13장. 문자열

1. Character

Wrapper class 중에 하나이며 한 문자를 관리하는 클래스이다.
Character 클래스의 객체는 'A', 'z'등 단일 문자의 값을 유지하고 한 번 문자가 할당되면 변경할 수 없는 특징을 가진다. 기본형 변수 char 의 차이는 Character 객체는 대소 문자를 판별하는 메소드, 객체끼리 비교하는 방법 등 편리한 메소드를 제공한다는 점이다.

Charater 의 주요 생성자 와 메소드는 다음과 같다.

생성자	설명
Charater (char)	Character 클래스 유일한 생성자로 인수로 주어진 char 값을 나타내는 값을 객체에 보관한다.

리턴값	메소드	설명
int	compareTo(Character)	메소드를 호출하는 개체와 인수의 개체가 보유하고 값을 비교합니다. 값이 동일한 경우는 0 을 돌려줍니다. 메서드 호출 개체가 더 큰 값을 보유하고 있는 경우는 정의 정수를 돌려줍니다. 인수에 주어진 객체가 더 작은 값을 보관 유지하고 있는 경우는 음의 정수를 반환합니다.
boolean	equals(Object)	메소드를 호출하는 개체와 인수의 객체 비교합니다. 두 객체의 값이 동일한 경우에 true 를 돌려줍니다.
String	toString( )	Character 객체를 문자열로 변환하는 방법입니다. 인수의 객체는 길이 1 개체의 값을 유지 한 문자열로 변환됩니다.
char	charValue( )	Character 개체가 보유하고 값을 char 형으로 변환하여 반환합니다.

리턴값	클래스 메소드	설명
static boolean static boolean	isUpperCase(char) isLowerCase(char)	인수 char 형의 값이 대문자 (isUpperCase) 또는 문자 (isLowerCase) 여부를 결정합니다. 맞으면 true, 그렇지 않으면 false 를 반환합니다.
static char static char	toUpperCase(char) toLowerCase(char)	인수 char 형의 값을 대문자 (toUpperCase) 내지는 문자 (toLowerCase)을 반환합니다. 반환 값은 char 형입니다.
static	isLetter(char)	인수 char 형의 값이 문자 (isLetter) 또는 숫자 (isDigit) 또는

boolean static boolean static boolean	isDigit(char) isLetterOrDigit(char)	영문자 또는 숫자 (isLetterOrDigit) 여부를 결정합니다. 맞으면 true, 그렇지 않으면 fals 를 반환합니다.
static boolean	isWhitespace(char)	인수 char 형의 값이 Java 플랫폼의 사양에 따른 공백 문자 여부를 판단합니다. 맞으면 true 를 그렇지 않으면 false 를 반환합니다.
static boolean	isSpaceChar(char)	인수 char 형의 값이 Unicode 사양에 따른 공백 문자 여부를 판단합니다. 맞으면 true 를 그렇지 않으면 false 를 반환합니다.
static boolean	isLetterOrDigit (char ch)	지정된 문자가 범용 문자 또는 숫자인가를 판단

프로그램예를 살펴 보자.

```
public class TestCharacter {
	public static void main(String[] args) {
		Character objChar1, objChar2;
		objChar1 = new Character('a');
		objChar2 = new Character('오');

		System.out.println("objChar1 :" + objChar1.charValue());
		System.out.println("objChar2 :" + objChar2.charValue());

		boolean bln = objChar1.equals(objChar2); -→ 객체 값이 같은지 비교
		System.out.println("objChar1 = objChar2?" + bln);

		// static 메소드 사용
		System.out.println("---- 정적 메서드의 이용 ----------");
		char ch = 'a';
	boolean bln1 = Character.isLetterOrDigit(ch); -→ 'a'가 범용문자 ,숫자 여부
		boolean bln2 = Character.isDigit(ch);   -> 숫자인지 비교 여부

		System.out.println("ch :" + ch);

		if (bln1 == false) {
			System.out.println("범용 문자도 숫자도 없습니다.");
```

```
                } else if (bln2 == false) {
                        System.out.println("범용 문자입니다.");

                        boolean bln3 = Character.isUpperCase(ch); → 대문자인지 여부
                        if (bln3 == false) { -→ 소문자라면
                                char chU = Character.toUpperCase(ch); →대문자 변환
                                System.out.println("대문자로합니다 :" + chU);
                        } else {
                                System.out.println("대문자입니다.");
                        }

                } else {
                        System.out.println("숫자입니다.");
                }
        }

}
```

[실행결과]
objChar1 :a
objChar2 :오
objChar1 = objChar2?false
---- 정적 메서드의 이용 ----------
ch :a
범용 문자입니다.
대문자로합니다 :A

