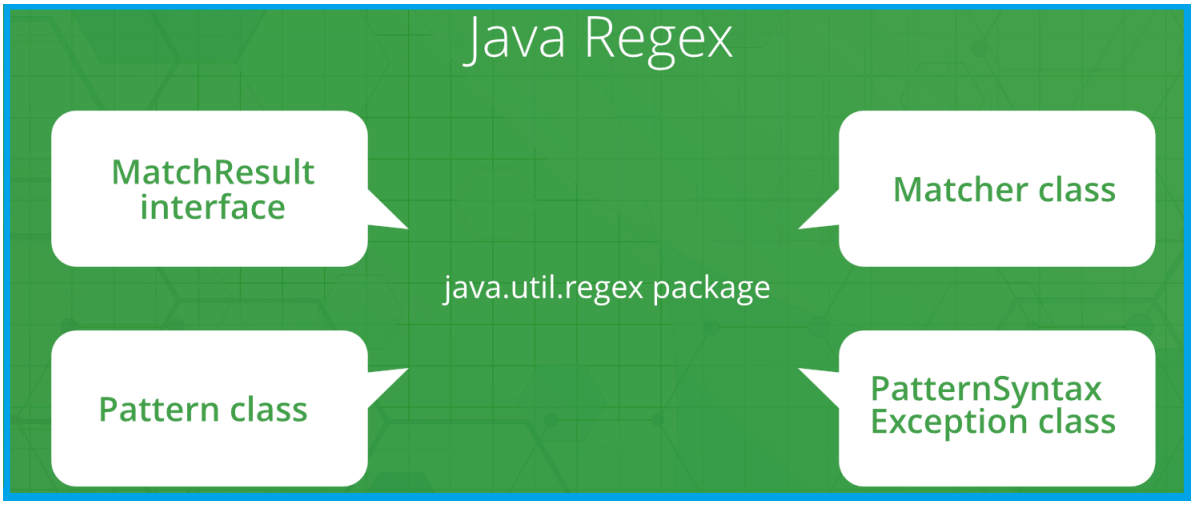
Regular Expressions(정규 표현식)

Regular Expressions 또는 줄여서 Regex(Regexp)는 Java에서 문자열을 검색, 조작 및 편집하는 데 사용할 수 있는 문자열 패턴을 정의하기 위한 API입니다. 이메일 유효성 검사 및 암호는 Regex가 제약 조건을 정의하는 데 널리 사용되는 문자열의 일부 영역입니다. 정규식은 java.util.regex 패키지에서 제공됩니다. 이것은 3개의 클래스와 1개의 인터페이스로 구성됩니다. **java.util.regex** 패키지는 주로 다음과 같은 표형식으로 아래에 설명된 세 가지 클래스로 구성됩니다.

Class	Description
util.regex.Pattern	패턴 정의에 사용
util.regex.Matcher	패턴을 사용하여 텍스트에 대한 일치 작업을 수행하는 데 사용됩니다.
PatternSyntaxException	정규식 패턴에서 구문 오류를 나타내는 데 사용됩니다.

Java의 Regex는 다음과 같이 아래에 나열된 두 가지 클래스를 제공합니다.

1. Pattern Class
2. Matcher Class



1. Pattern Class

이 클래스는 공개 생성자를 제공하지 않고 다양한 유형의 패턴을 정의하는 데 사용할 수 있는 정규식의 컴파일입니다. 정규식을 첫 번째 인수로 받아들이고 실행 후 패턴을 반환하는 compile() 메서드를 호출하여 만들 수 있습니다.

Method	Description
compile(String regex)	주어진 정규식을 패턴으로 컴파일하는 데 사용됩니다.
compile(String regex, int flags)	주어진 플래그를 사용하여 주어진 정규식을 패턴으로 컴파일하는 데 사용됩니다.
flags()	이 패턴의 일치 플래그를 반환하는 데 사용됩니다.
matcher(CharSequence input)	이 패턴에 대해 주어진 입력을 일치시킬 매처를 만드는 데 사용됩니다.
matches(String regex, CharSequence input)	주어진 정규식을 컴파일하고 주어진 입력을 이에 대해 일치시키려고 시도하는 데 사용됩니다.
pattern()	이 패턴이 컴파일된 정규식을 반환하는 데 사용됩니다.
quote(String s)	지정된 문자열에 대한 리터럴 패턴 문자열을 반환하는 데 사용됩니다.
split(CharSequence input)	주어진 입력 시퀀스를 이 패턴의 일치 항목으로 분할하는 데 사용됩니다.
split(CharSequence input, int limit)	주어진 입력 시퀀스를 이 패턴의 일치 항목으로 분할하는 데 사용됩니다. 한계 매개변수는 패턴이 적용되는 횟수를 제어합니다.
toString()	이 패턴의 문자열 표현을 반환하는 데 사용됩니다.

1-1. pattern()/toString() 메서드

```
import java.util.regex.Pattern;

public class RegExpEx01_Pattern1 {
    public static void main(String[] args) {
        // "^[0-9]*$" : 숫자만
        Pattern pattern = Pattern.compile("^[0-9]*$");
        System.out.println(pattern);
        // String java.util.regex.Pattern.toString()
        System.out.println(pattern.toString());
        // String java.util.regex.Pattern.pattern()
        System.out.println(pattern.pattern());
        System.out.println();
    }
}
```

결과

```
^[0-9]*$
^[0-9]*$
^[0-9]*$
```

1-2. matcher(CharSequence input)/matches(String regex, CharSequence input) 메서드

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegExpEx02_Pattern2 {
    public static void main(String[] args) {
        System.out.println("\".한\" : 두 글자이면서 두번째 글자가 \"한\"인지를 검사한다.\n");
        // Pattern 클래스 사용법 1
        System.out.println("1. Pattern.matches(String regex, CharSequence input)");
        // boolean java.util.regex.Pattern.matches(String regex, CharSequence input)
        for(String word : new String[] {"대한","소한","소백","대한민국"}) {
            System.out.println(word + " : " + Pattern.matches(".한", word));
        }
        System.out.println();

        // Pattern 클래스 사용법 2
        System.out.println("2. pattern.matches(CharSequence input)");
        Pattern pattern = Pattern.compile(".한");
        for(String word : new String[] {"대한","소한","소백","대한민국"}) {
            // Matcher java.util.regex.Pattern.matcher(CharSequence input)
            Matcher matcher = pattern.matcher(word);
            // boolean java.util.regex.Matcher.matches()
            System.out.println(word + " : " + matcher.matches());
        }
        System.out.println();

        // Pattern 클래스 사용법 3
        System.out.println("3. Pattern.compile(String regex).matcher(CharSequence input).matches()");
        for(String word : new String[] {"대한","소한","소백","대한민국"}) {
            System.out.println(word + " : " + Pattern.compile(".한").matcher(word).matches());
        }
        System.out.println();
    }
}
```

결과

```
".한" : 두 글자이면서 두번째 글자가 "한"인지를 검사한다.

1. Pattern.matches(String regex, CharSequence input)
대한 : true
소한 : true
소백 : false
대한민국 : false

2. pattern.matches(CharSequence input)
대한 : true
소한 : true
소백 : false
```

```
대한민국 : false

3. Pattern.compile(String regex).matcher(CharSequence input).matches()
대한 : true
소한 : true
소백 : false
대한민국 : false
```

1-3. asPredicate() 메서드

Predicate는 Type T 인자를 받고 boolean을 리턴하는 함수형 인터페이스입니다.

Predicate 객체의 test() 메서드에 인자를 전달하면 boolean이 리턴됩니다.

```
import java.util.function.Predicate;
import java.util.regex.Pattern;

public class RegExpEx03_Pattern3 {
    public static void main(String[] args) {
        Pattern pattern = Pattern.compile("한");
        // Predicate<String> java.util.regex.Pattern.asPredicate()
        // Predicate는 Type T 인자를 받고 boolean을 리턴하는 함수형 인터페이스입니다.
        Predicate<String> predicate = pattern.asPredicate();
        for(String word : new String[] {"한","대한","대한민국","우리나라"}) {
            // test()에 인자를 전달하면 boolean이 리턴됩니다.
            // boolean java.util.function.Predicate.test(String t)
            System.out.println(word + " : " + predicate.test(word));
        }
        System.out.println();
    }
}
```

결과

```
한 : true
대한 : true
대한민국 : true
우리나라 : false
```

1-4. split(CharSequence input)/split(CharSequence input, int limit) 메서드

양수일 때에는 limit 번째부터 split을 하지 않으며, 음수일 때에는 공백 또한 데이터에 포함합니다.

```
import java.util.Arrays;
import java.util.regex.Pattern;

public class RegExpEx04_Pattern4 {
    public static void main(String[] args) {
        // \s : 공백 문자
        String inputStr = "자바 정규표현식 :\tPattern클래스\nsplit메서드 연습";
        Pattern pattern1 = Pattern.compile("\\s");
```

```
        // String[] java.util.regex.Pattern.split(CharSequence input)
        String[] strArray = pattern1.split(inputStr);
        System.out.println(Arrays.toString(strArray));

        Pattern pattern2 = Pattern.compile(":");
        inputStr = "자바:정규표현식:Pattern클래스:::split메서드 연습";
        // String[] java.util.regex.Pattern.split(CharSequence input, int limit)
        String[] strArray2 = pattern2.split(inputStr, 4);
        System.out.println(Arrays.toString(strArray2));

        // 양수일 때에는 limit 번째부터 split을 하지 않으며, 음수일 때에는 공백 또한 데이터에 포함합니다.
        String[] strArray3 = pattern2.split(inputStr, -1);
        System.out.println(Arrays.toString(strArray3));
        System.out.println();
    }
}
```

결과

```
[자바, 정규표현식, :, Pattern클래스, split메서드, 연습]
[자바, 정규표현식, Pattern클래스, ::split메서드 연습]
[자바, 정규표현식, Pattern클래스, , , split메서드 연습]
```

1-5. quote(String s) 메서드

"[abc"를 포함하는지 검색하고 싶다. 하지만 Pattern.compile("[abc")은 "java.util.regex.PatternSyntaxException: Unclosed character class near index 7" 이란 에러가 발생발생합닌다. 이럴때 사용하는 메서드가 quote이다.
\Q 는 정규 표현식의 시작 표시
\E 는 정규 표현식의 끝을 표시

```
import java.util.regex.Pattern;

public class RegExpEx05_Pattern5 {
    public static void main(String[] args) {
        // "[abc"을 포함하는지 검색하고 싶다. 하지만 다음은 에러가 발생한다.
        // java.util.regex.PatternSyntaxException: Unclosed character class near index 7

