14. 기본 API 클래스

이 장에서는 기본 API 클래스들에 대해 설명합니다.

14장의 주요 내용입니다.

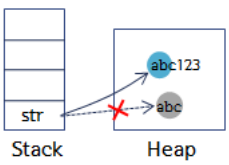
- 문자열 다루기
- 정규 표현식
- Object 클래스
- 날짜 다루기
- 지역과 시간대
- 형식화
- 수학 관련 클래스

14.1. 클래스

14.1.1. String, StringBuffer, StringBuilder

자바에서 문자열을 사용할 경우에 성능을 고려해야 합니다. 자바에서 문자열을 사용하기 위해 앞에서 언급한 String 클래스만 있는 것은 아닙니다. String 클래스는 기본 데이터 유형처럼 바로 값을 할당해서 사용하기 때문에 편리하다는 장점이 있지만 문자열의 내용이 자주 변경되는 경우에는 사용하는 것을 권장하지 않습니다. 그 이유는 String은 immutable 클래스이기 때문입니다. immutable은 문자열의 내용이 변경되면 기존의 메모리 공간에 저장된 데이터를 수정하는 것이 아니고 새로운 문자열 객체를 생성한 다음 새로운 객체를 참조함을 의미합니다.
다음과 같은 코드를 생각해 볼 수 있습니다. concat()은 문자열을 연결해주는 메서드입니다. 아래 코드는 str의 내용에 문자열을 붙여줍니다. 그런데 그림을 보면 문자열의 내용이 변경될 때 힙 메모리 영역에 새로운 String 객체가 생성된다는 것을 알 수 있습니다.

```
String str = "abc"
str = str.concat("123");
```



문자열의 내용이 자주 변경될 경우에 StringBuffer와 StringBuilder를 사용할 것을 권장합니다.
다음 표는 String, StringBuffer, StringBuilder 클래스를 비교했습니다.

	String	StringBuffer	StringBuilder
API 정의	String objects are immutable they can be shared.	A thread-safe, mutable sequence of characters.	A mutable sequence of characters.
스레드 공유	can be shared	thread-safe	can be shared
주요 메서드	concat, replace, substring	append, replace, substring, insert	append, replace, substring, insert
사용 버전	JDK 1.0	JDK 1.0	JDK 1.5

StringBuffer 클래스는 String 클래스에서 제공하지 않는 추가, 삭제 등의 기능을 제공합니다. 문자열을 폭넓게 사용할 필요가 있을 때 StringBuffer 클래스를 이용합니다.

클래스 이름에서 buffer는 문자열을 기억할 메모리, 즉 문자형 배열(char[])을 의미합니다. 문자열을 추가하거나 삭제하면 buffer의 크기가 변한다는 의미이며, 생성자에 따라 버퍼의 크기가 달라집니다.

다음은 StringBuffer 클래스의 생성자입니다.

생성자와 설명
StringBuffer() buffer의 초기 크기(initial capacity)가 16 character 크기인 string buffer를 생성합니다.
StringBuffer(CharSequence seq) 인자로 주어진 CharSequence와 같은 문자열을 갖는 string buffer를 생성합니다.
StringBuffer(int capacity) buffer의 크기를 지정하여 string buffer를 생성합니다.
StringBuffer(String str) 주어진 문자열의 길이만큼 초기화 된 string buffer를 생성합니다.

다음은 String, StringBuffer, StringBuilder클래스의 주요 메서드들입니다.