2. String

Character 클래스가 단일 문자 만 취급하지 못했던 반면 String 클래스는 여러 문자 (문자열)를 처리 할 수 있다.
그러나 Character 클래스와 같이, 일단 문자열이 개체에 할당되면 그 문자열을 수정할 수 없다..
String 클래스는 다른 개체간에 동일한 문자열 공유 할 수 있는 등 효율적인 구조로 되어 있다.
메모리의 유효 이용을 감안할 때 할당 된 문자열의 변경이 없으면 우선적으로 String 클래스를 사용하도록 한다

String 객체를 생성하는 주요 수단으로 다음의 두 가지 방법이 있다.

```
String 객체 명 = "문자열";
String 객체 이름 = new String ( "문자열");
```

new 연산자를 사용하지 상단 방법은 new 연산자를 사용하여 하단의 방법에 비해 메모리 사용 관계 상 효율이 좋다. 상단의 방법은 컴파일시 지정된 문자열이 메모리 영역에 만들어지는 반면, 하단의 방법은 컴파일시 지정된 문자열이 메모리 영역에 만들어지는 외에 new String () 실행시도 같은 지정된 문자열을 포함하는 인스턴스가 만들어진다. 이것은 여분의 메모리 영역이며, String 클래스에 제공되는 편리한 생성자를 사용하는 경우를 제외하고 new 연산자를 사용할 필요가 없다.

주요 생성자와 메소드는 다음과 같다.

[생성자]

String ()	새롭게 생성 된 String 객체를 초기화하여 빈 문자열을 가진다.
String (byte [] bytes)	플랫폼의 기본 문자 인코딩을 사용하여 지정된 바이트 배열을 변환함으로써 새로운 String 를 가진다.
String (char [] value)	새로운 String 를 할당 해, 이것이 문자 배열 인수에 현재 포함 된 문자열을 나타낸다.
String (String value)	새롭게 생성 된 String 오브젝트를 초기화 해, 인수와 같은 문자열을 나타낸다.
String (StringBuffer buffer)	StringBuffer 형의 인수에 현재 포함되어있는 캐릭터 라인을 가지는 새로운 캐릭터 라인을 구축한다.

```
class TestStringInst {
    public static void main (String args []) {
            String objSt1 = new String ( "안녕하세요!");
            String objSt2 = "Hello!";
            System.out.println (objSt1);
            System.out.println (objSt2);
    }
}
```

String 클래스의 객체는 기본 데이터 형의 변수와 같은 형식으로 인스턴스화 하도록 정해져 있다.

```
안녕하세요!
Hello!
```

[주요 메소드]

반환 형식	메소드	개요
char	charAt (int index)	지정된 인덱스 위치에 있는 문자를 돌려줍니다.
boolean	equals (Object anObject)	이 캐릭터 라인과 지정된 오브젝트를 비교합니다.
byte []	getBytes ()	String 를 플랫폼의 디폴트의 문자 인코딩에 따라 바이트로 변환 해, 결과를 새로운 바이트 배열에 저장합니다.
void	getChars (int srcBegin, int srcEnd, char [] dst, int dstBegin)	이 캐릭터 라인으로부터, 카피 처의 문자 배열에 문자를 카피합니다.
int	indexOf (int ch)	이 캐릭터 라인 내에서, 지정된 문자가 최초로 출현하는 위치의 인덱스를 돌려줍니다.
int	indexOf (String str)	이 캐릭터 라인 내에서, 지정된 부분 캐릭터 라인이 최초로 출현하는 위치의 인덱스를 돌려줍니다.
int	length ()	이 문자열의 길이를 반환합니다.
String	replace (char oldChar, char newChar)	이 캐릭터 라인 내에있는 모든 oldChar 를 newChar 에 치환 한 결과 생성되는 새로운 캐릭터 라인을 돌려줍니다.
String	substring (int beginIndex, int endIndex)	이 캐릭터 라인의 부분 캐릭터 라인 인 새로운 캐릭터 라인을 돌려줍니다.
char []	toCharArray ()	이 문자열을 새로운 문자 배열로 변환합니다.
String	toLowerCase ()	Locale.getDefault 에 의해 돌려 주어지는 디폴트 로케일의 규칙을 사용해,이 String 내의 모든 문자를 소문자로 변환합니다.

String	toString ()	이 오브젝트 (벌써 캐릭터 라인이다) 자신이 돌려 주어집니다.
String	toUpperCase ()	Locale.getDefault 에 의해 돌려 주어지는 디폴트 로케일의 규칙을 사용해,이 String 내의 모든 문자를 대문자로 변환합니다.
String	trim ()	이 문자열의 끝에서 공백을 제거합니다.