        // Pattern pattern5 = Pattern.compile("[abc");
        // System.out.println(pattern5);

        // 이럴때 사용하는 메서드가 quote이다.
        // \Q 는 정규 표현식의 시작 표시
        // \E 는 정규 표현식의 끝을 표시
        // String java.util.regex.Pattern.quote(String s)
        String regex = Pattern.quote("[abc");
        Pattern pattern = Pattern.compile(regex);
        System.out.println(regex + " : " + pattern);
        System.out.println("[abc : " + pattern.matcher("[abc").matches());
        System.out.println();
    }
}
```

결과

```
\Q[abc\E : \Q[abc\E
[abc : true
```

1-6. Pattern.flags() 메서드

사용한 flags를 확인하고 싶다면, java.util.regex.Pattern.flags() 를 통해 확인할 수 있습니다. 참고로, 위의 mehtod는 public int flags() 으로 정의되어 있습니다.

Parttern 플래그 값 사용(상수)

- Pattern.CANON_EQ : None표준화된 매칭 모드를 활성화합니다.
- Pattern.CASE_INSENSITIVE : 대소문자를 구분하지 않습니다.
- Pattern.COMMENTS : 공백과 #으로 시작하는 주석이 무시됩니다. (라인의 끝까지).
- Pattern.MULTILINE : 수식 '^' 는 라인의 시작과, '$' 는 라인의 끝과 match 됩니다.
- Pattern.DOTALL : 수식 '.'과 모든 문자와 match 되고 '\n' 도 match 에 포함됩니다.
- Pattern.UNICODE_CASE : 유니코드를 기준으로 대소문자 구분 없이 match 시킵니다.
- Pattert.UNIX_LINES : 수식 '.' 과 '^' 및 '$'의 match시에 한 라인의 끝을 의미하는 '\n'만 인식됩니다.

```
import java.lang.reflect.Field;
import java.util.regex.Pattern;

public class RegExpEx06_Pattern6 {
    public static void main(String[] args) {
        Pattern pattern1 = Pattern.compile("\\S", Pattern.CASE_INSENSITIVE);
        // int java.util.regex.Pattern.flags()
        System.out.println("pattern6 flags : " + pattern1.flags());
        Pattern pattern2 = Pattern.compile("\\S", Pattern.UNICODE_CASE |
Pattern.DOTALL);
        System.out.println("pattern7 flags : " + pattern2.flags());
        System.out.println();

        // Field 확인
        System.out.println("상수확인 하기");
        Field[] fields = pattern2.getClass().getDeclaredFields();
        for(Field field : fields) {
            if(field.toString().startsWith("public static final"))
                try {
                    // 전체 형식
                    // System.out.println(field);
                    //                 이름                        값
                    System.out.println(field.getName() + " : " +
field.get(pattern2));
                } catch (IllegalArgumentException | IllegalAccessException e) {
                    e.printStackTrace();
                }
        }
    }
}
```

결과

```
pattern6 flags : 2
pattern7 flags : 96

상수확인 하기
UNIX_LINES : 1
CASE_INSENSITIVE : 2
COMMENTS : 4
MULTILINE : 8
LITERAL : 16
DOTALL : 32
UNICODE_CASE : 64
CANON_EQ : 128
UNICODE_CHARACTER_CLASS : 256
```

2. Matcher Class

Matcher 클래스는 대상 문자열의 패턴을 해석하고 주어진 패턴과 일치하는지 판별할 때 주로 사용됩니다. Matcher 클래스의 입력값으로는 CharSequence라는 새로운 인터페이스가 사용되는데 이를 통해 다양한 형태의 입력 데이터로부터 문자 단위의 매칭 기능을 지원 받을 수 있습니다. Matcher객체는 Pattern객체의 matcher() 메소드를 호출하여 받아올 수 있습니다.

Method	Description
find()	주로 텍스트에서 정규식의 다중 발생을 검색하는 데 사용됩니다.
find(int start)	주어진 인덱스에서 시작하는 텍스트에서 정규 표현식의 발생을 검색하는 데 사용됩니다.
start()	find() 메서드를 사용하여 찾은 일치 항목의 시작 인덱스를 가져오는 데 사용됩니다.
end()	find() 메서드를 사용하여 찾은 일치 항목의 끝 인덱스를 가져오는 데 사용됩니다. 마지막으로 일치하는 문자 옆의 문자 인덱스를 반환합니다.
groupCount()	일치하는 하위 시퀀스의 총 수를 찾는 데 사용됩니다.
group()	일치하는 하위 시퀀스를 찾는 데 사용됩니다.
group(int group)	일치하는 인덱스의 하위 시퀀스를 찾는 데 사용됩니다.
matches()	정규식이 패턴과 일치하는지 여부를 테스트하는 데 사용됩니다.
appendReplacement(StringBuffer sb, String replacement)	문자열과 패턴이 일치하는 위치에 replacement를 대체하는 데 사용됩니다.
appendTail(StringBuffer sb)	마지막에 남은 문자열을 붙여주는 데 사용됩니다.
replaceAll(String replacement)	전체를 replacement를 대체하는 데 사용됩니다.

참고: Pattern.matches()는 전체 텍스트가 패턴과 일치하는지 여부를 확인합니다. 다른 방법(아래에 설명됨)은 주로 텍스트에서 여러 패턴을 찾는 데 사용됩니다.

2-1. matches() 메서드

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegExpEx07_Matcher1 {
    public static void main(String[] args) {
        String inputStr = "아름다운 금강산";
        // Matcher java.util.regex.Pattern.matcher(CharSequence input)
        Matcher matcher = Pattern.compile(".*금강산.*").matcher(inputStr);
        if(matcher.matches()) {
            System.out.println("일치하는 내용 찾음");
            System.out.println(matcher.group());
        }else {
            System.out.println("일치하는 내용 못찾음");
        }
        System.out.println();
        matcher = Pattern.compile(".*한라산.*").matcher(inputStr);
        if(matcher.matches()) {
            System.out.println("일치하는 내용 찾음");
            System.out.println(matcher.group());
        }else {
            System.out.println("일치하는 내용 못찾음");
        }
    }
}
```

결과

```
일치하는 내용 찾음
아름다운 금강산

일치하는 내용 못찾음
```

2-2. find()/find(int start) 메서드

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegExpEx08_Matcher1 {
    public static void main(String[] args) {
        String inputStr = "아 지리산 아~ 금강산 아아~~ 한라산 아아아~~~ 백두산 아아아아
~ 뒷동산";
        // Pattern java.util.regex.Pattern.compile(String regex)
        Pattern pattern = Pattern.compile("아*~* ..산");

        // Matcher java.util.regex.Pattern.matcher(CharSequence input)
        Matcher matcher = pattern.matcher(inputStr);
        System.out.print("\"" + inputStr + "\"");
        System.out.println(" 중에서 \"" + pattern + "\"으로 찾는중!!!");
        int count = 0;
        // boolean java.util.regex.Matcher.find()
```

```
        while(matcher.find()){
            count++;
            System.out.println(count + ". " + matcher.group());
        };
        System.out.println("전체 " + count + "개 찾음");
        System.out.println();

        matcher = pattern.matcher(inputStr);
        System.out.print("\"" + inputStr + "\"");
        System.out.println(" 중에서 \"" + pattern + "\"으로 20번째 부터 찾는중!!!");
        count = 0;
        if(matcher.find(20)) {
            do{
                count++;
                System.out.println(count + ". " + matcher.group());
            }while(matcher.find());
            System.out.println("전체 " + count + "개 찾음");
        }else {
            System.out.println("못찾음");
        }
        System.out.println();

        matcher = pattern.matcher(inputStr);
        System.out.print("\"" + inputStr + "\"");
        System.out.println(" 중에서 \"" + pattern + "\"으로 40번째 부터 찾는중!!!");
        count = 0;
        if(matcher.find(40)) {
            do{
                count++;
                System.out.println(count + ". " + matcher.group());
            }while(matcher.find());
            System.out.println("전체 " + count + "개 찾음");
        }else {
            System.out.println("못찾음");
        }
    }
}
```

결과

```
"아 지리산 아~ 금강산 아아~~ 한라산 아아아~~~ 백두산 아아아아~ 뒷동산" 중에서 "아*~* ..
산"으로 찾는중!!!
1. 아 지리산
2. 아~ 금강산
3. 아아~~ 한라산
4. 아아아~~~ 백두산
5. 아아아아~ 뒷동산
전체 5개 찾음

"아 지리산 아~ 금강산 아아~~ 한라산 아아아~~~ 백두산 아아아아~ 뒷동산" 중에서 "아*~* ..
산"으로 20번째 부터 찾는중!!!
1. 아아아~~~ 백두산
2. 아아아아~ 뒷동산
전체 2개 찾음

"아 지리산 아~ 금강산 아아~~ 한라산 아아아~~~ 백두산 아아아아~ 뒷동산" 중에서 "아*~* ..
산"으로 40번째 부터 찾는중!!!
```

```
못찾음
```

2-3. start()/end()메서드

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegExpEx09_Matcher3 {
    public static void main(String[] args) {
        String inputStr = "아 지리산 아~ 금강산 아아~~ 한라산 아아아~~~ 백두산 아아아아
~ 뒷동산";
        Pattern pattern = Pattern.compile("..산");
        Matcher matcher = pattern.matcher(inputStr);
        System.out.println("\"" + inputStr + "\" 중에서 \"..산\"이란 문자가 있는 위
치");
        int count = 0;
        while (matcher.find()) {
            System.out.print("찾은 위치값 :\t" + matcher.start() + "~" +
(matcher.end()-1));
            System.out.println(" ==> " +
inputStr.substring(matcher.start(),matcher.end()));
            count++;
        }
        System.out.println("총 " + count + "개 찾음\n");
    }
}
```

결과

```
"아 지리산 아~ 금강산 아아~~ 한라산 아아아~~~ 백두산 아아아아~ 뒷동산" 중에서 "..산"이란
문자가 있는 위치
찾은 위치값 :    2~4 ==> 지리산
찾은 위치값 :    9~11 ==> 금강산
찾은 위치값 :    18~20 ==> 한라산
찾은 위치값 :    29~31 ==> 백두산
찾은 위치값 :    39~41 ==> 뒷동산
총 5개 찾음
```

2-4. groupCount() 메서드

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegExpEx10_Matcher4 {
    public static void main(String[] args) {
        // int java.util.regex.Matcher.groupCount() : 패턴에 있는 그룹 ()의 개수
        String inputStr = "아 지리산 아~ 금강산 아아~~ 한라산 아아아~~~ 백두산 아아아아
~ 뒷동산";
        Pattern pattern1 = Pattern.compile("아*~* ..산");
        Matcher matcher1 = pattern1.matcher(inputStr);
        System.out.println(pattern1 + ".groupCount() : " + matcher1.groupCount()
+ "개");
```

