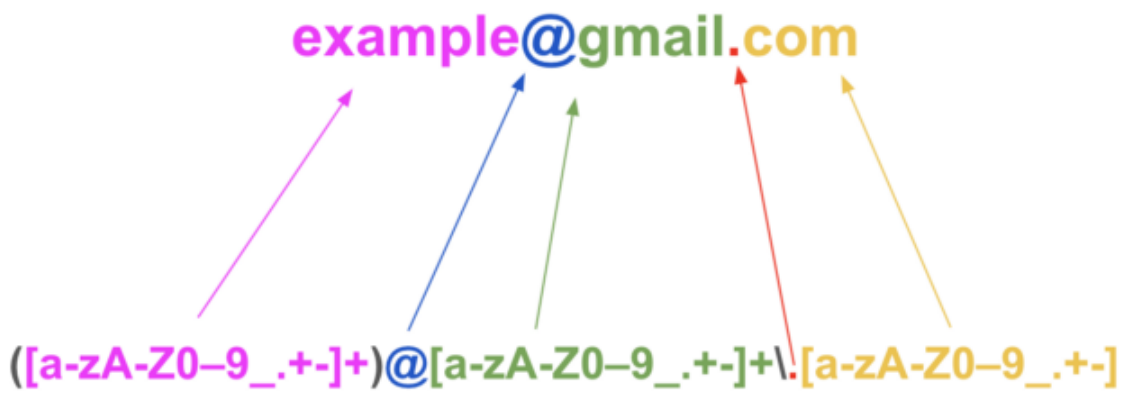
# 정규표현식 이란

정규표현식(Regular Expression)이란 문자열 데이터 중에서 원하는 조건(패턴)과 일치하는 문자열 부분을 찾아내기 위해 사용하는 것으로, 미리 정의된 기호와 문자를 이용해서 작성한 문자열을 말한다.

개발을 하다보면 전화번호, 주민등록번호, 이메일등과 같이 정해져있는 형식이 있고 사용자가 그 형식대로 제대로 입력했는지 검증을 해야하는 경우가 종종 있다. 이런 입력값을 정해진 형식에 맞는지 검증해야 할 때에는 정규표현식을 사용하면 쉽게 구현할 수 있다.




이처럼 정규식을 이용하면 많은 양의 텍스트 파일 중에서 원하는 데이터를 손쉽게 뽑아낼 수 있고, 입력된 데이터가 형식에 맞는지 체크할 수도 있다. 하지만 위의 사진을 보듯이 복잡한 문자 기호 조합으로 가독성이 떨어진다는 단점도 있다. (난이도가 있다)

자바 외에 대부분의 언어들도 정규표현식을 지원한다. 다만 그 사용법들이 언어마다 약간 차이가 있지만 90% 이상은 동일한 문법을 차용하기 때문에 한번 익혀두면 다른 언어의 정규식을 이용하는데 있어 수월해 질 수 있다.

---

# 정규식 문법 기호 모음

## 정규식 기본 기호

| 기호 | 설명 | 예제 |
|---|---|---|
| . | 임의의 문자 1개를 의미 | |
| ^ | 시작을 의미한다<br>[] 괄호 안에 있다면 일치하지 않는 부정의 의미로로 쓰인다 | ^a : a로 시작하는 단어<br>[^a] : a가 아닌 철자인 문자 1개 |
| $ | $앞의 문자열로 문자가 끝나는지를 의미한다 | a$ : a로 끝나는 단어 |
| [] | [] 괄호 안의 문자가 있는지를 확인한다 | [ab][cd] : a,b중 한 문자와 c,d중 한 문자<br>→ ac ad bc bd |
| [^] | [] 대괄호 안에 ^ 문자가 있으면, 제외를 뜻함<br>- 대괄호 안에 ^ 가 쓰이면 제외의 뜻<br>- 대괄호 밖에 ^ 가 쓰이면 시작점의 뜻 | [^a-z] : 알파벳 소문자 a부터 z까지를 제외한 모든 문자 |
| - | 사이의 문자 혹은 숫자를 의미한다 | [a-z] : 알파벳 소문자 a부터 z까지 하나<br>[a-z0-9] : 알파벳 소문자 전체,0~9 중 한 문자 |
| \| | 또는 | [a\|b] : a 혹은 b |
| () | 그룹 | 01(0\|1) : 01뒤에 0 또는 1이 들어간다<br>→ 010(o), 011(o), 012(x) |
| {} | 개수 | a{3}b : a가 3번 온 후 b가 온다<br>→ aab(x), aaab(o), aaaab(o) |
| ₩b | 공백, 탭, ",", "/" 등을 의미한다 | apple₩b : apple뒤에 공백 탭등이 있다<br>→ apple juice (o), apple.com (x) |
| ₩B | ₩b의 부정<br>공백 탭등이 아닌 문자인 경우 매치한다 | apple₩B<br>→ apple.com (o) |
| ₩d | 0~9 사이의 숫자<br>[0-9]와 동일 | |
| ₩D | ₩d의 부정<br>숫자가 아닌 어떤 문자, [^0-9]와 동일 | |
| ₩s | 공백, 탭 | |
| ₩S | 공백, 탭이 아닌 문자 | |
| ₩w | 알파벳 대소문자+숫자+"_"<br>[a-zA-Z_0-9]와 동일 | |