그 밖에도 정적 메소드 (static)로 기본 데이터 형을 인수에 있는 valueOf ()메서드 등이 있습니다. 이 메소드는 인수의 기본 데이터 형의 변수와 상수를 문자열로 변화한다. 문자열 비교의 메소드 equals ()의 사용법을 기억해두면 좋을 것입니다. 문자열 비교는 비교 연산자==아닌 객체 비교 메소드 equals ()를 사용한다.

열을 다룰 때 깨진 관련된 오류가 자주 보고되고 있습니다. 먼저 Java 에서는 UNICODE 를 사용하고 있음을 재확인하고, UNICODE 에서 문자 코드가 어떻게 되어 있는지를 확인합니다

```
class StringDump {
    public static void main (String [] args) {
            String str = "16 진법으로 바꿉니다. ";
            System.out.println (str);
            char [] buf = str.toCharArray ();
            for (int i = 0; i <buf.length; i + +) {
                System.out.print (Integer.toString (buf [i], 16) + "");
            }
    }
}
```

```
16 진법으로 바꿉니다.
31 36 c9c4 bc95 c73c b85c 20 bc14 afc9 b2c8 b2e4 2e
```

UNICODE 는 하위 7 비트 ASCII (Latin-1)와 호환성을 유지하도록 설계되어 있기 때문에 공백 류 문자, 알파벳, 숫자는 ASCII 와 동일하다. 특히, 줄 바꿈, 공백 등의 공백 류 문자 내용은 시스템 의존성이 있기 때문에 인식 해두면 좋을 것이다.

개행 코드 호환성 http://www.unicode.org/reports/tr13/tr13-9.html 참조

	Unicode	ASCII	EBCDIC	
CR	000D	0D	0D	0D
LF	000A	0A	25	15
CRLF	000D, 000A	0D, 0A	0D 25	0D 15

NEL	0085	85	15	25
VT	000B	0B	0B	0B
FF	000C	0C	0C	0C
LS	2028	n / a	n / a	n / a
PS	2029	n / a	n / a	n / a

EBCDIC 는 IBM 호환 메인 프레임 기계 (호스트 계라고도 함)에서 사용되는 문자 인코딩 방식으로 엔터프라이즈 표준입니다.

3. StringBuffer

문자열은 String 클래스의 객체로 유지하지만, 가변의 경우는 StringBuffer 를 사용 한다. StringBuffer 클래스는 String 클래스와 달리 한 번 개체의 값을 할당 한 후에도 그 값의 변경이 가능하나 그러나 값이 불변 인 String 클래스와 비교하면 값이 동일한 경우에도 메모리에 저장되어있는 값을 공유 할 수 없는 단점을 가진다. 또한 초기 용량을 지정하지 않고 StringBuffer 객체를 생성 한 경우 값의 변화를 예측하고 할당 된 값에 16 자를 더한 값이 메모리 영역에 확보되어 버리는 특징이 있다.

생성자	설명
StringBuffer ()	메모리 할당 영역이 16 문자 인 빈 문자열 버퍼를 생성
StringBuffer (int)	지정된 문자의 메모리 할당 공간을 가진 빈 문자열 버퍼를 생성
StringBuffer (String)	지정된 String 에 초기화 된 캐릭터 라인 버퍼를 생성 메모리 할당 영역은 지정된 String 의 길이에 16 자를 더한 공간을 확보

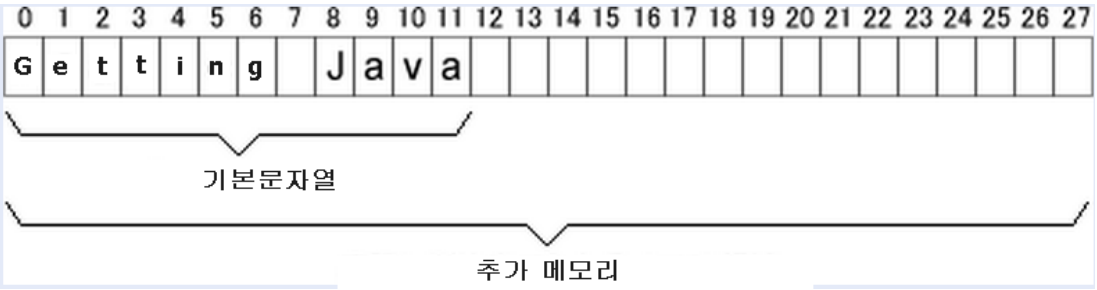
생성자를 통해 객체와 메모리를 살펴 보자.

```
public class TestStringBuffer {

        public static void main(String[] args) {
                StringBuffer aaa = new StringBuffer ( "Getting Java");
                System.out.println(aaa.capacity());
                System.out.println(aaa.length()); -→ 문자의 길이
        }

}
```

[실행결과]

28

12



인덱스가 0 부터 시작하여 메모리에 관리된다.