```
        Pattern pattern2 = Pattern.compile("(아*)~* ..산");
        Matcher matcher2 = pattern2.matcher(inputStr);
        System.out.println(pattern2 + ".groupCount() : " + matcher2.groupCount()
+ "개");

        Pattern pattern3 = Pattern.compile("(아*)(~*) (..산)");
        Matcher matcher3 = pattern3.matcher(inputStr);
        System.out.println(pattern3 + ".groupCount() : " + matcher3.groupCount()
+ "개");

        Pattern pattern4 = Pattern.compile("((아*)(~*)( )(..산))");
        Matcher matcher4 = pattern4.matcher(inputStr);
        System.out.println(pattern4 + ".groupCount() : " + matcher4.groupCount()
+ "개");
    }
}
```

결과

```
아*~* ..산.groupCount() : 0개
(아*)~* ..산.groupCount() : 1개
(아*)(~*) (..산).groupCount() : 3개
((아*)(~*)( )(..산)).groupCount() : 5개
```

2-5. group()/group(int group) 메서드

group()과 group(0)

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegExpEx11_Matcher5 {
    public static void main(String[] args) {
        String inputStr = "아 지리산 아~ 금강산 아아~~ 한라산 아아아~~~ 백두산 아아아아
~ 뒷동산";
        Pattern pattern = Pattern.compile("(아*)(~*)( )(..산)");
        Matcher matcher = pattern.matcher(inputStr);
        System.out.println("그룹의 개수 : " + matcher.groupCount() + "개");
        while(matcher.find()) {
            // String java.util.regex.Matcher.group()
            System.out.println("group() :\t" + matcher.group());
            for(int i=0;i<=matcher.groupCount();i++) {
                // String java.util.regex.Matcher.group(int group)
                System.out.println(String.format("group(%d)
:\t%s",i,matcher.group(i)));
            }
            System.out.println();
        }
    }
}
```

결과

```
그룹의 개수 : 4개
group() :  아 지리산
group(0) :  아 지리산
group(1) :  아
group(2) :
group(3) :
group(4) :  지리산

group() :  아~ 금강산
group(0) :  아~ 금강산
group(1) :  아
group(2) :  ~
group(3) :
group(4) :  금강산

group() :  아아~~ 한라산
group(0) :  아아~~ 한라산
group(1) :  아아
group(2) :  ~~
group(3) :
group(4) :  한라산

group() :  아아아~~~ 백두산
group(0) :  아아아~~~ 백두산
group(1) :  아아아
group(2) :  ~~~
group(3) :
group(4) :  백두산

group() :  아아아아~ 뒷동산
group(0) :  아아아아~ 뒷동산
group(1) :  아아아아
group(2) :  ~
group(3) :
group(4) :  뒷동산
```

2-6. appendReplacement(StringBuffer sb, String replacement)/appendTail(StringBuffer sb) 메서드

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegExpEx12_Matcher6 {
    public static void main(String[] args) {
        String inputStr = "아 지리산 아~ 금강산 아아~~ 한라산 아아아~~~ 백두산 아아아아
~ 뒷동산";
        System.out.println("원본 : " + inputStr);
        Pattern pattern = Pattern.compile("~");
        Matcher matcher = pattern.matcher(inputStr);
        StringBuffer buffer = new StringBuffer();
        while (matcher.find()) {
            // Matcher java.util.regex.Matcher.appendReplacement(StringBuffer
sb, String replacement)
            // 문자열과 패턴이 일치하는 위치에 replacement를 대체하는 데 사용됩니다.
            matcher.appendReplacement(buffer, "!");
```

```
            System.out.println(buffer);
        }
        System.out.println(); // 맨뒤 문자열 뒷동산이 사라짐 이때 appendTail()을 사용함
        // StringBuffer java.util.regex.Matcher.appendTail(StringBuffer sb)
        // 뒤에 남은 문자열을 붙여주는데 사용합니다.
        matcher.appendTail(buffer);
        System.out.println(buffer);
        System.out.println();
    }
}
```

결과

```
원본 : 아 지리산 아~ 금강산 아아~~ 한라산 아아아~~~ 백두산 아아아아~ 뒷동산
아 지리산 아!
아 지리산 아! 금강산 아아!
아 지리산 아! 금강산 아아!!
아 지리산 아! 금강산 아아!! 한라산 아아아!
아 지리산 아! 금강산 아아!! 한라산 아아아!!
아 지리산 아! 금강산 아아!! 한라산 아아아!!!
아 지리산 아! 금강산 아아!! 한라산 아아아!!! 백두산 아아아아!

아 지리산 아! 금강산 아아!! 한라산 아아아!!! 백두산 아아아아! 뒷동산
```

2-7. Matcher.replaceAll(String replacement)/String.replaceAll(String regex, String replacement) 메서드

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegExpEx13_Matcher7 {
    public static void main(String[] args) {
        String inputStr = "아 지리산 아~ 금강산 아아~~ 한라산 아아아~~~ 백두산 아아아아~ 뒷동산";
        Pattern pattern = Pattern.compile("~");
        Matcher matcher = pattern.matcher(inputStr);

        // String java.util.regex.Matcher.replaceAll(String replacement)
        System.out.println("원본 : " + inputStr);
        String outputStr1 = matcher.replaceAll("!!");
        System.out.println("변경 : " + outputStr1);
        System.out.println();

        // String java.lang.String.replaceAll(String regex, String replacement)
        // replaceAll메서드 에서도 정규 표현식 사용가능
        System.out.println("원본 : " + inputStr);
        String outputStr2 = inputStr.replaceAll("아", "오");
        System.out.println("변경 : " + outputStr2);
        outputStr2 = inputStr.replaceAll("~", "!!");
        System.out.println("변경 : " + outputStr2);
    }
}
```

결과

```
원본 : 아 지리산 아~ 금강산 아아~~ 한라산 아아아~~~ 백두산 아아아아~ 뒷동산
변경 : 아 지리산 아!! 금강산 아아!!!! 한라산 아아아!!!!!! 백두산 아아아아!! 뒷동산

원본 : 아 지리산 아~ 금강산 아아~~ 한라산 아아아~~~ 백두산 아아아아~ 뒷동산
변경 : 오 지리산 오~ 금강산 오오~~ 한라산 오오오~~~ 백두산 오오오오~ 뒷동산
변경 : 아 지리산 아!! 금강산 아아!!!! 한라산 아아아!!!!!! 백두산 아아아아!! 뒷동산
```

3. 정규표현식 문법

정규 표현식	설명
^	문자열 시작
$	문자열 종료
.	임의의 한 문자(단 \은 넣을 수 없음)
*	앞 문자가 없을 수도 무한정 많을 수도 있음
+	앞 문자가 하나 이상
?	앞 문자가 없거나 하나 있음
[ ]	문자의 집합이나 범위를 나타내며 두 문자 사이는 - 기호로 범위를 나타냅니다. [] 내에서 ^ 가 선행하여 존재하면 not을 나타낸다.
{ }	횟수 또는 범위를 나타냅니다.
( )	소괄호 안의 문자를 하나의 문자로 인식
|	패턴 안에서 or 연산을 수행할 때 사용
\	정규 표현식 역슬래시()는 확장문자 (역슬래시 다음에 일반 문자가 오면 특수문자로 취급하고 역슬래시 다음에 특수문자가 오면 그 문자 자체를 의미)
\b	단어의 경계
\B	단어가 아닌것에 대한 경계
\A	입력의 시작 부분
\G	이전 매치의 끝
\Z	입력의 끝이지만 종결자가 있는 경우
\z	입력의 끝
\s	공백 문자
\S	공백 문자가 아닌 나머지 문자
\w	알파벳이나 숫자
\W	알파벳이나 숫자를 제외한 문자
\d	숫자 [0-9]와 동일
\D	숫자를 제외한 모든 문자
(?i)	앞 부분에 (?!)라는 옵션을 넣어주게 되면 대소문자는 구분하지 않습니다.

3-1. 정규 표현식 문법 1

정규 표현식	설명
^[0-9]*$	숫자
^[a-zA-Z]*$	영문자
^[ㄱ-ㅎㅏ-ㅣ가-힣]*$	한글

숫자

```
import java.util.Arrays;
import java.util.regex.Pattern;

public class RegExpEx14_Example1 {
    public static void main(String[] args) {
        String inputStr = "0사과1배2수박3바나나123-qwerty&$#@45QWERTY77~!@#$%";
        System.out.println("원본 : " + inputStr);
        System.out.println();
        // 1. 바꾸기
        System.out.println("1. 숫자만 뽑고싶다.");
        // 숫자가 아닌 문자만 지우기
        System.out.println(inputStr.replaceAll("[^0-9]+", ""));
        System.out.println(inputStr.replaceAll("\\D+", ""));
        System.out.println();

        System.out.println("2. 숫자가 아닌 문자만 뽑고싶다.");
        // 숫자만 지우기
        System.out.println(inputStr.replaceAll("[0-9]+", ""));
        System.out.println(inputStr.replaceAll("\\d+", ""));
        System.out.println();

        // 2. 나누기
        System.out.println("3. 숫자를 구분자로 배열 만들기");
        System.out.println(Arrays.toString(inputStr.split("\\d+")));

System.out.println(Arrays.toString(Pattern.compile("\\d+").split(inputStr)));
        System.out.println();

        System.out.println("4. 숫자가 아닌 문자를 구분자로 배열 만들기");
        System.out.println(Arrays.toString(inputStr.split("\\D+")));

System.out.println(Arrays.toString(Pattern.compile("\\D+").split(inputStr)));
        System.out.println();

        // 3. 검사하기
        String str1 = "1234567890";
        String str2 = "123456789a";
        System.out.println("5. 숫자만으로 이루어졌는지 검사하기");
        System.out.print(str1 + " : " + Pattern.matches("[0-9]+", str1));
        System.out.println(" : " + (Pattern.matches("[0-9]+", str1)? "숫자로만 구
성":"숫자가 아닌 문자포함"));
        System.out.print(str2 + " : " + Pattern.matches("\\d+", str2));
        System.out.println(" : " + (Pattern.matches("\\d+", str2)? "숫자로만 구
성":"숫자가 아닌 문자포함"));
    }
```

```
}
```