String 클래스의 주요 메서드

	메서드와 설명
char	charAt(int index) index 위치의 문자를 반환합니다.
int	compareTo(String anotherString) 이 문자열과 주어진 anotherString 문자열과 알파벳 순서를 이용하여 크기 비교를 합니다.
String	concat(String str) 이 문자열에 주어진 str 문자열을 연결합니다.
boolean	contains(CharSequence s) 주어진 문자열을 포함하는지 테스트합니다.
boolean	equals(Object anObject) 이 문자열과 주어진 객체와 내용이 같은지 비교합니다.
boolean	equalsIgnoreCase(String anotherString) 대/소문자를 구분하지 않고 두 문자열이 같은지 비교합니다.
static String	format(Locale l, String format, Object... args) 주어진 로케일, 포맷 그리고 인자들을 이용하여 문자열의 포맷을 지정합니다.
static String	format(String format, Object... args) 디폴트 로케일, 포맷 그리고 인자들을 이용하여 문자열의 포맷을 지정합니다.
byte[]	getBytes() 이 문자열을 바이트 배열로 반환합니다.
byte[]	getBytes(String charsetName) 이 문자열을 주어진 문자셋을 갖는 바이트 배열로 반환합니다.
void	getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin) 이 문자열의 srcBegin 위치부터 srcEnd 위치까지 문자열을 dst 배열의 dstBegin 위치에 복사합니다.
int	indexOf(int ch)

	주어진 문자의 위치를 반환합니다..
int	indexOf(int ch, int fromIndex) fromIndex위치 이후부터 주어진 문자의 위치를 반환합니다.
int	indexOf(String str) 주어진 문자열의 위치를 반환합니다.
int	indexOf(String str, int fromIndex) fromIndex위치 이후부터 주어진 문자의 위치를 반환합니다.
boolean	isEmpty() length()가 0이면 true를 반환합니다.
int	length() 문자열의 길이를 반환합니다.
boolean	matches(String regex) 주어진 정규표현식과 일치하는지 확인합니다.
String	replace(char oldChar, char newChar) 이 문자열에서 주어진 oldChar 문자를 새로운 문자 newChar로 바꿉니다.
String	replaceAll(String regex, String replacement) 주어진 정규 표현식과 일치하는 모든 문자열을 replacement로 바꿉니다.
String[]	split(String regex) 주어진 정규표현식으로 문자열을 쪼개어 배열로 반환합니다.
String	substring(int beginIndex) beginIndex위치 이후의 부분 문자열을 반환합니다.
String	substring(int beginIndex, int endIndex) beginIdex위치부터 endIndex까지 부분 문자열을 반환합니다.
String	toLowerCase() 소문자로 변환한 문자열을 반환합니다.
String	toUpperCase() 대문자로 변환한 문자열을 반환합니다.
String	trim() 문자열 앞/뒤의 공백을 제거한 문자열을 반환합니다.

StringBuffer 클래스의 주요 메서드

	메서드와 설명
StringBuffer	append(String str) 문자열에 주어진 문자열을 연결합니다.
char	charAt(int index) 주어진 위치 문자를 반환합니다.
StringBuffer	delete(int start, int end) 이 문자열에서 start 위치에서 end 위치까지 삭제합니다.
StringBuffer	deleteCharAt(int index) 주어진 index 위치 문자를 삭제합니다.
void	getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin) 이 문자열의 srcBegin 위치부터 srcEnd 위치까지 문자열을 dst 배열의 dstBegin 위치에 복사합니다.
int	indexOf(String str)

	문자열의 위치를 반환합니다.
StringBuffer	insert(int offset, String str) 주어진 offset 위치에 주어진 str 문자열을 추가합니다.
int	length() 문자열의 길이를 반환합니다.
StringBuffer	replace(int start, int end, String str) start 위치부터 end 위치까지 문자열을 str 문자열로 바꿉니다.
StringBuffer	reverse() 이 문자 순서를 순서의 역으로 바꿉니다.
String	substring(int beginIndex) beginIndex위치 이후의 부분 문자열을 반환합니다.
String	substring(int beginIndex, int endIndex) beginIdex위치부터 endIndex까지 부분 문자열을 반환합니다.

StringBuilder 클래스의 주요 메서드들

	메서드와 설명
StringBuilder	append(String str) 이 문자열에 주어진 문자열을 연결합니다.
char	charAt(int index) 주어진 위치 문자를 반환합니다.
StringBuilder	delete(int start, int end) 이 문자열에서 start 위치에서 end 위치까지 삭제합니다.
StringBuilder	deleteCharAt(int index) 주어진 index 위치 문자를 삭제합니다.
void	getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin) 이 문자열의 srcBegin 위치부터 srcEnd 위치까지 문자열을 dst 배열의 dstBegin 위치에 복사합니다.
int	indexOf(String str) 주어진 문자열의 위치를 반환합니다.
StringBuilder	insert(int offset, String str) 주어진 offset 위치에 주어진 str 문자열을 추가합니다.
int	length() 문자열의 길이를 반환합니다.
StringBuilder	replace(int start, int end, String str) start 위치부터 end 위치까지 문자열을 str 문자열로 바꿉니다.
StringBuilder	reverse() 이 문자 순서를 순서의 역으로 바꿉니다.
String	substring(int beginIndex) beginIndex위치 이후의 부분 문자열을 반환합니다.
String	substring(int beginIndex, int endIndex) beginIdex위치부터 endIndex까지 부분 문자열을 반환합니다.