| | |
|---|---|
| ₩W | ₩w의 부정<br>[^a-zA-Z_0-9] |

## 정규식 수량 기호

| 기호 | 설명 | 예제 |
|---|---|---|
| ? | 앞의 표현식이 없거나 or 최대 한개만 | a1? : 1이 최대 한개만 있거나 없을수도 있다<br>→ a(o), a1(o), a11(x), a111(x) |
| * | 앞의 표현식이 없거나 or 있거나 (여러개) | a1* : 1이 있을수도 없을수도 있다<br>→ a(o), a1(o), a11(o), a111(o) |
| + | 앞의 표현식이 1개 이상 or 여러개 | a1* : 1이 1개 이상있다<br>→ a(x), a1(o), a11(o), a111(o) |
| {n} | 딱 n개 있다 | a{3} : a가 딱 3개 있다<br>→ aa(x), aaa(o), aaaa(x), aaaaaaa(x) |
| {n, m} | n개 이상 m개 이하 | a{3,5} : a가 3개 이상 5개 이하 있다<br>→ aa(x), aaa(o), aaaa(o), aaaaaaa(x) |
| {n,} | n개 이상 | a{3,} : a가 3개 이상 있다<br>→ aa(x), aaa(o), aaaa(o), aaaaaaa(o) |

## 정규식 그룹 캡쳐 기호

| 기호 | 설명 |
|---|---|
| () | 그룹 및 캡쳐 |
| (?:) | 찾지만 그룹에 포함 안됨 |
| (?=) | =앞 문자를 기준으로 전방 탐색 |
| (?<=) | =뒤 문자를 기준으로 후방 탐색 |

## 자주 사용되는 정규식 샘플

| 정규 표현식 | 설명 |
|---|---|
| ^[0-9]*$ | 숫자 |
| ^[a-zA-Z]*$ | 영문자 |
| ^[가-힣]*$ | 한글 |
| ₩₩w+@₩₩w+₩₩.₩₩w+(₩₩.₩₩w+)? | E-Mail |
| ^₩d{2,3}-₩d{3,4}-₩d{4}$ | 전화번호 |
| ^01(?:0\|1\|[6-9])-(?:₩d{3}\|₩d{4})-₩d{4}$ | 휴대전화번호 |
| ₩d{6} ₩- [1-4]₩d{6} | 주민등록번호 |
| ^₩d{3}-₩d{2}$ | 우편번호 |

# 자바 정규식 문법

## String 클래스의 정규식 문법

String 문자열에 바로 정규표현식을 적용하여 필터링이 가능하다.

String 클래스에서 지원하는 정규식 메소드로는 다음 3가지가 존재한다.

| String 정규식 메서드 | 설명 |
|---|---|
| boolean matches(String regex) | 인자로 주어진 정규식에 매칭되는 값이 있는지 확인 |
| String replaceAll(String regex, String replacement) | 문자열내에 있는 정규식 regex와 매치되는 모든 문자열을 replacement 문자열로 바꾼 문자열을 반환 |

| String[] split(String regex) | 인자로 주어진 정규식과 매치되는 문자열을 구분자로 분할 |
|---|---|

JAVA

```
// matches (일치하는지 확인)
String txt = "123456";
boolean result1 = txt.matches("[0-9]+"); // 숫자로 이루어져 있는지
System.out.println(result1); // true
```