결과

```
원본 : 0사과1배2수박3바나나123-qwerty&$#@45QWERTY77~!@#$%

1. 숫자만 뽑고싶다.
01231234577
01231234577

2. 숫자가 아닌 문자만 뽑고싶다.
사과배수박바나나-qwerty&$#@QWERTY~!@#$%
사과배수박바나나-qwerty&$#@QWERTY~!@#$%

3. 숫자를 구분자로 배열 만들기
[, 사과, 배, 수박, 바나나, -qwerty&$#@, QWERTY, ~!@#$%]
[, 사과, 배, 수박, 바나나, -qwerty&$#@, QWERTY, ~!@#$%]

4. 숫자가 아닌 문자를 구분자로 배열 만들기
[0, 1, 2, 3, 123, 45, 77]
[0, 1, 2, 3, 123, 45, 77]

5. 숫자만으로 이루어졌는지 검사하기
1234567890 : true : 숫자로만 구성
123456789a : false : 숫자가 아닌 문자포함
```

영문자

```
import java.util.Arrays;
import java.util.regex.Pattern;

public class RegExpEx15_Example2 {
    public static void main(String[] args) {
        String inputStr = "0사과1배2수박3바나나123-qwerty&$#@45QWERTY77~!@#$%";
        System.out.println("원본 : " + inputStr);
        System.out.println();
        // 1. 바꾸기
        System.out.println("1. 영문자만 뽑고싶다.");
        // 영문자가 아닌 문자만 지우기
        System.out.println(inputStr.replaceAll("[^a-zA-Z]+", ""));
        System.out.println();

        System.out.println("2. 영문자가 아닌 문자만 뽑고싶다.");
        // 영문자만 지우기
        System.out.println(inputStr.replaceAll("[a-zA-Z]+", ""));
        System.out.println();

        // 2. 나누기
        System.out.println("3. 영문자를 구분자로 배열 만들기");
        System.out.println(Arrays.toString(inputStr.split("[a-zA-Z]+")));
        System.out.println(Arrays.toString(Pattern.compile("[a-zA-
Z]+").split(inputStr)));
        System.out.println();

        System.out.println("4. 영문자가 아닌 문자를 구분자로 배열 만들기");
```

```
        System.out.println(Arrays.toString(inputStr.split("[^a-zA-Z]+")));
        System.out.println(Arrays.toString(Pattern.compile("[^a-zA-
Z]+").split(inputStr)));
        System.out.println();

        // 3. 검사하기
        String str1 = "qwertyQWERTY";
        String str2 = "qwertyQWERTY2";
        System.out.println("5. 영문자만으로 이루어졌는지 검사하기");
        System.out.print(str1 + " : " + Pattern.matches("[a-zA-Z]+", str1));
        System.out.println(" : " + (Pattern.matches("[a-zA-Z]+", str1)? "영문자로
만 구성":"영문자가 아닌 문자포함"));
        System.out.print(str2 + " : " + Pattern.matches("[a-zA-Z]+", str2));
        System.out.println(" : " + (Pattern.matches("[a-zA-Z]+", str2)? "영문자로
만 구성":"영문자가 아닌 문자포함"));
    }
}
```

결과

```
원본 : 0사과1배2수박3바나나123-qwerty&$#@45QWERTY77~!@#$%

1. 영문자만 뽑고싶다.
qwertyQWERTY

2. 영문자가 아닌 문자만 뽑고싶다.
0사과1배2수박3바나나123-&$#@4577~!@#$%

3. 영문자를 구분자로 배열 만들기
[0사과1배2수박3바나나123-, &$#@45, 77~!@#$%]
[0사과1배2수박3바나나123-, &$#@45, 77~!@#$%]

4. 영문자가 아닌 문자를 구분자로 배열 만들기
[, qwerty, QWERTY]
[, qwerty, QWERTY]

5. 영문자만으로 이루어졌는지 검사하기
qwertyQWERTY : true : 영문자로만 구성
qwertyQWERTY2 : false : 영문자가 아닌 문자포함
```

한글

```
import java.util.Arrays;
import java.util.regex.Pattern;

public class RegExpEx16_Example3 {
    public static void main(String[] args) {
        String inputStr = "0사과1배2수박3바나나123ㄱㄹㄴㅏㅕㅡㅣqwerty";
        System.out.println("원본 : " + inputStr);
        System.out.println();
        // 1. 바꾸기
        System.out.println("1. 한글만 뽑고싶다.");
        // 한글이 아닌 문자만 지우기
        System.out.println(inputStr.replaceAll("[^ㄱ-ㅎㅏ-ㅣ가-힣]+", ""));
        System.out.println();
```

```
        System.out.println("2. 한글이 아닌 문자만 뽑고싶다.");
        // 한글만 지우기
        System.out.println(inputStr.replaceAll("[ㄱ-ㅎㅏ-ㅣ가-힣]+", ""));
        System.out.println();

        // 2. 나누기
        System.out.println("3. 한글을 구분자로 배열 만들기");
        System.out.println(Arrays.toString(inputStr.split("[ㄱ-ㅎㅏ-ㅣ가-힣]+")));
        System.out.println(Arrays.toString(Pattern.compile("[ㄱ-ㅎㅏ-ㅣ가-
힣]+").split(inputStr)));
        System.out.println();

        System.out.println("4. 한글이 아닌 문자를 구분자로 배열 만들기");
        System.out.println(Arrays.toString(inputStr.split("[^ㄱ-ㅎㅏ-ㅣ가-
힣]+")));
        System.out.println(Arrays.toString(Pattern.compile("[^ㄱ-ㅎㅏ-ㅣ가-
힣]+").split(inputStr)));
        System.out.println();

        // 3. 검사하기
        String str1 = "한글만으로구성됨";
        String str2 = "한글만으로구성됨2";
        System.out.println("5. 한글로만 이루어졌는지 검사하기");
        System.out.print(str1 + " : " + Pattern.matches("[ㄱ-ㅎㅏ-ㅣ가-힣]+",
str1));
        System.out.println(" : " + (Pattern.matches("[ㄱ-ㅎㅏ-ㅣ가-힣]+", str1)?
"한글만으로 구성":"한글이 아닌 문자포함"));
        System.out.print(str2 + " : " + Pattern.matches("[ㄱ-ㅎㅏ-ㅣ가-힣]+",
str2));
        System.out.println(" : " + (Pattern.matches("[ㄱ-ㅎㅏ-ㅣ가-힣]+", str2)?
"한글만으로 구성":"한글이 아닌 문자포함"));
        // 잘못된 정규 표현식
        str1 = "한글만으로구성됨ㄱㄴㄷㄹ";
        str2 = "한글만으로구성됨ㅏㅑㅓㅕ";
        System.out.print(str1 + " : " + Pattern.matches("[가-힣]+", str1));
        System.out.println(" : " + (Pattern.matches("[가-힣]+", str1)? "한글만으로
구성":"한글이 아닌 문자포함"));
        System.out.print(str2 + " : " + Pattern.matches("[가-힣]+", str2));
        System.out.println(" : " + (Pattern.matches("[가-힣]+", str2)? "한글만으로
구성":"한글이 아닌 문자포함"));
    }
}
```

결과

```
원본 : 0사과1배2수박3바나나123ㄱㄹㄴㅏㅑㅡㅣqwerty

1. 한글만 뽑고싶다.
사과배수박바나나ㄱㄹㄴㅏㅑㅡㅣ

2. 한글이 아닌 문자만 뽑고싶다.
0123123qwerty

3. 한글을 구분자로 배열 만들기
[0, 1, 2, 3, 123, qwerty]
[0, 1, 2, 3, 123, qwerty]
```

```
4. 한글이 아닌 문자를 구분자로 배열 만들기
[, 사과, 배, 수박, 바나나, ㄱㄹㄴㅏㅑㅕㅡㅣ]
[, 사과, 배, 수박, 바나나, ㄱㄹㄴㅏㅑㅕㅡㅣ]

5. 한글로만 이루어졌는지 검사하기
한글만으로구성됨 : true : 한글만으로 구성
한글만으로구성됨2 : false : 한글이 아닌 문자포함
한글만으로구성됨ㄱㄴㄷㄹ : false : 한글이 아닌 문자포함
한글만으로구성됨ㅏㅑㅓㅕ : false : 한글이 아닌 문자포함
```

공백제거

```
import java.util.regex.Pattern;

public class RegExpEx17_Example4 {
    // 문자열에서 공백 제거하기
    public static void main(String[] args) {
        String str1 = "   공백 공백 공백   ";
        String str2 = "     탭       탭   탭   ";
        String str3 = "공백 \f탭\t탭\n탭\r\n개행";
        System.out.println("문자열 1 : " + str1);
        System.out.println("문자열 2 : " + str2);
        System.out.println("문자열 3 : " + str3);
        System.out.println();

        System.out.println(str1.replaceAll("\\s", ""));
        System.out.println(str1.replaceAll("[\\f\\t\\r\\n ]", ""));
        System.out.println(Pattern.compile("\\s").matcher(str1).replaceAll(""));
        System.out.println();

        System.out.println(str2.replaceAll("\\s", ""));
        System.out.println(str2.replaceAll("[\\f\\t\\r\\n ]", ""));
        System.out.println(Pattern.compile("\\s").matcher(str2).replaceAll(""));
        System.out.println();

        System.out.println(str3.replaceAll("\\s", ""));
        System.out.println(str3.replaceAll("[\\f\\t\\r\\n ]", ""));
        System.out.println(Pattern.compile("\\s").matcher(str3).replaceAll(""));
        System.out.println();
    }
}
```

결과

```
문자열 1 :    공백 공백 공백
문자열 2 :          탭       탭   탭
문자열 3 : 공백 •탭   탭
탭
개행

공백공백공백
공백공백공백
공백공백공백

탭탭탭
탭탭탭
```

```
탭탭탭

공백탭탭탭개행
공백탭탭탭개행
공백탭탭탭개행
```

3-2. 정규 표현식 문법 2

n글자씩 잘라 배열/리스트로 만들기

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegExpEx18_Example5 {
    public static void main(String[] args) {
        String inputStr = "대한민국123456qwertyQWERTY";
        // 1글자씩 잘라 배열/리스트로 만들기
        Pattern pattern = Pattern.compile("");
        System.out.println(Arrays.toString(pattern.split(inputStr)));
        System.out.println(Arrays.toString(inputStr.split("")));
        System.out.println(Arrays.asList(inputStr.split("")));
        System.out.println();

        System.out.println(splitLength(inputStr, 1));
        System.out.println(splitLength(inputStr, 3));
        System.out.println(splitLength(inputStr, 5));
        System.out.println(splitLength(null, 7));
    }
    // 문자열을 길이만큼씩 잘라 List<String>로 리턴하는 메서드
    // parameter : 문자열,길이
    public static List<String> splitLength(String inputStr, int length){
        List<String> list = null;
        if(inputStr!=null) {
            list = new ArrayList<String>();
            int end = 0;
            Pattern pattern = Pattern.compile(".{"+length+"}");
            Matcher matcher = pattern.matcher(inputStr);
            while (matcher.find()) {
                list.add(matcher.group());
                end = matcher.end();
            }
            if(end<inputStr.length()) list.add(inputStr.substring(end));
        }
        return list;
    }
}
```

결과