StringBuffer 의 주요 메소드는 다음과 같다.

반환형	메소드	설명
int	length ()	문자의 길이를 리턴
int	capacity ()	문자열 버퍼 수 (메모리 할당 영역)을 리턴
char	charAt (int)	지정된 인덱스 번호 위치에 있는 문자를 리턴
String	substring (int)	지정된 인덱스 번호 위치에서 끝까지의 문자열을 리턴, 두 번째 인수가 지정되는 경우, 제 1 인수에서 두 번째 인수까지의 문자열을 반환
String	substring (int, int)	
StringBuffer	append (boolean) append (char) append (char []) append (char [], int, int) append (double) append (float) append (int) append (long) append (Object) append (String)	전화 StringBuffer 객체의 끝에 인수로 지정된 데이터를 추가. 데이터는 문자열로 변환되고 나서 추가. char 형 배열의 두번째 인자는 배열의 오프셋을 제 3 인수는 오프셋 길이를 나타냄
StringBuffer	delete (int, int)	전화 StringBuffer 객체의 문자열을 인수로 지정된 범위에서 삭제
StringBuffer	insert (int, boolean) insert (int, char)	전화 StringBuffer 객체의 문자열에 두 번째 인수로 지정된 데이터를 삽입. 제 1 인수로

	insert (int, char []) insert (int, char [], int, int) insert (int, double) insert (int, float) insert (int, int) insert (int, long) insert (int, Object) insert (int, String)	삽입 위치를 지정 지정된 인덱스 번호 앞에 데이터가 삽입, 선두에 문자를 삽입하려면 0 을 지정. 데이터 삽입 될 때 문자열로 변환 된 후 삽입. char 형 배열의 제 3 인수는 배열의 오프셋을 제 4 인수는 오프셋 길이를 나타낸다.
StringBuffer	replace (int, int, String)	StringBuffer 객체의 문자열을 인수로 지정된 범위에서 문자열로 바꿈
void	setLength (int)	문자열 버퍼 수 (메모리 할당 영역)을 설정
StringBuffer	reverse ()	StringBuffer 객체의 문자열의 순서를 역순으로 리턴.

append () 와 insert () 를 이용하여 프로그램을 구현해 보자

```
public class TestStringBuffer02 {

	public static void main(String[] args) {
		// 인스턴스화
		StringBuffer sb1 = new StringBuffer ( "StringBuffer");
				StringBuffer sb2 = new StringBuffer ( "Object");
				System.out.println ( "obj1 :"+ sb1);
				System.out.println ( "obj2 :"+ sb2);

				// append ()
				System.out.println ( "-------- append () --------");
				System.out.println ( "char :"+ sb1.append ( '-') );
			System.out.println ( "String :"+ sb1.append ( "문자열") );
				System.out.println ( "Object :"+ sb1.append (sb2) );

				System.out.println ( "obj1 :"+ sb1);

			// insert ()
			System.out.println ( "-------- insert () --------");
				int i = sb1.length () ;
			System.out.println ( "0, int :"+ sb1.insert (0, i) );
```

```
                System.out.println ( "2, char :"+ sb1.insert (2, '-') );

                System.out.println ( "obj1 :"+ sb1);

        }

}
```

[실행결과]

```
char :StringBuffer-
String :StringBuffer-문자열
Object :StringBuffer-문자열Object
obj1 :StringBuffer-문자열Object
-------- insert () --------
0, int :22 StringBuffer-문자열Object
2, char :22-StringBuffer-문자열Object
obj1 :22-StringBuffer-문자열Object
```

append () 는 char 형, String 형 객체를 추가한다. 이어 length () 메소드 문자열 버퍼의 길이를 int 형으로 리턴한다. insert () 는 int 형, char 형을 삽입한다 삽입 위치를 나타내는 오프셋 값은 글자 위치가 0 에서 지시 한 위치의 직전에 삽입된다. 즉, 0 을 지정하면 처음에 삽입된다.

4. StringBulider

StringBuilder 클래스는 기능(메소드)은 StringBuffer 클래스와 똑같다
만든 이유는 스레드 안전과 성능에 관련되 있다.

StringBuffer 클래스의 모든 메서드는 synchronized 로 되어 동기화가 되어있다. 따라서 여러 스레드에서 사용되는 경우에도 안전하게 사용할 수 있다.

한편 동기화에는 비용이 소요된다. 간단하게 말하면 메소드를 synchronized 하면 기능이 늦어져 버린다. 그래서 등장한 것이 스레드로부터 안전하지 않은 StringBuilder 클래스이다.
즉 동기화를 사용하지 않을 때는 StringBuilder 가 훨씬 효율적이다. 스레드에 안전 여부에 따라 StringBuilder 의 선택 여부를 결정하면 된다.

