(regex);
for(int i = 0; i < ss.length; i++){
        System.out.println(ss[i]);
}
```

msg String변수에 "one--two**three" 문자열을 지정한 후, regex String변수에 "--|₩₩*₩₩*" 정규표현식을 지정하고 있습니다. 해당 정규 표현식의 의미는 "--"와 일치하거나 "**"와 일치하는 문자열을 말합니다. 이때 "*"문자 앞에 "₩₩"이 사용된 의미는 "*" 문자가 정규표현식에서 이미 정의된 특수문자이기 때문입니다. 정규표현식에서 "*"문자를 특수문자가 아닌 일반 문자로 사용하기 위하여는 앞에 "₩(역 슬래시)"가 붙어야 하는데 자바 문법에서는 큰따옴표 안에서 "₩"문자의 표현은 "₩₩"로 해야 하기 때문에 "₩₩*"와 같이 사용되었습니다.

split메소드에 regex변수를 지정하여 문자열을 나눠 배열로 나뉘게 됩니다. 이 경우, msg문자열에는 "--"문자열과 "**" 문자열을 기준으로 쪼개지면서 ss배열에는 3개의 원소가 지정되게 됩니다. 3개의 원소는 "one", "two" 그리고 "three"값을 말합니다.

for 반복문을 이용하여 ss배열의 길이만큼 출력한 결과는 따라서 그림4)와 같이 나오게 되는 것입니다.

1-2) java.util.regex 패키지를 이용한 정규표현식의 처리

java.util.regex 패키지는 J2SDK1.4이후에 추가된 패키지입니다. java.util.regex패키지

는 Matcher, Pattern 2가지 클래스와 PatternSyntaxException 예외로 구성되어져 있습니다. 즉, java.util.regex 패키지를 이용한다는 것은 위의 3가지 클래스를 이용하는 것을 의미합니다.

사용 예를 하나 들어보도록 하겠습니다.

예제) java.util.regex 패키지의 이용

ReTest5.java

```
import java.util.regex.*;
public class ReTest5{
        public static void main(String[] args){
                String str = "hello++world--hi++world";
                String regex = "(₩₩+₩₩+)|(--)";

                Pattern p = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);
                Matcher m = p.matcher(str);
                while (m.find())
                {
                        System.out.println("Found '" + m.group(0) + "' at position " +
m.start(0));
                }
        }
}
```

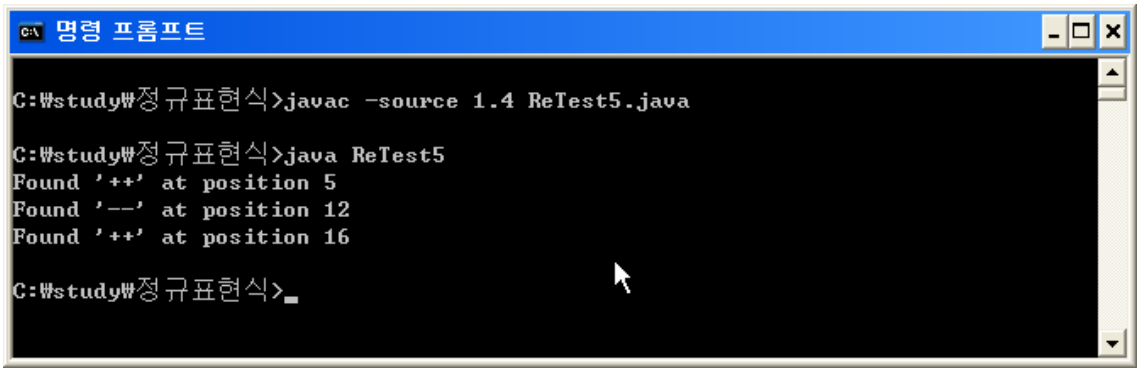

그림 5 ReTest5.java의 컴파일과 실행

ReTest5.java 소스를 보면, Pattern class의 static 메소드인 compile메소드에 regex문자열(패턴)을 인자로 넣어 Pattern객체를 할당 받는 것을 알 수 있습니다. 그 후에, Pattern class가 가지고 있는 static 메소드인 macher메소드의 인자로 str문자열을 집어넣어 Matcher 객체를 할당 받습니다. Matcher 객체는 complie메소드의 인자로 넣은 패턴과 macher메소드의 인자로 넣은 문자열과 일치하는 정보를 객체화 하고 있는 객체입니다.

Macher객체를 얻은 후에는 Macher class가 가지고 있는 메소드를 이용하여, str문자열에서 regex패턴과 일치하는 문자열의 위치 정보를 모두 출력하게 됩니다.

이때 일치하는 문자열이 없을 때까지 모두 뽑아내기 위하여는 find메소드를 이용하였으며, 정규패턴그룹 0번째에 일치하는 문자열을 발견하기 위하여는 group메소드를, 일치하는 위치 값을 출력하기 위하여는 start메소드가 이용되었습니다.

참고하세요.

정규표현식에서 중요한 것으로 정규패턴그룹이 여기에서 사용되었는데, 정규패턴 그룹을 간단하게 설명하면 아래와 같습니다.

- 정규 표현 그룹이란 왼쪽에서 우측으로 괄호를 세는 것으로써, 번호를 붙일 수 있게 됩니다. 예를 들어 정규 표현식 ((A)(B(C))) 가 있을 경우 4개의 그룹으로 분류됩니다. 0번 값은 패턴 전체를 나타나게 됩니다.

1 ((A)(B(C)))
2 (A)
3 (B(C))
4(C)

예제) 문자열에서 URL값만 뽑아내기

ReTest7.java