JAVA

```
// replaceAll (정규표현식과 일치하는 모든 값 치환)
String txt2 = "power987*-;";
String result2 = txt2.replaceAll("[^a-z0-9]","0"); // 영문자와 숫자를 제외한 문자를 모두 0으로 치환
System.out.println(result2); // power987000
```

JAVA

```
// split (정규표현식과 일치하는 값 기준으로 나누기)
String txt3 = "power987*-;";
String[] txts = txt3.split("[0-9]+"); // 숫자부분을 기준으로 분할
System.out.println(txts[0]); // power
System.out.println(txts[1]); // *-;
```

---

# regex 패키지 클래스

자바에서 정규 표현식을 전문적으로 다루는 클래스인 java.util.regex 패키지를 제공해준다. 패키지 안의 클래스중 주로 **Pattern 클래스**와 **Matcher 클래스**가 사용된다.

이들 정규식 클래스의 장점으로는 정규식을 Pattern 객체로 미리 컴파일 해둘수 있어서 처리 속도가 좀 더 빠르고, 매칭된 데이터를 좀더 상세히 다룰 수 있다.

## Pattern 클래스

클래스명에서 볼수 있다싶이, 이 클래스의 주요 역할은 문자열을 정규표현식 패턴 객체로 변환해주는 역할을 한다. 물론 이때 문자열을 정규식 문법에 알맞게 구성해 주어야 한다. 그렇지않으면 예외(Exception)이 발생하게 된다.

Pattern 클래스는 일반 클래스처럼 공개된 생성자를 제공하지 않는다. 그래서 정규식 패턴 객체를 생성하려면 `compile()` 정적 메소드를 호출해야 한다. 이렇게 Pattern 객체로 컴파일된 정규식은 뒤의 Matcher 클래스에서 사용된다

JAVA

```
// 문자열 형태의 정규표현식 문법을 정규식 패턴으로 변환
String patternString = "^[0-9]*$";
Pattern pattern = Pattern.compile(patternString); // Pattern 객체로 컴파일된 정규식은 뒤의 Matcher 클래스에서 사용된다
```

혹은 바로 `matches()` 메소드를 활용하여 정규식 검증을 할 수도 있다.

`matches()` 메서드의 첫번째 입력값은 **정규식 문자열**이고, 두번째 입력값은 **검증 대상 문자열** 이다. 검증 후 대상문자열이 정규표현식과 일치하면 true, 그렇지 않다면 false값을 리턴한다.

JAVA

```
// 샘플 문자열
String txt1 = "123123";
String txt2 = "123이것은숫자입니다00";
boolean result = Pattern.matches("^[0-9]*$", txt1); // 첫번째 매개값은 정규표현식이고 두번째 매개값은 검증 대상 문자열
System.out.println(result); // true
boolean result2 = Pattern.matches("^[0-9]*$", txt2);
System.out.println(result2); // false
```

| Pattern 클래스 메서드 | 설명 |
|---|---|
| compile(String regex) | 정규표현식의 패턴을 작성 |
| matches(String regex, CharSequence input) | 정규표현식의 패턴과 문자열이 일치하는지 체크<br>일치할 경우 true, 일치하지 않는 경우 false를 리턴 |

| | |
|---|---|
| | (일부 문자열이 아닌 전체 문자열과 완벽히 일치 해야한다) |
| asPredicate() | 문자열을 일치시키는 데 사용할 수있는 술어를 작성 |
| pattern() | 컴파일된 정규표현식을 String 형태로 반환 |
| split(CharSequence input) | 문자열을 주어진 인자값 CharSequence 패턴에 따라 분리 |

## Matcher 클래스

Matcher 클래스는 대상 문자열의 패턴을 해석하고 주어진 패턴과 일치하는지 판별하고 반환된 필터링된 결과값들을 지니고 있다. Matcher 클래스 역시 Pattern 클래스와 마찬가지로 공개된 생성자가 없다. Matcher객체는 Pattern 객체의 `matcher()` 메소드를 호출해서 얻는다.

이처럼 자바(Java) 언어는 다른 언어와는 달리 정규표현식 생성 문법이 조금 복잡한 편이다. 자바의 정규식 결과 반환 로직 순서를 정리하자면 다음과 같다.

`Pattern.compile()` 을 통해 정규식문자열을 패턴 객체로 변환

패턴 객체에서 `matcher()` 메소드를 통해 문자열을 비교하고 검사한 결과값을 담은 매처 객체를 반환

매처 객체에서 메소드로 원하는 결과값을 뽑음

`Pattern.matches()` 메소드는 단순히 참/거짓 만 결과를 반환하지만 Matcher 클래스의 `group()` 메소드를 통해 필터링된 문자열을 출력할수 있다.