[java.lang.StringBuilder 클래스의 주요 메소드]

리턴형	메소드	설명
StringBuilder	append (String s)	끝에 s 를 추가
StringBuilder	insert (int x, String s)	a 번째로 s 를 삽입
StringBuilder	replace (int a, int b, String s)	a 번째 ~ b 번째 문자를 s 로 대체
void	setCharAt (int a, char s)	a 번째 문자를 s 로 대체
StringBuilder	delete (int a, int b)	a 번째 ~ b 번째 문자를 삭제
StringBuilder	deleteCharAt (int a)	a 번째 문자를 삭제
char	charAt (int a)	a 번째 문자를 char 형으로 얻을
int	indexOf (String s)	s 가 가장 먼저 출현 인덱스 번호를 반환 (없으면 -1 을 반환)
int	indexOf (String s, int a)	s 이 a 번째부터 가장 먼저 출현 인덱스 번호를 반환 (없으면 -1 을 반환)
int	lastIndexOf (String s)	s 가 마지막에 출현하는 인덱스 번호를 반환 (없으면 -1 을 반환)
int	lastIndexOf (String s, int a)	s 이 a 번째보다 앞에서 가장 먼저 출현 인덱스 번호를 반환 (없으면 -1 을 반환)

[java.lang.StringBuilder 클래스의 주요 메소드]

리턴형	메소드	설명
int	length ()	문자열의 길이 (문자 수)를 반환
StringBuilder	reverse ()	문자열의 순서를 역순으로 리턴
String	substring (int a)	a 번째 이상을 String 형으로 반환
String	substring (int a, int b)	a 번째 ~ b 번째를 String 형으로 반환
void	toString ()	String 형으로 반환

5. 문자열 분할

>> StringTokenizer
데이터가 관리되는 프로그램에서 리턴 하는 값이 여러 형태로 관리 되기 때문에 선언된 문자열을 분할해서 사용하고 할 때가 있다.
주소록파일도 텍스트로 넘겨와 "홍길동 서울 010-000-000"으로 공백 또는 콜론등으로 관리되거나 데이터 베이스에서 사번, 주민번호등 복합 키워드가 하나의 스트링으로 넘겨오는 값들을 기준으로 분할하고자 할 때 사용된다

StringTokenizer 클래스, String 의 split 메서드 또는 java.util.regex 패키지 (정규 표현식) 을 이용해서 문자열을 분할 할 수 있다. 그 중 StringTokenizer 클래스는 문자열을 지정된 구분 기호로 분리 할 수 있는 클래스로. java.util 팩키지에 있는 클래스이다. 특히 CSV 데이터를 처리할 때 등 매우 편리하다

생성자	설명
StringTokenizer (String)	인수로 지정한 문자열을 분할하는 객체를 생성합니다. 구분자는 "₩ t ₩ n ₩ r ₩ f"(공백, 탭, 개행 문자, 구간 용지 공급 문자)가 사용됩니다.

StringTokenizer (String, String)	첫 번째 인수에 지정된 문자열을 두 번째 인수에 지정된 구분 기호로 분리하는 객체를 생성합니다.
StringTokenizer (String, String, boolean)	첫 번째 인수에 지정된 문자열을 두 번째 인수에 지정된 구분 기호로 분리하는 객체를 생성합니다. 제 3 인수에는 구분 기호를 토큰 (분할 된 문자열)으로 간주되는지 지정합니다. true 를 지정하면 단락 문자도 토큰으로 간주 출력됩니다.

다음과 같이 선언될 수 있다.

```
String str1 = new String ( "java Java JAVA"); // 인수에 지정한 문자열을 기본 구분 문자로
StringTokenizer st1 = new StringTokenizer (str1); // 분할 개체 st1 를 생성합니다.

String str2 = new String ( "java, Java, JAVA");// 인수에 지정된 문자열을 인수로 지정한 구분자 ,"로
StringTokenizer st2 = new StringTokenizer (str2 ""); // 분할 개체 st2 를 생성합니다.
```

주요 메소드는 다음과 같다.

리턴형	메소드	설명
int	countTokens ()	분할 수 있는 토큰의 개수를 반환
boolean	hasMoreTokens ()	다음의 토큰이 있는지 여부를 반환
String	nextToken ()	다음의 토큰을 반환

다음 프로그램을 살펴 보자

```
import java.util.StringTokenizer;

public class TestStringTokenizer {

        public static void main(String[] args) {
                // 쉼표로 구분 된 문자열
        String val = "AAA, 111, BBB, 222";
```

```
        // StringTokenizer 객체의 생성
        StringTokenizer st = new StringTokenizer (val, "" );    // 구분자도 동시에 지정

        // 분할 한 문자를 화면 출력
        while (st.hasMoreTokens ()) {
            System.out.println (st.nextToken ());
        }

        }

}
```

[실행결과]
AAA, 111, BBB, 222

>> java.util.regex

정규식은 문자, 기호를 이용하여 특정 문자 패턴을 표현하는 것을 말한다. 문자열이 패턴과 일치하는지 확인하거나 문자열의 문자 패턴과 일치하는 부분을 변경할 경우에 사용한다.