다음 코드는 위 클래스들의 내용을 수정하기 전/후 해시코드 값을 출력하는 예입니다. 실행 결과를 주의해서 살펴볼 필요가 있습니다.

StringExample.java

```
1:  package string;
2:
3:  public class StringExample {
4:     public static void main(String[] args) {
5:        String str = "abc";
6:        System.out.println(str.hashCode());
7:        str = str.concat("123");
8:        System.out.println(str.hashCode());
9:
10:       StringBuffer strBuf = new StringBuffer("abc");
11:       System.out.println(strBuf.hashCode());
12:       strBuf = strBuf.append("123");        //할당문이 없어도 됩니다.
13:       System.out.println(strBuf.hashCode());
14:
15:       StringBuilder strBld = new StringBuilder("abc");
16:       System.out.println(strBld.hashCode());
17:       strBld = strBld.append("123");        //할당문이 없어도 됩니다.
18:       System.out.println(strBld.hashCode());
19:    }
20: }
```

```
Console
<terminated> String
96354
-1424436592
366712642
366712642
1829164700
1829164700
```

14.1.2. StringTokenizer

java.util.StringTokeninzer는 문자열을 분리하기 위한 클래스입니다. 생성자의 인자로 분리하기 위한 문자열과 구분자(delimiters)들을 가질 수 있습니다. StringTokenizer 클래스는 기본생성자가 없으며 구분자(delimiters)를 지정하지 않으면 문자열을 구분하는 구분자는 공백(“ ₩t₩n₩r₩f”)[38]입니다.

다음은 StringTokenizer 클래스의 생성자입니다.

설명
StringTokenizer(String str) , ₩t, ₩n, ₩r, ₩f 문자로 str 문자열을 구분한 객체를 생성합니다.
StringTokenizer(String str, String delim) 주어진 delim 문자열안의 각 문자로 문자로 str 문자열을 구분한 객체를 생성합니다.
StringTokenizer(String str, String delim, boolean returnDelims) 주어진 delim 문자열안의 각 문자로 문자로 str 문자열을 구분한 객체를 생성합니다. 구분자도 토큰에 포함하게 할지 returnDelims 값을 지정할 수 있습니다. returnDelims 기본 값은 false입니다.

38) space, tab, newline, carriage-return

다음은 StringTokenizer 클래스의 주요 메서드입니다.

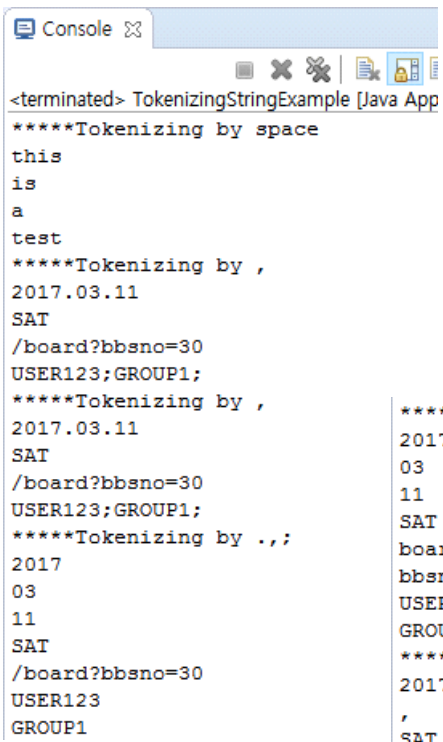
	메서드
int	countTokens() 발생되기 전에 nextToken 메서드가 호출될 수 있는 횟수를 계산합니다.
boolean	hasMoreElements() hasMoreTokens 메서드와 같은 값을 반환합니다..
boolean	hasMoreTokens() 현재 토크나이저 문자열로부터 더 이상의 토큰이 있는지 테스트합니다.
Object	nextElement() nextToken 메서드와 같은 값을 반환합니다. 선언된 리턴 타입은 String이 아니고 Object입니다.
String	nextToken() 현재 토크나이저로부터 다음 문자열을 반환합니다.
String	nextToken(String delim) 현재 문자열에서 delim으로 구분되는 다음 문자열을 반환합니다.