JAVA

```java
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class Main {
    public static void main(String[] args) {
        // 비교할 문자열
        String txt = "1487안녕";
        // 문자열 형태의 정규표현식 문법
        String patternString = "^[0-9]+";
        // 1) 문자열 형태의 정규표현식 문법을 정규식 패턴으로 변환
        Pattern pattern = Pattern.compile(patternString);

        // 2) 패턴 객체로 matcher 메서드를 통해 문자열을 검사하고 필터링된 결과를 매처 객체로 반환
        Matcher matcher = pattern.matcher(txt);
        // 3) 정규식 필터링된 결과를 담은 matcher에서 메소드를 통해 결과를 출력
        System.out.println(matcher.find()); // 매칭된 결과가 있는지? : true
        System.out.println(matcher.group()); // 매칭된 부분을 반환 : 1487
    }
}
```

**Tip**

Matcher 클래스의 입력값으로는 CharSequence라는 인터페이스 타입을 받는데, 이를 통해 단순히 문자열 뿐만 아니라 다형성 기능을 통해 다양한 형태의 입력 데이터로부터 문자 단위의 매칭 기능을 지원 받을 수 있다.

```
java.util.regex.Pattern

public Matcher matcher(CharSequence input)
```

대표적으로 String 클래스의 implements 구성을 보면 CharSequence 인터페이스를 상속 받은걸 볼 수 있다.

compact1, compact2, compact3
java.lang
**Class String**

java.lang.Object
    java.lang.String

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented as instances of this class.
Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

| Matcher 클래스 메서드 | 설 명 |
|---|---|
| find( ) | 패턴이 일치하는 경우 true를 반환, 불일치하는 경우 false반환<br>(여러개가 매칭되는 경우 반복실행하면 일치하는 부분 다음부터 이어서 매칭됨) |
| find(int start) | start 위치 이후부터 매칭검색 |
| start( ) | 매칭되는 문자열의 시작위치 반환 |
| start(int group) | 지정된 그룹이 매칭되는 시작위치 반환 |
| end( ) | 매칭되는 문자열 끝위치의 다음 문자위치 반환 |
| end(int group) | 지정된 그룹이 매칭되는 끝위치의 다음 문자위치 반환 |
| group( ) | 매칭된 부분을 반환 |
| group(int group) | 그룹화되어 매칭된 패턴중  group 번째 부분 반환 |
| groupCount( ) | 괄호로 지정해서 그룹핑한 패턴의 전체 개수 반환 |
| matches( ) | 패턴이 전체 문자열과 일치할 경우 true반환<br>(일부 문자열이 아닌 전체 문자열과 완벽히 일치 해야한다) |

### 1. 다중 결과값 출력하기

만일 필터링된 결과값이 여러개일 경우 다음과 같이 반복문을 통해 순회하여 출력해야 한다. (약간 번거롭다)



```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class Main {
    public static void main(String[] args) {
        String something = "hello987*-;hi66"; // 비교할 문자열
        Pattern pattern = Pattern.compile("[a-z]+[0-9]+"); // 정규표현식 문자열로 패턴 객체 생성
        Matcher matcher = pattern.matcher(something); // 패턴 객체로 문자열을 필터링한뒤 그 결과값들을 담은 매처 객체 생성
        while (matcher.find()) {
            System.out.println(matcher.group());
            // 루프 1번 : hello987
            // 루프 2번 : hi66
        }
    }
}
```

### 2. 그룹핑 결과 출력하기

그룹핑을 적용한 경규표현식일 경우 조금 복잡하다.

패턴을 그룹 괄호 ( ) 로 이용해서 세 부분으로 묶으면, 각 그룹마다 매칭 되게 된다.
따라서  group(1) ,  group(3) ,  group(3) …으로 호출할 수 있다.
group() 이나  group(0) 은 그룹으로 매칭 된 문자열 전체를 반환한다.