```
import java.util.regex.*;
public class ReTest7{
        public static void main(String[] args){
```

```
                String regex = "([₩Wp{Alnum}]+)://([a-z0-9.₩W-&%=?:@~₩W_]+)";
                //String regex = "([[a-zA-Z0-9]]+)://([a-z0-9.₩W-&%=?:@~₩W_]+)";
                String strHTML = "http://sunny.sarang.net <h1> http://www.yahoo.co.kr </h1> hohoho http://carami.sarang.net http://linux-sarang.net";

                Pattern p = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);
                Matcher m = p.matcher(strHTML);
                while (m.find())
                {
                        System.out.println(m.group(0));
                        System.out.println(m.group(1));
                        System.out.println(m.group(2));
                        System.out.println("----------------");
                }
        }
}
```

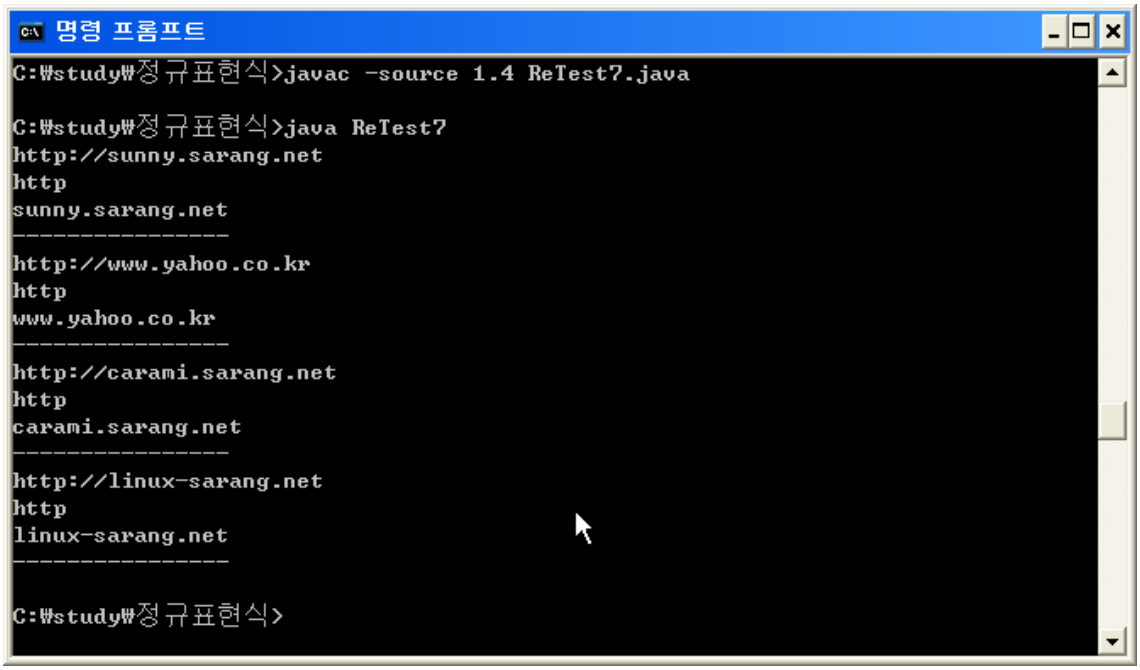

그림 6 ReTest7.java의 컴파일과 실행

ReTest7.java예제는 strHTML 문자열에 있는 내용 중 URL정보만을 뽑아서 출력하는 예제입니다. "([₩₩p{Alnum}]+)://([a-z0-9.₩₩-&%=?:@~₩₩_]+)" 와 같은 복잡한 패턴이 이용된 것을 알 수가 있습니다.

예제) URL을 자동으로 링크 걸기

ReTest8.java