다음은 StringTokenizer 클래스를 이용하여 문자열을 분리한 예입니다.

StringTokenizer.java

```
1:  import java.util.StringTokenizer;
2:
3:  public class StringTokenizerExample {
4:
5:     public static void main(String[] args) {
6:
7:        StringTokenizer st = new StringTokenizer("this is a test");
8:        System.out.println("*****Tokenizing by space");
9:        while (st.hasMoreTokens()) {
10:          System.out.println(st.nextToken());
11:       }
12:
13:       String accessLog = "2017.03.11,SAT,/board?bbsno=30,USER123;GROUP1;";
14:
15:       StringTokenizer accessTokens1 = new StringTokenizer(accessLog, ",");
16:       System.out.println("*****Tokenizing by ,");
17:       while(accessTokens1.hasMoreTokens()) {
18:          System.out.println(accessTokens1.nextToken());
19:       }
20:
21:       System.out.println("*****Tokenizing by ,");
22:       StringTokenizer accessTokens5 = new StringTokenizer(accessLog);
23:       while(accessTokens5.hasMoreTokens()) {
24:          System.out.println(accessTokens5.nextToken(","));
25:       }
26:
27:       System.out.println("*****Tokenizing by .,;");
28:       StringTokenizer accessTokens2 = new StringTokenizer(accessLog, ".,;");
29:       while(accessTokens2.hasMoreTokens()) {
30:          System.out.println(accessTokens2.nextToken());
```

```
31:         }
32:
33:         System.out.println("*****Tokenizing by .,;/?");
34:         StringTokenizer accessTokens3 = new StringTokenizer(accessLog, ".,;/?");
35:         while(accessTokens3.hasMoreTokens()) {
36:            System.out.println(accessTokens3.nextToken());
37:         }
38:
39:         System.out.println("*****Tokenizing by ,(       토큰에 포함)");
40:         StringTokenizer accessTokens4 = new StringTokenizer(accessLog, ",",
    true);
41:         while(accessTokens4.hasMoreTokens()) {
42:            System.out.println(accessTokens4.nextToken());
43:         }
44:      }
45:
46: }
```



14.1.3. split()

split() 메서드는 String 클래스에 있는 메서드입니다. split() 메서드는 정규표현식을 이용하여 문자열을 분리할 수 있습니다. split() 메서드는 JDK 1.4부터 사용할 수 있습니다.

다음은 split() 메서드에 대한 설명입니다.

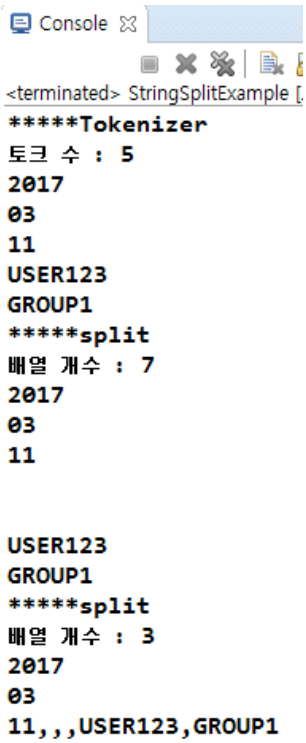
	메서드
String[]	split(String regex) 정규 표현식을 이용하여 문자열을 분리하고 그 결과를 문자열 배열로 반환합니다.
String[]	split(String regex, int limit) 주어진 정규 표현식을 이용하여 문자열을 분리하고 그 결과를 문자열 배열로 반환합니다. 결과의 수를 limit 개수로 제한합니다.