```
[대, 한, 민, 국, 1, 2, 3, 4, 5, 6, q, w, e, r, t, y, Q, W, E, R, T, Y]
[대, 한, 민, 국, 1, 2, 3, 4, 5, 6, q, w, e, r, t, y, Q, W, E, R, T, Y]
[대, 한, 민, 국, 1, 2, 3, 4, 5, 6, q, w, e, r, t, y, Q, W, E, R, T, Y]

[대, 한, 민, 국, 1, 2, 3, 4, 5, 6, q, w, e, r, t, y, Q, W, E, R, T, Y]
[대한민, 국12, 345, 6qw, ert, yQW, ERT, Y]
[대한민국1, 23456, qwert, yQWER, TY]
null
```

특정 문자열을 포함하는지 판단

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegExpEx19_Example6 {
    public static void main(String[] args) {
        String inputStr = "동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라만
세";
        System.out.println("원본 : " + inputStr);

        // 특정 문자열을 포함하는지 판단
        // .* : 임의의 문자가 0개 이상
        String findStr = "나라";
        boolean isIncluded = inputStr.matches(".*" + findStr + ".*");
        System.out.println(findStr + " " + (isIncluded ? "포함" : "미포함"));

        isIncluded = inputStr.contains(findStr);
        System.out.println(findStr + " " + (isIncluded ? "포함" : "미포함"));

        isIncluded = Pattern.matches(".*" + findStr + ".*", inputStr);
        System.out.println(findStr + " " + (isIncluded ? "포함" : "미포함"));

        Pattern pattern = Pattern.compile(".*" + findStr + ".*");
        Matcher matcher = pattern.matcher(inputStr);
        isIncluded = matcher.matches();
        System.out.println(findStr + " " + (isIncluded ? "포함" : "미포함"));
        System.out.println();

        findStr = "국가";
        isIncluded = inputStr.matches(".*" + findStr + ".*");
        System.out.println(findStr + " " + (isIncluded ? "포함" : "미포함"));

        isIncluded = inputStr.contains(findStr);
        System.out.println(findStr + " " + (isIncluded ? "포함" : "미포함"));

        isIncluded = Pattern.matches(".*" + findStr + ".*", inputStr);
        System.out.println(findStr + " " + (isIncluded ? "포함" : "미포함"));

        pattern = Pattern.compile(".*" + findStr + ".*");
        matcher = pattern.matcher(inputStr);
        isIncluded = matcher.matches();
        System.out.println(findStr + " " + (isIncluded ? "포함" : "미포함"));
        System.out.println();

        // contains()와 matches()의 차이점
        // contains()는 단순히 인자로 전달된 문자열이 존재하는지 여부를 리턴합니다.
```

```
        // 반면에 matches()는 정규표현식을 인자로 받고 동일한 패턴의 문자열이라면 true를
리턴합니다.
        // 대소문자 구분없이 일치하는지 검사하고 싶다.
        // [Tt] : T 또는 t
        System.out.println("contains() 사용");
        inputStr = "True";
        System.out.println("원본 : " + inputStr);
        boolean isMatches = inputStr.contains("true") ||
inputStr.contains("True");
        System.out.println(isMatches ? "일치" : "불일치");
        isMatches = inputStr.toLowerCase().contains("true");
        System.out.println(isMatches ? "일치" : "불일치");
        System.out.println();

        System.out.println("정규표현식 사용");
        isMatches = inputStr.matches("[Tt]rue");
        System.out.println(isMatches ? "일치" : "불일치");
        isMatches = Pattern.matches("[Tt]rue", inputStr);
        System.out.println(isMatches ? "일치" : "불일치");
        System.out.println();
    }
}
```

결과

```
원본 : 동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라만세
나라 포함
나라 포함
나라 포함
나라 포함

국가 미포함
국가 미포함
국가 미포함
국가 미포함

contains() 사용
원본 : True
일치
일치

정규표현식 사용
일치
일치
```

임의의 한 문자와 매치

```
import java.util.Arrays;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegExpEx20_Example7 {
    public static void main(String[] args) {
        // . : 임의의 한 문자와 매치됩니다.
        Pattern pattern = Pattern.compile("..");
```

```
        String inputStr = "대한민국123456qwertyQWERTY";
        System.out.println("원본 : " + inputStr);

        // 2글자씩 잘라내기
        Matcher matcher = pattern.matcher(inputStr);
        System.out.print(pattern + " : ");
        while(matcher.find()) {
            System.out.print(matcher.group() + " ");
        }
        System.out.println("\n");

        // ".s" : 첫번째 글자는 임의의 1문자이고 두번째 문자가 "s"인 문자인가?
        inputStr = "s,is,AS,was";
        String[] strArray = inputStr.split(",");
        System.out.println("원본 : " + Arrays.toString(strArray));

        for(String str : strArray) {
            System.out.println(str + " : " +
Pattern.compile(".s").matcher(str).matches());
        }
        System.out.println();

        // ".[sS]" : 첫번째 글자는 임의의 1문자이고 두번째 문자가 "s" 또는 "S"인 문자인
가?
        for(String str : strArray) {
            System.out.println(str + " : " + Pattern.compile(".
[sS]").matcher(str).matches());
        }
        System.out.println();

        // Pattern.CASE_INSENSITIVE : 대소문자 무시 옵션
        for(String str : strArray) {
            System.out.println(str + " : " + Pattern.compile(".s",
Pattern.CASE_INSENSITIVE).matcher(str).matches());
        }
        System.out.println();
    }
}
```

결과

```
원본 : 대한민국123456qwertyQWERTY
.. : 대한 민국 12 34 56 qw er ty QW ER TY

원본 : [s, is, AS, was]
s : false
is : true
AS : false
was : false

s : false
is : true
AS : true
was : false

s : false
is : true
```

```
AS : true
was : false
```

시작과 끝 일치

```
import java.util.regex.Pattern;

public class RegExpEx21_Example8 {
    public static void main(String[] args) {
        // ^ : 문자열의 시작을 나타냄
        // $ : 문자열의 끝을 나타냄
        String[] domains =
"http://www.aaa.com,https://bbb.com,https://bbb.net,www.bbb.org".split(",");
        // System.out.println("원본 : " + Arrays.toString(domains));
        System.out.println("도메인 이름이 http로 시작되는가?");
        for(String domain : domains) {
            System.out.println(domain + " : " + Pattern.matches("^http(.*)",
domain));
        }
        System.out.println();
        System.out.println("도메인 이름이 .com으로 끝나는가?");
        for(String domain : domains) {
            System.out.println(domain + " : " + Pattern.matches(".*\\.com$",
domain));
            //System.out.println(domain + " : " + Pattern.matches(".*\\.com",
domain));
        }
        System.out.println();

        System.out.println("도메인 이름이 http로 시작해서 .com으로 끝나는가?");
        for(String domain : domains) {
            System.out.println(domain + " : " + Pattern.matches("^(http)(.*)
(\\.com)$", domain));
        }
        System.out.println();

        // 그런데 만약, 문자열에 포함되어있는 .(온점)을 검색하고 싶을 때에는 어떻게 해야할
까요?
        // 다음은 This로 시작하고 "."으로 끝나는 패턴을 찾습니다. 만약 "."으로 끝나지 않
으면 false를 리턴합니다.
        System.out.println("This is the right decision.".matches("This.*\\."));
        System.out.println("This is me".matches("This.*\\."));
        System.out.println("This is my mistake.".matches("This.*\\."));
        System.out.println();
    }
}
```

결과

```
도메인 이름이 http로 시작되는가?
http://www.aaa.com : true
https://bbb.com : true
https://bbb.net : true
www.bbb.org : false
```

```
도메인 이름이 .com으로 끝나는가?
http://www.aaa.com : true
https://bbb.com : true
https://bbb.net : false
www.bbb.org : false

도메인 이름이 http로 시작해서 .com으로 끝나는가?
http://www.aaa.com : true
https://bbb.com : true
https://bbb.net : false
www.bbb.org : false

true
false
true
```

몇글자 일치

```
import java.util.regex.Pattern;

public class RegExpEx22_Example9 {
    public static void main(String[] args) {
        // ? : 바로 앞의 문자가 없거나 하나임
        // + : 바로 앞의 문자가 하나이상 있음
        // * : 바로 앞의 문자가 없거나 하나이상 있음
        // | : 또는, or, 합집합의 개념으로 사용
        //() : '()'안의 문자열을 하나로 묶어서 다룸
        //[] : '[]'안은 범위와 집합의 개념으로 다룸
        //{n,m} : 바로 앞의 문자가 n글자 이상 m글자개 반복, m이 없으면 n개 이상
        String[] inputArray = "az,abz,acz,abbz,abbbz".split(",");
        System.out.println("a로시작되고 중간에 임의문자가 없거나 1개있으며 z로 종료 ");
        for(String str : inputArray) {
            System.out.println(str + " : " + Pattern.matches("a.?z", str));
        }
        System.out.println();

        System.out.println("a로시작되고 중간에 b가 없거나 1개있으며 z로 종료 ");
        for(String str : inputArray) {
            System.out.println(str + " : " + Pattern.matches("ab?z", str));
        }
        System.out.println();

        System.out.println("a로시작되고 중간에 b가 없거나 n개있으며 z로 종료 ");
        for(String str : inputArray) {
            System.out.println(str + " : " + Pattern.matches("ab*z", str));
        }
        System.out.println();

        System.out.println("a로시작되고 중간에 b가 n개있으며 z로 종료 ");
        for(String str : inputArray) {
            System.out.println(str + " : " + Pattern.matches("ab+z", str));
        }
        System.out.println();

        System.out.println("a로시작되고 중간에 b 또는 c가 1개있으며 z로 종료 ");
```

```
        for(String str : inputArray) {
            System.out.println(str + " : " + Pattern.matches("a(b|c)z", str));
        }
        System.out.println();

        System.out.println("a로시작되고 중간에 b 또는 c가 1개있으며 z로 종료 ");
        for(String str : inputArray) {
            System.out.println(str + " : " + Pattern.matches("a[bc]z", str)); //
b or c
        }
        System.out.println();

        System.out.println("a로시작되고 중간에 b가 2개있으며 z로 종료 ");
        for(String str : inputArray) {
            System.out.println(str + " : " + Pattern.matches("ab{2,2}z", str));
// b가 2개
            // System.out.println(str + " : " + Pattern.matches("abbz", str));
//
        }
        System.out.println();

        System.out.println("a로시작되고 중간에 b가 2개 이상 있으며 z로 종료 ");
        for(String str : inputArray) {
            System.out.println(str + " : " + Pattern.matches("ab{2,}z", str));
// b가 2개 이상
        }
        System.out.println();

        System.out.println("a로시작되고 중간에 b가 2개 이하 있으며 z로 종료 ");
        for(String str : inputArray) {
            System.out.println(str + " : " + Pattern.matches("ab{0,2}z", str));
// b가 2개 이하
        }
        System.out.println();
    }
}
```