예를 들면 정규식의 표현에서 정규식의 *는 앞의 문자가 0 번 이상 표현되는 것을 말한다 즉, . a * b 는 aab 와 aaaaaab 문자 패턴을 나타낸다. 정규식의 []는 [] 안의 모든 문자가 포함 된 것을 나타낸다 . a [xyz] a 는 axa 과 aya 문자 패턴을 나타낸다.
다음은 정규식에 사용되는 주요 패턴이다 이러한 패턴을 조합하여 정규식을 만든다.

패턴	의미	예		
		패턴	문자열	결과
^	시작 부분과 일치하는 패턴	"^ ab"	"abcd" "cdab"	true false
$	끝에 일치하는 패턴	"ab $"	"abcd" "cdab"	false true

패턴	의미	예		
		패턴	문자열	결과
.	임의의 단일 문자와 일치하는 패턴	"ac"	"abc" "abbc" "ac"	true false false
?	앞의 문자가 0 개 또는 1 개에 일치하는 패턴	"^ A? $"	"" "A" "AA" "B"	true true false false
*	앞의 문자가 0 개 이상 일치하는 패턴	"^ A * $"	"" "A" "AA" "B"	true true true false
+	앞의 문자가 1 개 이상 일치하는 패턴	"^ A + $"	"" "A" "AA" "B"	false true true false
₩ d	반각 숫자 (0 ~ 9)에 맞는 ※ [0-9]와 같은 패턴	"₩₩ d"	"0123" "0123""abc"	true false false
₩ D	반각 숫자 (0 ~ 9) 이외에 맞는 패턴 ※ [^ 0-9]와 같습니다	"₩₩ D"	"0123" "0123""abc"	false true true
₩ w	영문과 숫자 (0 ~ 9, a ~ z, A ~ Z _)에 맞는 패턴	"₩₩ w"	"012_AbC" "12Ab"	true false

패턴	의미	예		
		패턴	문자열	결과
	※ [0-9a-zA-Z_]와 같은		"# $ % &"	false
₩ W	영문과 숫자 (0 ~ 9, a ~ z, A ~ Z _) 이외에 맞는 패턴 ※ [^ 0-9a-zA-Z_]와 같은 패턴	"₩₩ W"	"012_AbC" "12Ab" "# $ % &"	false true true
[]	문자 중 하나와 일치하는 패턴	"[ABC]"	"A" "B" "D"	true true false
()	하나의 그룹으로 취급 패턴	"(ABC)"	"ABC" "CBA"	true false
{n}	직전의 문자에 n 번 일치하는 패턴	"^ A {3} $"	"AA" "AAA" "AAAA"	false true false
{n,}	직전의 문자에 n 번 이상 일치하는 패턴	"^ A {3} $"	"AA" "AAA" "AAAA"	false true true
{n, m}	직전의 문자에 n 회 이상 m 회 이하 일치하는 패턴	"^ A {3,4} $"	"AA" "AAA" "AAAA" "AAAAA"	false true true false
|	문자열 중 하나와 일치하는 패턴	"ABC | DEF"	"ABC" "DEF" "CBA"	true true false

패턴	의미	예		
		패턴	문자열	결과
			"DE"	false
-	범위를 지정 패턴	"[3-7]"	"2" "3" "7" "8"	false true true false
^	부정 ([]에서 사용) 패턴	"[^ AB]"	"A" "B" "C"	false false true
&&	양쪽 두개의 패턴이 모두 충족되는 패턴	"[0-9 && [^ 4]"	"3" "4" "5"	true false true

※ 위 정규식에서 사용되는 문자를 패턴으로 사용하려면 문자 앞에 「₩」를 선언한다.

java.util.regex 패키지는 Pattern 클래스, Matcher 클래스로 구성되어 있다.
Pattern 클래스에서는 정규식을 컴파일하고 정규 표현식 처리를 할 Matcher 클래스의 객체를 생성한다
Matcher 클래스에서는 정규식 처리 (문자열이 정규 표현식하거나 정규 표현식에 매치 한 문자열 대체 등)을 실시하는 메소드가 준비되어 있다.