JAVA

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class Main {
    public static void main(String[] args) {
        String source = "011-4632-1290, 02-889-7661";
        String pattern = "(0WWd{1,2})-(WWd{3,4})-(WWd{4})";
        Matcher matcher = Pattern.compile(pattern).matcher(source); // 한방에 매처 객체 반환
        System.out.println("그룹의 개수 : " + matcher.groupCount());    //그룹화된 개수가 몇개인지 출력
        int i = 0;
        while (matcher.find()) {
            System.out.println(++i + ": " + matcher.group() + " -> " + matcher.group(1) + " 와 " + matcher.group(2) + " 와 " + matcher.group(3
        }
    }
}
```

```
"C:\Program Files\Java\jdk1.8.0_202\bin\java.exe" ...
그룹의 개수 : 3
1: 011-4632-1290 -> 011 와 4632 와 1290
2: 02-889-7661 -> 02 와 889 와 7661

종료 코드 0(으)로 완료된 프로세스
```

### 3. 매칭 위치 출력하기

start() 와 end() 메서드로 매칭되는 문자열 부분의 위치를 알아낼 수 있다.

JAVA

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class Main {
    public static void main(String[] args) {
        String source = "동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라 만세";
        String pattern = "백두산";
        Matcher matcher = Pattern.compile(pattern).matcher(source);
        while (matcher.find()) {
            // start()와 end()로 일치하는 부분의 위치를 알아낼 수 있다.
            System.out.println("매칭되는 문자열 위치 : " + matcher.start() + " ~ " + matcher.end());
        }
    }
}
```

```
"C:\Program Files\Java\jdk1.8.0_202\bin\java.exe" ...
매칭되는 문자열 위치 : 5 ~ 8

종료 코드 0(으)로 완료된 프로세스
```

## 정규식 플래그

정규식 플래그는 정규식을 생성할 때 **고급 검색**을 위한 전역 옵션을 설정할 수 있도록 지원하는 기능이다. 패턴이 매치되는 방법에 전박적인 영향을 미치는 기능을 담고 있다.

정규표현식에 패턴을 주는 방법은 기호를 조합하는 방법과 패턴 상수를 주는 방법이 있다.

JAVA

```
// 정규식 기호로 전달
Pattern.compile("(?i)([a-z]+)")
// 상수로 전달
Pattern.compile("(^.*$)", Pattern.CASE_INSENSITIVE)
```

| Pattern 플래그 상수 | 기호 | 설명 |
| --- | --- | --- |
| Pattern.CANON_EQ | None | 표준화된 매칭 모드를 활성화합니다.<br>이 모드가 켜지면 a를 나타내는 유니코드 "₩u00E5"와 a와 상단고리 유니코드를 쓴 "a₩u030A"를 같다고 매칭합니다. |

| | | |
|---|---|---|
| Pattern.CASE_INSENSITIVE | (?i) | 대소문자를 구분하지 않습니다. |
| Pattern.COMMENTS | (?x) | 공백과 주석이 무시됩니다. 주석은 #부터 그 행 끝까지 입니다. |
| Pattern.MULTILINE | (?m) | 다중행 모드를 사용여 모든 ^와 $가 인식됩니다.<br>기본값은 입력값 전체를 하나의 시작과 끝으로 인식합니다. |
| Pattern.DOTALL | (?s) | .가 개행문자 까지 포함하는 모든 문자로 매칭됩니다. |
| Pattern.LITERAL | None | 입력의 메타문자와 이스케이프된 문자를 일반 문자로 취급합니다.<br>CASE_INSENSITIVE와 UNICODE_CASE는 기능이 유지됩니다. |
| Pattern.UNICODE_CASE | (?u) | 이 모드가 활성화 되면 대소문자 매칭이 유니코드 표준을 따릅니다.<br>기본은 US-ASCII 문자 집합을 따릅니다. |
| Pattern.UNIX_LINES | (?d) | ^와 $를 처리시 UNIX 개행을 사용합니다. |

JAVA

```java
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class Main {
    public static void main(String[] args) {
        // 개행 문자가 들어간 비교할 문자열
        String txt = "this₩nis₩ntest₩n";
        // . 기호는 모든 문자를 포함하지만 개행 문자는 포함하지 않기 때문에 일치하는 검색 결과가 없음
        Matcher matcher1 = Pattern.compile("(^.*$)").matcher(txt);
        System.out.println(matcher1.find()); // false
        // . 기호가 개행 문자까지 포함하도록
        Matcher matcher2 = Pattern.compile("(?s)(^.*$)").matcher(txt);
        while (matcher2.find()) {
            System.out.println(matcher2.group()); // this, is, test
        }
        // 다중 행 모드로 각 행마다 정규식 검사
        Matcher matcher3 = Pattern.compile("(?m)(^.*$)").matcher(txt);
        while (matcher3.find()) {
            System.out.println(matcher3.group()); // this, is, test
        }
        // 대소문자 구분 X
        Matcher matcher4 = Pattern.compile("(?i)([a-z]+)").matcher("HelloWorld"); // 정규식 기호로 전달
        Matcher matcher5 = Pattern.compile("(^.*$)", Pattern.CASE_INSENSITIVE).matcher("HelloWorld"); // 상수로 전달
        while (matcher5.find()) {
            System.out.println(matcher5.group()); // HelloWorld
        }
    }
}
```