결과

```
a로시작되고 중간에 임의문자가 없거나 1개있으며 z로 종료
az : true
abz : true
acz : true
abbz : false
abbbz : false

a로시작되고 중간에 b가 없거나 1개있으며 z로 종료
az : true
abz : true
acz : false
abbz : false
abbbz : false

a로시작되고 중간에 b가 없거나 n개있으며 z로 종료
az : true
abz : true
acz : false
```

```
abbz : true
abbbz : true

a로시작되고 중간에 b가 n개있으며 z로 종료
az : false
abz : true
acz : false
abbz : true
abbbz : true

a로시작되고 중간에 b 또는 c가 1개있으며 z로 종료
az : false
abz : true
acz : true
abbz : false
abbbz : false

a로시작되고 중간에 b 또는 c가 1개있으며 z로 종료
az : false
abz : true
acz : true
abbz : false
abbbz : false

a로시작되고 중간에 b가 2개있으며 z로 종료
az : false
abz : false
acz : false
abbz : true
abbbz : false

a로시작되고 중간에 b가 2개 이상 있으며 z로 종료
az : false
abz : false
acz : false
abbz : true
abbbz : true

a로시작되고 중간에 b가 2개 이하 있으며 z로 종료
az : true
abz : true
acz : false
abbz : true
abbbz : false
```

영문자/숫자 일치

```java
import java.util.regex.Pattern;

public class RegExpEx23_Example10 {
    public static void main(String[] args) {
        // ? : 바로 앞의 문자가 없거나 하나임
        // + : 바로 앞의 문자가 하나이상 있음
        // * : 바로 앞의 문자가 없거나 하나이상 있음
        // | : 또는, or, 합집합의 개념으로 사용
```

```
        //() : '()'안의 문자열을 하나로 묶어서 다룸
        //[] : '[]'안은 범위와 집합의 개념으로 다룸
        //{n,m} : 바로 앞의 문자가 n글자 이상 m글자개 반복, m이 없으면 n개 이상
        String[] inputArray =
"1,12,123,1234,a,ab,abc,abcd,A,AB,ABC,ABCD".split(",");
        System.out.println("숫자 만으로 1개 이상");
        for(String str : inputArray) {
            System.out.println(str + " : " + Pattern.matches("[0-9]+", str));
        }
        System.out.println();

        System.out.println("알파벳 소문자 만으로 1개 이상");
        for(String str : inputArray) {
            System.out.println(str + " : " + Pattern.matches("[a-z]+", str));
        }
        System.out.println();

        System.out.println("알파벳 대, 소문자 구분없이 1개 이상");
        for(String str : inputArray) {
            System.out.println(str + " : " + Pattern.matches("[a-zA-Z]+", str));
        }
        System.out.println();

        System.out.println("알파벳 대, 소문자 구분없이 2,3개");
        for(String str : inputArray) {
            System.out.println(str + " : " + Pattern.matches("[a-zA-Z]{2,3}",
str));
        }
        System.out.println();

        System.out.println("숫자, 알파벳 대, 소문자 구분없이 2,3개");
        for(String str : inputArray) {
            System.out.println(str + " : " + Pattern.matches("[0-9a-zA-Z]{2,3}",
str));
        }
        System.out.println();

        // []안의 ^는 반대의 개념임
        System.out.println("숫자, 알파벳 대문자가 아닌 2,3개");
        for(String str : inputArray) {
            System.out.println(str + " : " + Pattern.matches("[^0-9A-Z]{2,3}",
str));
        }
        System.out.println();
    }
}
```

결과

```
숫자 만으로 1개 이상
1 : true
12 : true
123 : true
1234 : true
a : false
ab : false
abc : false
```

```
abcd : false
A : false
AB : false
ABC : false
ABCD : false

알파벳 소문자 만으로 1개 이상
1 : false
12 : false
123 : false
1234 : false
a : true
ab : true
abc : true
abcd : true
A : false
AB : false
ABC : false
ABCD : false

알파벳 대, 소문자 구분없이 1개 이상
1 : false
12 : false
123 : false
1234 : false
a : true
ab : true
abc : true
abcd : true
A : true
AB : true
ABC : true
ABCD : true

알파벳 대, 소문자 구분없이 2,3개
1 : false
12 : false
123 : false
1234 : false
a : false
ab : true
abc : true
abcd : false
A : false
AB : true
ABC : true
ABCD : false

숫자, 알파벳 대, 소문자 구분없이 2,3개
1 : false
12 : true
123 : true
1234 : false
a : false
ab : true
abc : true
abcd : false
A : false
```

```
AB : true
ABC : true
ABCD : false

숫자, 알파벳 대문자가 아닌 2,3개
1 : false
12 : false
123 : false
1234 : false
a : false
ab : true
abc : true
abcd : false
A : false
AB : false
ABC : false
ABCD : false
```

그룹개념 이해하기

```
import java.util.regex.Pattern;

public class RegExpEx24_Example11 {
    public static void main(String[] args) {
        // 그룹 개념 이용하기
        String[] inputArray = "설악산,지리산,금산,설악봉,지리봉,금봉,설악대,지리대,금
대".split(",");
        System.out.print("산 : ");
        for(String str : inputArray) {
            if(Pattern.matches(".*산", str)) System.out.print(str + " ");
        }
        System.out.println();

        System.out.print("지리 : ");
        for(String str : inputArray) {
            // if(Pattern.matches("지리산|지리봉|지리대", str))
System.out.print(str + " ");
            if(Pattern.matches("지리(산|봉|대)", str)) System.out.print(str + "
");
        }
        System.out.println();

        System.out.print("지리/설악 : ");
        for(String str : inputArray) {
            if(Pattern.matches("(지리|설악)(산|봉|대)", str)) System.out.print(str
+ " ");
        }
        System.out.println();
    }
}
```

결과

```
산 : 설악산 지리산 금산
지리 : 지리산 지리봉 지리대
지리/설악 : 설악산 지리산 설악봉 지리봉 설악대 지리대
```

```java
import java.util.Scanner;
import java.util.regex.Pattern;

public class RegExpEx25_Example12 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String menu = "";
        while(true) {
            System.out.print("뭐 먹을래(부타동,에비동,규동,.은 종료)  1개 주문해야
함.");
            menu = sc.nextLine();
            if(Pattern.matches("\\.", menu)) break;
            if(Pattern.matches("부타동|에비동|규동", menu)) {
                System.out.println(menu + "!! 주문입니다.");
            }else {
                System.out.println("없는 메뉴 입니다.");
            }
        }
        System.out.println("----------------------------------");
        while(true) {
            System.out.print("뭐 먹을래(부타동,에비동,규동,.은 종료) 1개 주문해야
함.");
            menu = sc.nextLine();
            if(Pattern.matches("\\.", menu)) break;
            if(Pattern.matches("(부타|에비|규)동", menu)) {
                System.out.println(menu + "!! 주문입니다.");
            }else {
                System.out.println("없는 메뉴 입니다.");
            }
        }
        System.out.println("----------------------------------");
        while(true) {
            System.out.print("뭐 먹을래(부타동,에비동,규동,.은 종료) 2개 이상 주문해야
함.");
            menu = sc.nextLine();
            if(Pattern.matches("\\.", menu)) break;
            if(Pattern.matches("(부타|에비|규)(동)([, ]*(부타|에비|규)(동))+",
menu)) {
                System.out.println(menu + "!! 주문입니다.");
            }else {
                System.out.println("없는 메뉴 입니다.");
            }
        }
        System.out.println("----------------------------------");
        while(true) {
            System.out.print("뭐 먹을래(부타동,에비동,규동,.은 종료) 1개 이상 주문해야
함.");
            menu = sc.nextLine();
            if(Pattern.matches("\\.", menu)) break;
```

```
            if(Pattern.matches("(부타|에비|규)(동)([, ]*(부타|에비|규)(동))*",
menu)) {
                System.out.println(menu + "!! 주문입니다.");
            }else {
                System.out.println("없는 메뉴 입니다.");
            }
        }
        System.out.println("----------------------------------");
        /*
        이 정규 표현식의 두 번째 그룹이 '(동)'이고, 그 이후에 '동'이 반복하여 나올 것이므
로,
        백슬래쉬와 몇 번째 그룹인지를 사용하면 그룹의 문자열을 역참조하여 해당 정규 표현식에
서 사용할 수 있다.
        여기서 주의해야 할 점은, 입력된 문장에서, 앞의 '(부타|에비|규)'에 해당하는 첫 번째
그룹에서 가츠가
        나왔다면, 첫 번째 그룹은 '(부타|에비|규)'의 표현과 일치하는 것이 아니고 '가츠'로
정해진다.
        만약 여기서 '(부타|에비|규)' 그룹을 순서에 포함시키고 싶지 않다면, 즉 캡쳐하고 싶
지 않다면,
        다음과 같이 정규표현식을 작성하면 된다. (?:regex)와 같은 그룹을 non-capturing
group 이라고 한다.
        "(?:부타|에비|규)(동)([, ]*(부타|에비|규)(\\1))*"
        */
        while(true) {
            System.out.print("뭐 먹을래(부타동,에비동,규동,.은 종료) 1개 이상 주문해야
함.");
            menu = sc.nextLine();
            if(Pattern.matches("\\.", menu)) break;
            if(Pattern.matches("(부타|에비|규)(동)([, ]*(부타|에비|규)(\\2))*",
menu)) {
                System.out.println(menu + "!! 주문입니다.");
            }else {
                System.out.println("없는 메뉴 입니다.");
            }
        }
        sc.close();
    }
}
```