다음 코드는 StringTokenizer클래스와 split() 메서드를 비교 설명하기 위한 예제입니다.

StringSplitExample.java

```
1:  import java.util.StringTokenizer;
2:
3:  public class StringSplitExample {
4:
5:     public static void main(String[] args) {
6:        String accessLog = "2017,03,11,,,USER123,GROUP1";
7:        System.out.println("*****Tokenizer");
8:        StringTokenizer accessTokens = new StringTokenizer(accessLog, ",");
9:        System.out.println("토크 수 : " + accessTokens.countTokens());
10:       while(accessTokens.hasMoreTokens()) {
11:          System.out.println(accessTokens.nextToken());
12:       }
13:
14:       System.out.println("*****split");
15:       String[] splits = accessLog.split(",");
16:       System.out.println("배열 개수 : " + splits.length);
17:       for(String str : splits) {
18:          System.out.println(str);
19:       }
20:
21:       System.out.println("*****split");
22:       String[] splits2 = accessLog.split(",", 3);
23:       System.out.println("배열 개수 : " + splits2.length);
24:       for(String str : splits2) {
25:          System.out.println(str);
26:       }
27:    }
28:
29: }
```

StringTokenizer를 이용하여 문자열을 분리할 경우에는 구분자 사이에 아무것도 없을 경우 토큰이 생성되지 않습니다. 그러나 split() 메서드는 구분자 사이에 아무것도 없더라도 “”(널스트링)을 생성하기 때문에 같은 문자열을 분리하였을 경우 분리되는 문자열의 개수가 다릅니다.



14.2. 표현식

String 클래스의 split()와 replaceAll() 메서드는 정규 표현식을 이용하여 문자열을 처리할 수 있습니다. 정규 표현식(regular expression)은 특정한 규칙을 가진 문자열을 표현하는데 사용하는 형식 언어입니다. 정규 표현식은 여러 텍스트 편집기와 프로그래밍 언어에서 문자열을 찾거나 바꾸기 위해 지원하고 있습니다. 자바 언어에서도 문자열을 다루는 여러 클래스와 메서드에서 정규 표현식을 지원하고 있습니다.

14.2.1. 정규 표현식

표현식 기본 개념

정규 표현식은 패턴(pattern) 또는 정규식으로 불리기도 합니다. 정규 표현식은 특정 목적을 위해 필요한 문자열 집합을 지정하기 위해 사용합니다.

다음 표는 정규 표현식의 기본 개념에 대한 설명입니다.

	기능	설명
수직선(|)	또는(or)	항목들 중 하나를 선택하기 위해 구분합니다. 예를 들면 "gray|grey"는 "gray" 또는 "grey"와 일치합니다.
괄호(())	그룹 묶기	연산자의 범위와 우선순위를 지정할 수 있습니다. 예를 들면 "gray|grey"와 "gr(a|e)y"는 "gray" 또는 "grey" 집합 둘 다 의미하는 같은 패턴입니다.
물음표(?)	0회 또는 1회	0번 또는 1번까지의 발생을 의미합니다. 예를 들면 "colou?r"는 "color"와 "colour" 둘 다 일치 시킵니다.
별표(*)	0회 이상	0번 이상 발생을 의미합니다. 예를 들면 "ab*c"는 "ac", "abc", "abbc", "abbbc" 등을 일치 시킵니다.
덧셈기호(+)	1회 이상	1번 이상의 발생을 의미합니다. 예를 들면 "ab+c"는 "abc", "abbc", "abbbc" 등을 일치 시킵니다. 그러나 "ac"는 일치시키지 않습니다.
{n}	n회	정확히 n번만큼 일치시킵니다.
{m, }	m회 이상	m번 이상만큼 일치시킵니다.
{m, n}	m회 이상 n회 이하	적어도 m번만큼 일치시킵니다. 그러나 n번을 초화하여 일치시키지는 않습니다. "a{1,3}b"는 "ab", "aab", "aaab"를 포함하지만, "b"나 "aaaab"는 포함하지 않습니다

표준 문법

문자들과 "["와 "]" 사이에 포함된 문자 집합 외부에서는 12개 문자가, 내부에서는 오직 5개의 문자("₩", "^", "-", "[", "]")만 특수문자를 의미합니다.