## PatternSyntaxException 예외

정규식 패턴의 문법 오류를 나타내는 unchecked Exception 종류 이다.

JAVA

```java
import java.util.regex.PatternSyntaxException;
public class Main {
    public static void main(String[] args) {
        try {
            String txt = "power987*-;";
            String[] splitStr = txt.split("+"); // 오류 문법
        } catch (PatternSyntaxException e) {
            e.printStackTrace();
        }
    }
}
```

```
"C:\Program Files\Java\jdk1.8.0_202\bin\java.exe" ...
java.util.regex.PatternSyntaxException: Dangling meta character '+' near index 0
+
^
    at java.util.regex.Pattern.error(Pattern.java:1957)
    at java.util.regex.Pattern.sequence(Pattern.java:2125)
    at java.util.regex.Pattern.expr(Pattern.java:1998)
    at java.util.regex.Pattern.compile(Pattern.java:1698)
    at java.util.regex.Pattern.<init>(Pattern.java:1351)
    at java.util.regex.Pattern.compile(Pattern.java:1028)
    at java.lang.String.split(String.java:2380)
    at java.lang.String.split(String.java:2422)
    at Main.main(Main.java:11)
```

위의 오류가 발생하는 이유는 +, [, ], (, ) 등의 특수문자를 사용할 때는 앞에 ₩₩를 붙여주어야 하기 때문이다. (이스케이프 처리)

JAVA

```java
String txt = "power987*-;";
String[] splitStr = txt.split("₩₩+"); // 특문 이스케이프 처리
```

# 자바 정규식 테스트 사이트

**RegexPlanet: online regular expression tes…**

www.regexplanet.com

RegexPlanet: online regular expression testing for Java

# 정규식 사이트 사용법

정규식 온라인 사이트 사용법은, 1번란에 정규식을 입력하고, 2번란에 정규식을 테스트할 문자열을 입력한 뒤 Test 버튼을 클릭하면된다. 위에서 배웠던 자바의 Matcher 클래스 메소드들 결과값들이 표로 출력되어 나온다.

## Expression to test

**Regular expression:** ([a-z]+)([0-9]+) 정규표현식

**Options:**

- [ ] Force canonical equivalence (CANON_EQ)
- [ ] Case insensitive (CASE_INSENSITIVE)
- [ ] Allow comments in regex (COMMENTS)
- [ ] Dot matches line terminator (DOTALL)
- [ ] Treat as a sequence of literal characters (LITERAL)
- [ ] ^ and $ match EOL (MULTILINE)
- [ ] Unicode case matching (UNICODE_CASE)
- [ ] Only consider '\n' as line terminator (UNIX_LINES)

**Replacement:**

Test | More Inputs

**Input 1:** hello987*-;hi66 문자열 샘플

**Input 2:** 가나다라마바@555

**Input 3:** abds123456789

**Input 4:**

**Input 5:**

Test | More Inputs

# Test Results

| | |
|---|---|
| Regular Expression | ([a-z]+)([0-9]+) |
| as a Java string | "([a-z]+)([0-9]+)" |
| Replacement | |
| groupCount() | 2 |

Matcher 클래스의 메소드들

| Test | Target String | matches() | replaceFirst() | replaceAll() | lookingAt() | find() | group(0) | group(1) | group(2) |
|---|---|---|---|---|---|---|---|---|---|
| 1 | hello987*-;hi66 | No | *-;hi66 | *-; | Yes | Yes | hello987 | hello | 987 |
| | | | | | next find() | Yes | hi66 | hi | 66 |
| 2 | 가나다라마바@555 | No | 가나다라마바@555 | 가나다라마바@555 | No | No | | | |
| 3 | abds123456789 | Yes | | | Yes | Yes | abds123456789 | abds | 123456789 |