결과

```
뭐 먹을래(부타동,에비동,규동,.은 종료)  1개 주문해야 함.규동
규동!! 주문입니다.
뭐 먹을래(부타동,에비동,규동,.은 종료)  1개 주문해야 함.에비동
에비동!! 주문입니다.
뭐 먹을래(부타동,에비동,규동,.은 종료)  1개 주문해야 함.규동 에비동
없는 메뉴 입니다.
뭐 먹을래(부타동,에비동,규동,.은 종료)  1개 주문해야 함..
----------------------------------
뭐 먹을래(부타동,에비동,규동,.은 종료) 1개 주문해야 함.규동
규동!! 주문입니다.
뭐 먹을래(부타동,에비동,규동,.은 종료) 1개 주문해야 함.부타동
부타동!! 주문입니다.
뭐 먹을래(부타동,에비동,규동,.은 종료) 1개 주문해야 함.부타동 규동
없는 메뉴 입니다.
뭐 먹을래(부타동,에비동,규동,.은 종료) 1개 주문해야 함..
----------------------------------
```

```
뭐 먹을래(부타동,에비동,규동,.은 종료) 2개 이상 주문해야 함.부타동
없는 메뉴 입니다.
뭐 먹을래(부타동,에비동,규동,.은 종료) 2개 이상 주문해야 함.부타동 규동
부타동 규동!! 주문입니다.
뭐 먹을래(부타동,에비동,규동,.은 종료) 2개 이상 주문해야 함.규종,에비동
없는 메뉴 입니다.
뭐 먹을래(부타동,에비동,규동,.은 종료) 2개 이상 주문해야 함..
---------------------------------
뭐 먹을래(부타동,에비동,규동,.은 종료) 1개 이상 주문해야 함.규동
규동!! 주문입니다.
뭐 먹을래(부타동,에비동,규동,.은 종료) 1개 이상 주문해야 함.부타동
부타동!! 주문입니다.
뭐 먹을래(부타동,에비동,규동,.은 종료) 1개 이상 주문해야 함.규동,부타동
규동,부타동!! 주문입니다.
뭐 먹을래(부타동,에비동,규동,.은 종료) 1개 이상 주문해야 함..
---------------------------------
뭐 먹을래(부타동,에비동,규동,.은 종료) 1개 이상 주문해야 함.규동
규동!! 주문입니다.
뭐 먹을래(부타동,에비동,규동,.은 종료) 1개 이상 주문해야 함.부타동
부타동!! 주문입니다.
뭐 먹을래(부타동,에비동,규동,.은 종료) 1개 이상 주문해야 함.규동,부타동 에비동
규동,부타동 에비동!! 주문입니다.
뭐 먹을래(부타동,에비동,규동,.은 종료) 1개 이상 주문해야 함..
```

4. 자주 사용되는 정규표현식

4-1. 전화번호 정규표현식

일반 전화번호 정규식 : ^\\d{2,3}-\\d{3,4}-\\d{4}$

0으로 시작되고 1~2자리의 숫자의 지역번호로 시작해야 하므로 "^0\\d{2,3}" : 02, 031, 043 ....

사이에 -가 들어가서 : -

3~4자리의 전화국번이 들어가야 하므로 "\\d{3,4}" : 1234,5678 ...

사이에 -가 들어가서 : -

4자리 숫자의 전화번호로 끝나야 하므로 들어가야 하므로 "\\d{4}$" : 3456, 7890...

완성하면 ""^\\d{2,3}-\\d{3,4}-\\d{4}$""

```java
import java.util.regex.Pattern;

public class RegExpEx26_PhoneNumber1 {
    public static void main(String[] args) {
        // 일반 전화번호 정규식
        String patternStr = "^\\d{2,3}-\\d{3,4}-\\d{4}$";

        String[] phoneNumbers = "02-111-2222,031-1234-1234,1234-2345,02-12-
9870,02-222-33333".split(",");

        for (String tel : phoneNumbers) {
            System.out.print(tel + "은 ");
            if (Pattern.matches(patternStr, tel)) {
                System.out.println("맞는형식");
```

```
            } else {
                System.out.println("잘못된 형식");
            }
        }
    }
}
```

핸드폰 전화번호 정규식 : ^01[016789]-\\d{3,4}-\\d{4}$

01로 시작되고 자리의 숫자의 지역번호로 시작해야 하므로 "^01[016789]" : 010,011,016,017,018,019 등

사이에 -가 들어가서 : -

3~4자리의 전화국번이 들어가야 하므로 "\\d{3,4}" : 1234,5678 ...

사이에 -가 들어가서 : -

4자리 숫자의 전화번호로 끝나야 하므로 들어가야 하므로 "\\d{4}$" : 3456, 7890...

완성하면 "^01[016789]-\\d{3,4}-\\d{4}$"

```
import java.util.regex.Pattern;

public class RegExpEx27_PhoneNumber2 {
    public static void main(String[] args) {
        // 핸드폰 전화번호 정규식
        String patternStr = "^01[016789]-\\d{3,4}-\\d{4}$";

        String[] phoneNumbers = "010-111-2222,110-1234-1234,016-1234-2345,010-
12-9870,02-2252-3333".split(",");

        for (String tel : phoneNumbers) {
            System.out.print(tel + "은 ");
            if (Pattern.matches(patternStr, tel)) {
                System.out.println("맞는형식");
            } else {
                System.out.println("잘못된 형식");
            }
        }
    }
}
```

결과

```
010-111-2222은 맞는형식
110-1234-1234은 잘못된 형식
016-1234-2345은 맞는형식
010-12-9870은 잘못된 형식
02-2252-3333은 잘못된 형식
```

4-2. 주민등록번호 정규 표현식

```
import java.util.regex.Pattern;

public class RegExpEx28_JuminNumber1 {
    // 주민등록번호 정규식
    public static void main(String[] args) {
        // 주민등록번호
        String patternStr = "^\\d{6}-[1234]\\d{6}$";
        System.out.println(Pattern.matches(patternStr, "111111-1111111"));
        System.out.println(Pattern.matches(patternStr, "111111-4111111"));
        System.out.println(Pattern.matches(patternStr, "111111-5111111"));
        System.out.println(Pattern.matches(patternStr, "11111-111111"));

    }
}
```

결과

```
true
true
false
false
```

```
package kr.green.regexp1;

import java.util.regex.Pattern;
/*
주민등록번호 구성은 아래와 같습니다.
예) 123456 - 1234567

1. 첫 6자리
12 : 출생년도 마지막 두자리
34 : 출생 월
56 : 출생 일

2. 마지막 7자리
1 : 성별 (1: 남자, 2: 여자, 3: 2000년 이후 출생자 남자, 4: 2000년 이후 출생자 여자)
2345 : 지역코드
6 : 출생신고지 기준 접수 순번
7 : 검증번호 (앞 12 자리 숫자를 특정 공식에 대입 시 딱 하나의 숫자만 나올 수 있음)
*/
public class RegExpEx29_JuminNumber2 {
    public static void main(String[] args) {
        // 앞 6자리 패턴 만들기
        // 1. 출생년도 : 년도는 00~99까지 가능하다. ===>  ^\\d{2}    또는 ^[0-9]{2}
        // 위 두가지 표현식 모두 동일한 의미 입니다. 0 ~ 9 사이 두자릿수 숫자 허용
        String yearPattern = "^\\d{2}";
        System.out.println("년도");
        System.out.println(Pattern.matches(yearPattern, "00"));
        System.out.println(Pattern.matches(yearPattern, "99"));
        // 2. 출생 월  : 01 ~ 12
        String monthPattern = "(0[1-9]|1[0-2])";
        System.out.println("월");
        System.out.println(Pattern.matches(monthPattern, "00"));
        System.out.println(Pattern.matches(monthPattern, "01"));
```

```
        System.out.println(Pattern.matches(monthPattern, "12"));
        System.out.println(Pattern.matches(monthPattern, "23"));
        // 2. 출생 일  : 01 ~ 31
        String datePattern = "(0[1-9]|[12][0-9]|3[01])";
        System.out.println("일");
        System.out.println(Pattern.matches(datePattern, "00"));
        System.out.println(Pattern.matches(datePattern, "01"));
        System.out.println(Pattern.matches(datePattern, "31"));
        System.out.println(Pattern.matches(datePattern, "32"));
        System.out.println("주민앞자리");
        String juminPattern1 = yearPattern + monthPattern + datePattern;
        System.out.println(Pattern.matches(juminPattern1, "000101"));
        System.out.println(Pattern.matches(juminPattern1, "991231"));
        System.out.println(Pattern.matches(juminPattern1, "990031"));
        System.out.println(Pattern.matches(juminPattern1, "990044"));
        // 뒤 7자리 : 성별은 1234중1개 나머지 6자리숫자는 000000-999999
        String juminPattern2 = "[1-4]\\d{6}";
        System.out.println("뒷자리");
        System.out.println(Pattern.matches(juminPattern2, "0111111"));
        System.out.println(Pattern.matches(juminPattern2, "1111111"));
        System.out.println(Pattern.matches(juminPattern2, "2111111"));
        System.out.println(Pattern.matches(juminPattern2, "3111111"));
        System.out.println(Pattern.matches(juminPattern2, "4111111"));
        System.out.println(Pattern.matches(juminPattern2, "5111111"));

        String juminPattern = juminPattern1 + "-" + juminPattern2;
        System.out.println("주민번호패턴 : " + juminPattern);
        System.out.println(Pattern.matches(juminPattern, "111111-1111111"));
        System.out.println(Pattern.matches(juminPattern, "111111-4111111"));
        System.out.println(Pattern.matches(juminPattern, "111111-5111111"));
        System.out.println(Pattern.matches(juminPattern, "110019-1111111"));
    }
}
```

결과

```
년도
true
true
월
false
true
true
false
일
false
true
true
false
주민앞자리
true
true
false
false
뒷자리
false
true
```

```
true
true
true
false
주민번호패턴 : ^\d{2}(0[1-9]|1[0-2])(0[1-9]|[12][0-9]|3[01])-[1-4]\d{6}
true
true
false
false
```