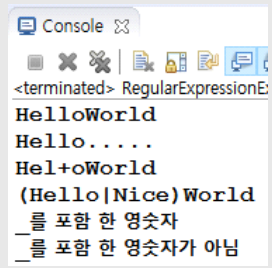
	기능	설명
.		문자 한 개와 일치시킵니다.
[ ]	문자들	여러 문자들 중 하나의 문자와 일치시킵니다.
[^]	부정	해당 문자들을 포함하지 않는 문자들을 찾습니다.
^	처음	처음 시작하는 부분을 의미합니다.
$	끝	끝나는 부분을 의미합니다.
( )	하위 식	하위 식을 정의합니다.
₩w	"_"와 영숫자	"_"를 포함한 영문자와 숫자를 일치시킵니다.
₩W	₩w 반대	"_"와 영문자 그리고 숫자를 제외한 다른 문자열들을 일치 시킵니다.
₩s	공백	공백 문자와 일치시킵니다.
₩S	공백 제외	공백을 제외한 어떤 것이든 일치시킵니다.
₩d	숫자	숫자와 일치시킵니다.
₩D	숫자 제외	숫자가 아닌 항목을 일치시킵니다.

다음 코드는 정규 표현식의 메타문자들에 대한 예입니다. 자바에서 ₩를 문자열 안에 사용하기 위해서는 ₩₩처럼 ₩를 하나 더 붙여 줘야 합니다.

RegularExpressionExample.java

```
1:  public class RegularExpressionExample {
2:     public static void main(String[] args) {
3:        String str1 = "HelloWorld";
4:        if(str1.matches("HelloWorld")) {
5:           System.out.println("HelloWorld");
6:        }
7:        if(str1.matches("Hello....."){
8:           System.out.println("Hello.....");
9:        }
10:       if(str1.matches("Hel+oWorld")) {
11:          System.out.println("Hel+oWorld");
12:       }
13:       str1 = "NiceWorld";
14:       if(str1.matches("(Hello|Nice)World")) {
```

```
15:            System.out.println("(Hello|Nice)World");
16:        }
17:        if(str1.matches("\\w*")) {
18:            System.out.println("_  포함 한 영숫자");
19:        }
20:        if(str1.matches("\\W*")) {
21:            System.out.println("_를 포함 한 영숫자");
22:        }else {
23:            System.out.println("_를 포함 한 영숫자가 아님");
24:        }
25:    }
26: }
```



다음 코드는 문자열에서 정규 표현식을 이용해서 전화번호를 마스크 처리하는 예입니다.

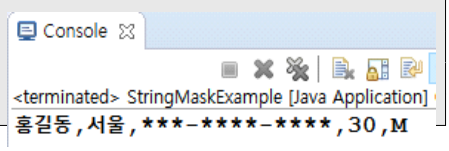
StringMaskExample.java

```
public class StringMaskExample {

 public static void main(String[] args) {
    String custInfo = "홍길동,서울,010-1234-5678,30,M";

    custInfo = custInfo.replaceAll("\\d{3}-\\d{4}-\\d{4}", "***-****-****");
    System.out.println(custInfo);
 }

}
```



14.2.2. Pattern & Matcher

Pattern 클래스와 Matcher 클래스를 이용하면 문자열에서 정규 표현식을 이용하여 원하는 형식을 갖는 문자열을 추출할 수 있습니다. 두 클래스 모두 JDK 1.4에 추가되었으며 java.util.regex 패키지 안에 정의되어 있습니다.

Pattern

Pattern 클래스는 정규 표현식을 컴파일 한 객체입니다. Pattern 클래스의 API 문서[39]를 참고하면 자바의 정규 표현식 문자열들에 대해 더 많은 예를 볼 수 있습니다. Pattern 클래스는 static 메서드를 이용해서 객체를 생성합니다.

39) http://docs.oracle.com/javase/8/docs/api/

다음 표는 Pattern 객체를 생성하기 위한 메서드입니다.