```
import java.util.regex.*;
public class ReTest8{
        public static void main(String[] args){

                String regex = "([₩₩p{Alnum}]+)://([a-z0-9.₩₩-&%=?:@~₩₩_]+)";
                //String regex = "([[a-zA-Z0-9]]+)://([a-z0-9.₩₩-&%=?:@~₩₩_]+)";
                String strHTML = "http://sunny.sarang.net <h1> http://www.yahoo.co.kr </h1> hohoho http://carami.sarang.net http://linux-sarang.net";

                Pattern p = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);
                Matcher m = p.matcher(strHTML);
                String s = m.replaceAll("<a href='http://$2' target=_blank>http://$2</a>");
                System.out.println(s);
        }
}
```

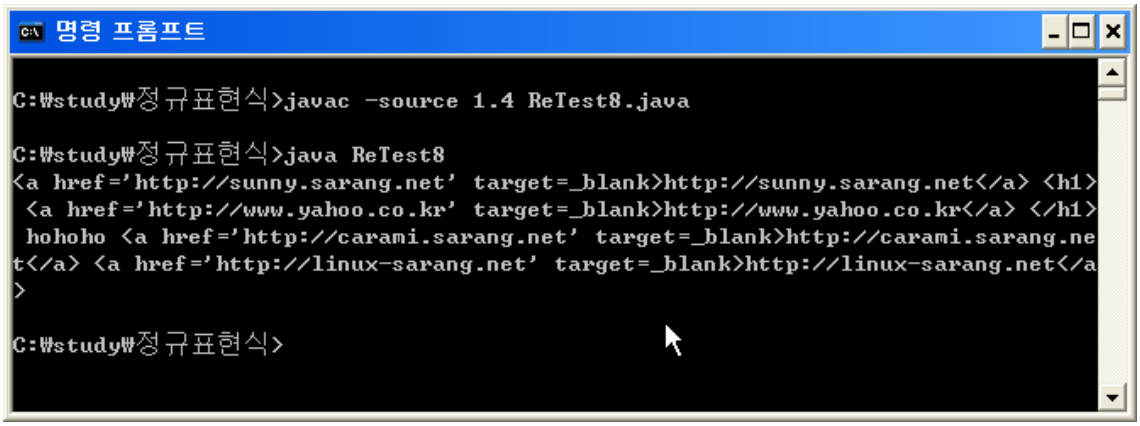

그림 7 ReTest8.java의 컴파일과 실행

ReTest8.java는 ReTest7.java를 약간 수정하였습니다. ReTest8.java 는 Matcher class가 가지고 있는 replaceAll메소드를 이용하여 자동으로 링크를 걸어주고 있습니다.

```
String s = m.replaceAll("<a href='http://$2' target=_blank>http://$2</a>");
```

위의 문장에서 $2 라고 나온 부분은 정규패턴 그룹의 2번째 부분을 말합니다. ReTest7.java에서 group(2)로 출력되는 결과를 말합니다.

마치며

정규표현식은 해당 문서에서 적은 예제 말고도, 다양한 예제를 가질 수가 있습니다. 예를 들자면, 웹 프로그래밍에서 사용자가 입력한 문자열 중에서 email 주소가 있을 경우 자동으로 링크를 걸어줄 수도 있으며, 사용자로부터 입력 받은 문자열을 검사하는 데에서도 이용될 수 있습니다.

정규표현식을 이용하게 되면, 문자열을 처리하는데 있어서 여러 줄로 처리해야 할 것을 간단하게 처리할 수 있다는 장점이 있습니다. 다만, 정규표현식이라는 것을 학습해야 하지만 말입니다.

게으른 프로그래머라고 들어보았을 것입니다. 한번 공부하기는 어렵고 번거롭지만, 한번 공부해두면 두고두고 유용하게 써먹을 수 있는 것이 정규표현식에 대한 용법입니다.

java.util.regex 패턴에 대하여 좀더 자세히 쓰고 싶었는데요. 나머지는 숙제로 맡겨도 괜찮겠다라는 생각이 들어서 중지했습니다. 아마도, 저의 이름으로 책을 쓰게 될 경우에는, 해당 문서보다 더욱 자세한 내용을 써야겠다라는 소망을 끝으로 글을 마칩니다. 읽어주셔서 감사합니다.

2003년 4월 1일 만우절 날 김성박.