4-3. Email 정규표현식

```
import java.util.regex.Pattern;
/*
정규표현식으로 이메일 주소 패턴 만들기
xxxxxx@xxxx.xxx
xxxxxx@xxxx.xxx.xx
일반적으로 이메일 주소 형태는 위와 같이 @ 를 기준으로 ID 부와 HOST 부로 나뉩니다.
이 ID 와 HOST 부에 적용할 이메일 주소 패턴의 형태를 정하겠습니다.

ID 부 설정
1. 영문 대소문자 허용
2. 숫자 허용
3. . _ % + -  등 5가지 특문 허용

HOST 부 설정
1. 영문 대소문자 허용
2. 숫자 허용
3. . - 등 2가지 특문 허용
4. 위 값이 최소 1개 이상 표기된 후  .  추가
5. 마침표 뒤에는 영문자 대소문자 만 허용
6. 마침표 뒤의 문자 셋은 최소 2 ~ 최대 6 자리 까지만 허용
*/
public class RegExpEx30_Email {
    public static void main(String[] args) {
        // 영문 대소문자 허용 : a-zA-Z
        // 숫자 허용 : 0-9
        // . _ % + -  등 5가지 특문 허용 : ._%+-
        // 1글자 이상 : +
        String idPattern = "^[a-zA-Z0-9._%+-]+";

        // 영문 대소문자 허용 : a-zA-Z
        // 숫자 허용 : 0-9
        // .  -  등 2가지 특문 허용 : .-
        // 1글자 이상 : +
        // .이나타나고 : \\.
        // 영문자 2~6자 : [a-zA-Z]{2,6}
        String hostPattern = "[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,6}";
        String emailPattern = idPattern + "@" + hostPattern;
        System.out.println("Email정규식 : " + emailPattern);

        String[] emails = "test5004@kimc.com,test"

 +"-5004@kimc.com,test%5004@kimc.com,test+5004@kimc.kr,test5004@kimc.co.kr".spli
t(",");
```

```
        for(String email : emails) {
            System.out.println(email + " : " + Pattern.matches(emailPattern,
email));
        }
        System.out.println();
        // 문제
        System.out.println("test+5004@kimc..kr : " +
Pattern.matches(emailPattern, "test+5004@kimc..kr"));
        System.out.println();

        // 두번째
        emailPattern = "^[0-9a-zA-Z]([._%+-]?[0-9a-zA-Z])*@[0-9a-zA-Z]([-_.]?[0-
9a-zA-Z])*.[a-zA-Z]{2,6}$";
        for(String email : emails) {
            System.out.println(email + " : " + Pattern.matches(emailPattern,
email));
        }
        System.out.println();
        System.out.println("test+5004@kimc..kr : " +
Pattern.matches(emailPattern, "test+5004@kimc..kr"));
        // 위의 내용도 완벽하지 않습니다. 도메인 형식이 아주 다양하기 때문입니다. 한글도메
인도 있습니다.
    }
}
```

결과

```
Email정규식 : ^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}
test5004@kimc.com : true
test-5004@kimc.com : true
test%5004@kimc.com : true
test+5004@kimc.kr : true
test5004@kimc.co.kr : true

test+5004@kimc..kr : true

test5004@kimc.com : true
test-5004@kimc.com : true
test%5004@kimc.com : true
test+5004@kimc.kr : true
test5004@kimc.co.kr : true

test+5004@kimc..kr : false
```

4-4. 비밀번호 정규 표현식

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
/*
비밀번호 : 숫자,영소문자, 영대문자, 특수문자 포함
*/
public class RegExpEx31_Password {
    public static void main(String[] args) {
```

```
        System.out.println(passwordCheck1("123") + " : " +
passwordCheck2("123"));
        System.out.println(passwordCheck1("!@#") + " : " +
passwordCheck2("!@#"));
        System.out.println(passwordCheck1("abcABC") + " : " +
passwordCheck2("abcABC"));
        System.out.println(passwordCheck1("1A!") + " : " +
passwordCheck2("1A!"));
        System.out.println(passwordCheck1("aA!") + " : " +
passwordCheck2("aA!"));
        System.out.println(passwordCheck1("1aA") + " : " +
passwordCheck2("1aA"));
        System.out.println(passwordCheck1("1a!") + " : " +
passwordCheck2("1a!"));
        System.out.println(passwordCheck1("1aA!") + " : " +
passwordCheck2("1aA!"));
        System.out.println(passwordCheck1("12abAB!@") + " : " +
passwordCheck2("12abAB!@"));
    }

    // 정규표현식을 사용하지 않는 경우
    public static boolean passwordCheck1(String password) {
        String specialChar = "~`!@#$%^&*()_-+=\\|![]{};:'\"?/>.<,";
        String numStr = "0123456789";
        String alphabetUpperStr = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        String alphabetLowerStr = "abcdefghijklmnopqrstuvwxyz";
        boolean isSpecialChar = false, isNum = false, isAlphabetUpperStr =
false, isAlphabetLowerStr = false;
        for (String s : password.split("")) {
            if (specialChar.contains(s)) isSpecialChar = true;
            if (numStr.contains(s)) isNum = true;
            if (alphabetUpperStr.contains(s)) isAlphabetUpperStr = true;
            if (alphabetLowerStr.contains(s)) isAlphabetLowerStr = true;
        }
        // System.out.println("\n" + password);
        // System.out.println(isSpecialChar ? "특수문자 포함" : "특수문자 미포함");
        // System.out.println(isNum ? "숫자 포함" : "숫자 미포함");
        // System.out.println(isAlphabetLowerStr ? "영소문자 포함" : "영소문자 미포
함");
        // System.out.println(isAlphabetUpperStr ? "영대문자 포함" : "영대문자 미포
함");
        return isSpecialChar && isNum && isAlphabetLowerStr &&
isAlphabetUpperStr;
    }

    // 정규표현식을 사용는 경우
    public static boolean passwordCheck2(String password) {
        // 최소 3자리에 대문자 1자리 소문자 1자리 숫자 1자리 특수문자 1자리 포함
        // (?=.*[a-z]) : 소문자 최소 1개
        // (?=.*[A-Z]) : 대문자 최소 1개
        // (?=.*\\d) : 숫자 최소 1개
        // (?=.*[$@$!%*?&]) : 특수문자 최소 1개
        // .{4,} : 선행문자 최소 4글자 이상
        Pattern pattern = Pattern.compile("^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d)(?=.*
[$@$!%*?&]).{4,}");
        Matcher matcher = pattern.matcher(password);
        return matcher.matches();
    }
```

```
}
```

결과

```
false : false
false : false
false : false
false : false
false : false
false : false
false : false
true : true
true : true
```

4-5. IP주소 정규식 패턴

```
import java.util.regex.Pattern;
/*
정규표현식으로 IP 주소 패턴 만들기
IP 주소는 다음과 같은 형식을 취하고 있습니다.
xxx . xxx . xxx . xxx
이는 우리가 일반적으로 많이 사용하고 있는 IP Version 4 (IPv4) 주소 체계입니다.
각 옥텟의 xxx 값은 0 ~ 255 까지의 숫자가 올 수 있습니다.
즉, 0.0.0.0 부터 255.255.255.25 까지의 주소가 가능합니다.
*/
public class RegExpEx32_IPAddress {
    public static void main(String[] args) {
        // 첫번째 자리는 0 또는 1만 허용하고 0 또는 1이 존재 하거나 존재하지 않아도 됩니
다.  : [01]? :
        // 두번째 자리는 숫자를 허용하며 값이 존재하거나 하지 않아도 됩니다. : \\d?
        // 세번째 자리는 숫자를 허용하며 무조건 존재 해야 합니다.  : \\d
        String ipPattern1 = "[01]?\\d?\\d"; // 0 ~ 199
        // 200~ 255처리
        String ipPattern2 = "2[0-4]\\d"; // 200 ~249
        String ipPattern3 = "25[0-5]"; // 250 ~255

        // 각각의 표현식을 합쳐 묶어보면
        String ipPattern4 = "(" + ipPattern1 + "|" + ipPattern2 + "|" +
ipPattern3 + ")";
        System.out.println(ipPattern4);

        // 위의 패턴이 .으로 구분하여 4자리 이므로
        String ipPattern = "^" + ipPattern4 + "\\." + ipPattern4 + "\\."+
ipPattern4 + "\\."+ ipPattern4 + "$";
        System.out.println(ipPattern);

        String[] ips =
"0.0.0.0,255.255.255.255,127.0.0.1,210.0.256.0".split(",");
        for(String ip : ips) {
            System.out.println(ip + " : " + Pattern.matches(ipPattern, ip));
        }
    }
}
```

결과

```
([01]?\d?\d|2[0-4]\d|25[0-5])
^([01]?\d?\d|2[0-4]\d|25[0-5])\.([01]?\d?\d|2[0-4]\d|25[0-5])\.([01]?\d?\d|2[0-4]\d|25[0-5])\.([01]?\d?\d|2[0-4]\d|25[0-5])$
0.0.0.0 : true
255.255.255.255 : true
127.0.0.1 : true
210.0.256.0 : false
```

4-6. Lotto번호 정규식 패턴

```
import java.util.regex.Pattern;
/*
로또번호 패턴 만들기
1~45번까지
*/
public class RegExpEx33_Lotto {
    public static void main(String[] args) {

        String lottoPattern1 = "^[1-9]{1}$"; // 1~91자리
        String lottoPattern2 = "^[1-3]{1}[0-9]{1}$"; // 10~39
        String lottoPattern3 = "^4{1}[0-5]{1}$"; // 40~45
        String lottoPattern = lottoPattern1 + "|" + lottoPattern2 + "|" +
lottoPattern3;
        System.out.println("로또패턴 : " + lottoPattern);
        System.out.println(Pattern.matches(lottoPattern, "0"));
        System.out.println(Pattern.matches(lottoPattern, "1"));
        System.out.println(Pattern.matches(lottoPattern, "45"));
        System.out.println(Pattern.matches(lottoPattern, "46"));
    }
}
```

결과

```
로또패턴 : ^[1-9]{1}$|^[1-3]{1}[0-9]{1}$|^4{1}[0-5]{1}$
false
true
true
false
```

4-7. HTML 태그 제거하기

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
/*
HTML태그 없애기
*/
public class RegExpEx34_RemoveHTMLTag {
    public static void main(String[] args) {
```

```
        String inputStr = "<h1>나는<img ><img src='xxx.png' width='100'/><span>
<b><q>제목</q></b></span>입니다.</h1>";

        try {
            System.out.println(removeTag(inputStr));
            System.out.println(removeTag(inputStr,"q"));
            System.out.println(removeTag(inputStr,"img"));
        } catch (Exception e) {
            e.printStackTrace();
        }

        Pattern pattern = Pattern.compile("<(/)?([a-zA-Z0-9]*)(\\s[a-zA-Z]*=
[^>]*)?(\\s)*(/)?>");
        Matcher matcher = pattern.matcher(inputStr);
        while(matcher.find()) {
            System.out.println(matcher.group());
        }
    }
    public static String removeTag(String html) throws Exception {
        return html.replaceAll("<(/)?([a-zA-Z0-9]*)(\\s[a-zA-Z]*=[^>]*)?(\\s)*
(/)?>", "");
    }
    public static String removeTag(String html,String tag) throws Exception {
        return html.replaceAll("<(/)?("+tag+")*(\\s[a-zA-Z]*=[^>]*)?(\\s)*(/)?
>", "");
    }
}
```

결과

```
나는제목입니다.
<h1>나는<img ><img src='xxx.png' width='100'/><span><b>제목</b></span>입니다.</h1>
<h1>나는<span><b><q>제목</q></b></span>입니다.</h1>
<h1>
<img >
<img src='xxx.png' width='100'/>
<span>
<b>
<q>
</q>
</b>
</span>
</h1>
```