	메서드와 설명
static Pattern	compile(String regex) 정규 표현식을 이용하여 패턴으로 컴파일 합니다.
static Pattern	compile(String regex, int flags) 주어진 플래그 값과 정규 표현식을 이용하여 패턴으로 컴파일 합니다. 플래그 값은 CASE_INSENSITIVE, MULTILINE, DOTALL, UNICODE_CASE, CANON_EQ, UNIX_LINES, LITERAL, UNICODE_CHARACTER_CLASS 그리고 COMMENTS 등을 가질 수 있으며 비트연산자를 이용하여 여러 개 지정할 수 있습니다

Pattern 클래스의 matcher() 메서드를 이용하여 Matcher 객체를 생성할 수 있습니다.

리턴타입	메서드와 설명
Matcher	matcher(CharSequence input) 이 패턴과 지정한 입력을 매치하는 정규 표현 엔진을 만듭니다.

Matcher

Matcher 클래스는 패턴을 이용하여 찾기 위한 클래스입니다. 다음 표는 Matcher 클래스의 주요 메서드들에 대한 설명입니다.

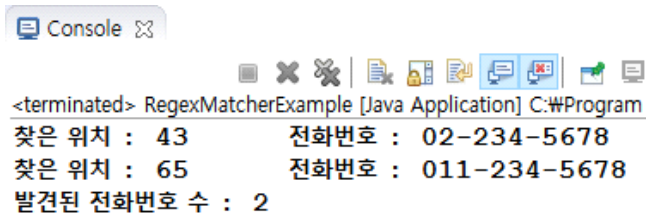
리턴타입	메서드와 설명
boolean	find() 문자열에서 다음 매치하는 문자열을 찾습니다.
boolean	find(int index) 문자열의 index위치 이후부터 다음 매치하는 문자열을 찾습니다.
String	group() 이전 match에서 찾은 문자열을 반환합니다.

다음 코드는 Pattern클래스와 Matcher 클래스를 이용하여 문자열에서 전화번호만 뽑아내는 예입니다.

RegexMatcherExample.java

```
1:  import java.util.regex.Matcher;
2:  import java.util.regex.Pattern;
3:
4:  public class RegexMatcherExample {
5:
6:     public static void main( String args[] ) {
7:        String customerInfo =
8:                 "홍길동은 30세 이며 서울시 강남구에 거주합니다. "
9:               + "그의 집 전화번호는 02-234-5678이며 "
10:              + "휴대폰 번호는 011-234-5678입니다.";
11:
```

```
12:         String pattern = "\\d{2,3}-\\d{3,4}-\\d{4}";
13:
14:         Pattern r = Pattern.compile(pattern);
15:
16:         Matcher m = r.matcher(customerInfo);
17:         int count = 0;
18:         while(m.find()) {
19:            count++;
20:            System.out.print("     위치 : " + m.start() + "\t");
21:            System.out.println("전화번호 : " + m.group());
22:         }
23:         System.out.println("발견된 전화번호 수 : " + count);
24:     }
25: }
```



14.3. Object

Object 클래스는 모든 클래스의 최상위 클래스입니다. Object 클래스의 메서드는 모든 메서드가 재정의 할 수 있습니다.

Object 클래스의 주요 메서드들의 메서드 선언입니다.

- protected Object clone() throws CloneNotSupportedException
- public int hashCode()
- public boolean equals(Object obj)
- public String toString()
- public final void wait(long timeout) throws InterruptedException
- public final void wait(long timeout, int nanos) throws InterruptedException
- public final void wait() throws InterruptedException
- public final void notify()
- public final void notifyAll()
- protected void finalize() throws Throwable

14.3.1. 객체 동등 비교

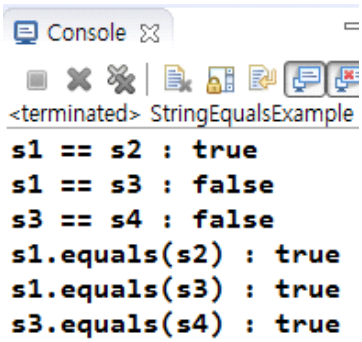
== 연산자는 두 변수의 값을 비교하여 같은 값이면 true를 반환합니다. 그러나 비교하는 변수가 객체일 경우에는 객체의 값을 이용하여 비교하는 것이 아니고 객체의 주소(해시코드)값을 비교하기 때문에 내용이 같더라도 다른 객체이면 false를 반환합니다.
동등 비교에 사용하는 변수가 객체 변수일 경우 equals() 메서드를 이용하여 비교할 수 있습니다. 대부분 Java API에서 제공하는 클래스들은 equals() 메서드를 재정의 하고 있으므로 equals() 메서드를 사용할 수 있습니다.