"Pattern 클래스" compile 메소드 인수에 정규식을 지정하고 정규 표현식을 컴파일한다.

matcher 메소드 인수에 정규 표현식 처리의 대상이되는 문자열을 지정하고 Matcher 클래스의 객체를 생성한다. split 메소드 인수에 지정된 문자열을 정규식에 따라 분할하고 분할 된 문자열을 배열로 리턴한다.

"Matcher 클래스" replaceAll 메소드 정규식과 일치하는 문자열을 인수로 지정한 문자열에 대체한다. matches 메소드 문자열이 정규식과 일치하는지 확인한다. find 메소드 문자열이 정규식과 일치하거나 순서대로 확인한다.

group 메소드 정규식에 일치하는 문자열을 반환한다.
주요메소드는 다음과 같다.

[Pattern 클래스]

반환형	메소드	설명
static Pattern	compile (String)	인수에 지정된 정규식을 컴파일합니다.
static Pattern	compile (String, int)	인수에 지정된 정규식을 컴파일합니다. 제 2 인수에 플래그를 지정하고 섬세한 적합 동작을 지정합니다. 플래그를 지정하면 처리 성능이 저하 될 수 있습니다.CASE_INSENSITIVE US-ASCII 문자에서 대소 문자를 구별하지 않는 적합 처리합니다. MULTILINE 여러 줄을 지원합니다. DOTALL 정규 표현식. 하행 말 기호를 포함한 모든 문자에 적합합니다. UNICODE_CASECASE_INSENSITIVE 와 함께 사용하면 US-ASCII 문자에 맞게 UNICODE 문자에서도 대소 문자를 구별하지 않는 적합 처리합니다. CANON_EQ 표준 등가를 유효하게 한 적합 처리합니다.
Matcher	matcher (CharSequence)	인수에 지정된 문자 순서 (String, StringBuffer 등)과 해당 패턴에 대한 Matcher 객체를 생성합니다.
String []	split (CharSequence)	인수에 지정된 문자 순서 (String, StringBuffer 등)을 해당 패턴에 적합하고 분할합니다.
String []	split (CharSequence, int)	인수에 지정된 문자 순서 (String, StringBuffer 등)을 해당 패턴에 적합하고 분할합니다. 제 2 인수의 수치는 분할 수를 지정합니다.

[Matcher 클래스]

반환형	메소드	설명
Matcher	appendReplacement (StringBuffer, String)	정규식에 맞는지를 차례로 조사하고 적합한 경우 그 부분을 제 2 인수의 캐릭터 라인에 옮겨 제 1 인수의 캐릭터 라인 버퍼에 추가합니다.
StringBuffer	appendTail (StringBuffer)	정규식 적합 조사를 실시하지 않은 나머지 문자 순서를 인수로 지정한 문자열 버퍼에 추가합니다.appendReplacement 메소드와 함께 사용합니다.
boolean	find ()	정규식에 맞는지 차례로 조사합니다.
boolean	find (int)	인수에 지정된 인덱스 위치에서 정규식에 맞는지 차례로 조사합니다.
String	group ()	이전 정규식에 적합한 문자열을 반환합니다.
String	group (int)	이전 정규식에 적합한 문자열 안에 인수에 지정된 정규 표현 그룹 ()에 맞는 문자열을 반환합니다. 그룹 ()는 정규 표현식의 왼쪽에서 1,2 입니다. 0 을 지정했을 경우는 gourp ();을 실행했을 경우와 같은 결과가되어, 정규 표현에 적합한 문자열 전체를 반환합니다.
boolean	matches ()	정규식에 적합한 지 확인하고 적합한 경우 true 를 반환합니다.
String	replaceAll (String)	정규식에 적합한 모든 문자열을 인수로 지정한 문자열로 바꿉니다.
String	replaceFirst (String)	정규식에 적합한 모든 문자열 중 첫 번째 문자열을 인수로 지정한 문자열로 바꿉니다.

다음 소스를 보고 패턴에 따를 문자열 반환식을 살펴 보자.

패턴 1의 정규식

①-1 정규식 패턴에 적합한 지 확인하는 문자열 text1을

```
                    생성합니다.
                ①-2.compile 메소드에서 정규식을 컴파일합니다.
                ①-3.matcher 메소드로 인수 정규식 적합 확인 문자
                    열을 지정하고 Matcher 객체를 생성합니다.
                ①-4.matches 메소드 정규식에 적합한 지 확인합니다.
                    적합한 경우 true를 반환합니다
                String text1 = new String ( "java1 java2 java3");
                Pattern pattern1 =
                  Pattern.compile ( "(java₩₩d)₩₩s(java₩₩d)₩₩s(java ₩₩d)");
                Matcher matcher1 = pattern1.matcher (text1);
                System.out.println (matcher1.matches ());
                System.out.println ( "---------------"+ "₩n");
```

[실행결과]

```
false
---------------
```

```
/* 패턴 2의 정규식
                ②-1 정규식 패턴에 적합한 지 확인하는 문자열 text2를
                    생성합니다.
                ②-2.compile 메소드에서 정규식을 컴파일합니다.
                ②-3.split 메소드 인수에 지정된 문자열을 정규식에 맞
                    춥니 분할하고 분할 된 문자열을 배열로 돌려줍니다.
                ②-4. 분할 된 문자열을 for 루프를 사용하여 표시합니다. */
                String text2 = new String ( "java1 java2 java3");
                Pattern pattern2 = Pattern.compile ( "₩₩s");
                String [] splitStr = pattern2.split (text2);
                for (int i = 0; i <splitStr.length; i++) {
                  System.out.println (splitStr [i]);
                }
                System.out.println ( "---------------"+ "₩n");
```

[실행결과]

```
java1
java2
java3
  ---------------
```

```
/* 패턴 3의 정규식
③-1 정규식 패턴에 적합한 지 확인하는 문자열 text3을
    생성합니다.
③-2.compile 메소드에서 정규식을 컴파일합니다.
③-3.matcher 메소드로 인수 정규식 적합 확인 문자
    열을 지정하고 Matcher 객체를 생성합니다.
③-4.replaceAll 메소드에서 정규식에 맞는 문자열 모든
    를 인수로 지정한 문자열로 바꿉니다. */
String text3 = new String ( "java1 java2 java3");
Pattern pattern3 = Pattern.compile ( "java");
Matcher matcher3 = pattern3.matcher (text3);
System.out.println (matcher3.replaceAll ( "test"));
System.out.println ( "---------------"+ "₩n");
```

[실행결과]

```
test1 test2 test3
---------------
```

```
/* 패턴 4의 정규식
④-1 정규식 패턴에 적합한 지 확인하는 문자열 text4을
    생성합니다.
④-2.compile 메소드에서 정규식을 컴파일합니다.
④-3.matcher 메소드로 인수 정규식 적합 확인 문자
    열을 지정하고 Matcher 객체를 생성합니다.
④-4.find 메소드에서 정규식에 맞는 문자열이 있는지를 순
    차례 조사합니다.
    appendReplacement 메소드로 문자열을 순차적으로 제 2 인수
    지정한 문자열로 대체 제 1 인수의 캐릭터 라인 버퍼에
    추가합니다.
    appendTail 메소드는 특정 위치 이후의 문자열을 인수 손가락
    정 된 문자열 버퍼에 추가합니다.
    find 메소드, appendReplacement 메소드, appendTail
    메소드를 사용하여 replaceAll 메소드와 같은 동작을
    합니다. */
String text4 = new String ( "java1 java2 java3");
Pattern pattern4 = Pattern.compile ( "java");
Matcher matcher4 = pattern4.matcher (text4);
StringBuffer appStrBuf = new StringBuffer ();
```

```
        while (matcher4.find ()) {
            matcher4.appendReplacement (appStrBuf, "test");
        }
        matcher4.appendTail (appStrBuf);
        System.out.println (appStrBuf.toString ());
        System.out.println ( "---------------"+ "₩n");
```

[실행결과]

```
test1 test2 test3
---------------
```

```
        /* 패턴 5 정규 표현식
        ⑤-1 정규식 패턴에 적합한 지 확인하는 문자열 text5을
            생성합니다.
        ⑤-2.compile 메소드에서 정규식을 컴파일합니다.
        ⑤-3.matcher 메소드로 인수 정규식 적합 확인 문자
            열을 지정하고 Matcher 객체를 생성합니다.
        ⑤-4.find 메소드 정규식에 맞는지를 조사합니다.
        ⑤-5.group 메소드로 이전 적합 조사에 적합한 문자열을 반환
            있습니다. group 메소드의 인수에 지정된 수치는 정규 표현식
            그룹 ()을 나타냅니다. 그룹과 일치하는 문자열을 반환
            있습니다. 그룹은 정규식 () 왼쪽에서 1,2,3입니다. */
        String text5 = new String ( "java1 java2 java3");
        Pattern pattern5 =
            Pattern.compile ( "(java₩₩d)₩₩s(java₩₩d)₩₩s(java₩₩d)");
        Matcher matcher5 = pattern5.matcher (text5);
        if (matcher5.find ()) {
            System.out.println (matcher5.group (0));
            System.out.println (matcher5.group (1));
            System.out.println (matcher5.group (2));
            System.out.println (matcher5.group (3));
        }
        System.out.println ( "---------------"+ "₩n");
    }
```

[실행결과]

```
java1 java2 java3
java1
java2
java3
---------------
```