다음 코드는 String 객체를 이용하여 == 연산자와 equals() 메서드 차이점을 보겠습니다.

StringEqualsExample.java

```
1:  public class StringEqualsExample {
2:      public static void main(String[] args) {
3:          String s1 = "Hello";
4:          String s2 = "Hello";
5:          String s3 = new String("Hello");
6:          String s4 = new String("Hello");
7:          System.out.println("s1 == s2 : " + (s1 == s2));
```

```
8:         System.out.println("s1 == s3 : " + (s1 == s3));
9:         System.out.println("s3 == s4 : " + (s3 == s4));
10:        System.out.println("s1.equals(s2) : " + s1.equals(s2));
11:        System.out.println("s1.equals(s3) : " + s1.equals(s3));
12:        System.out.println("s3.equals(s4) : " + s3.equals(s4));
13:    }
14: }
```



위 코드에서 s1, s2, s3, s4 변수는 모드 String 객체입니다. 그런데 s1과 s2는 문자열 상수를 직접 변수에 대입 시켰습니다. 그러나 s3, s4는 new 키워드를 이용하여 객체를 생성시켰습니다. 자바는 new 키워드를 이용하연 항상 새로운 객체를 생성합니다. 그러므로 s3, s4 객체는 값은 같지만 두 객체의 주소는 다릅니다. s3과 s4 변수의 내용이 같은지 비교하려면 String 클래스에 재정의 되어 있는 equals() 메서드를 이용하면 됩니다. 실행 결과에서도 보듯이 s3, s4 문자열 변수의 내용을 비교하기 위해서는 equals() 메서드를 이용해야 합니다.

14.3.2. equals()와 hashCode()

String 클래스에는 equals() 메서드가 재정의 되어 있기 때문에 equals() 메서드를 이용하여 객체의 동등 비교를 할 수 있습니다. 그러나 직접 클래스를 정의하고 객체 동등 비교를 하기 위해서라면 equals() 메서드를 재정의해야 합니다.

그런데, 객체 동등비교를 위해 재정의 하는 메서드가 equals() 만 있는 것은 아닙니다. 객체 동등비교를 매번 equals() 메서드를 이용한다면 객체가 가진 모든 속성들을 모두 비교해야 하기 때문에 성능이 저하될 수 있을 것입니다. 만약에 두 객체의 해시코드 값을 계산하여 두 객체의 해시코드 값이 다를 경우는 두 객체가 다른 객체 이므로 equals() 메서드를 이용해서 객체 동등 비교를 할 필요가 없을 것입니다. 해시코드는 Object 클래스의 hashCode() 메서드를 이용해 반환 된 값을 이용합니다. 그러므로 사용자 정의 클래스에 hashCode() 메서드를 재정의 하고 equals() 메서드와

hashCode() 메서드를 객체 동등 비교에 사용합니다. hashCode() 메서드는 리턴한 해시코드 값이 다르면 두 객체는 다름을 보장하도록 구현하면 됩니다.

객체 동등을 위해 hashCode() 메서드와 equals() 메서드를 재정의 하는 것은 어렵지 않습니다. 아이디와 색상 정보를 갖는 펜 클래스를 정의해 예를 들겠습니다.
펜 클래스는 아래와 같이 작성할 수 있습니다.

Pen.java

```
1:  public class Pen {
2:      String id
3:      String color
4:      public Pen(String id, String color) {
5:          this.id = id
6:          this.color = color
7:      }
8:      //      hashCode 메서드와 equals() 메서드 재정의
9:  }
```

Pen 클래스의 인스턴스를 생성하고 이를 비교하는 코드를 작성하면 다음과 같습니다.
PenEqualsExample.java

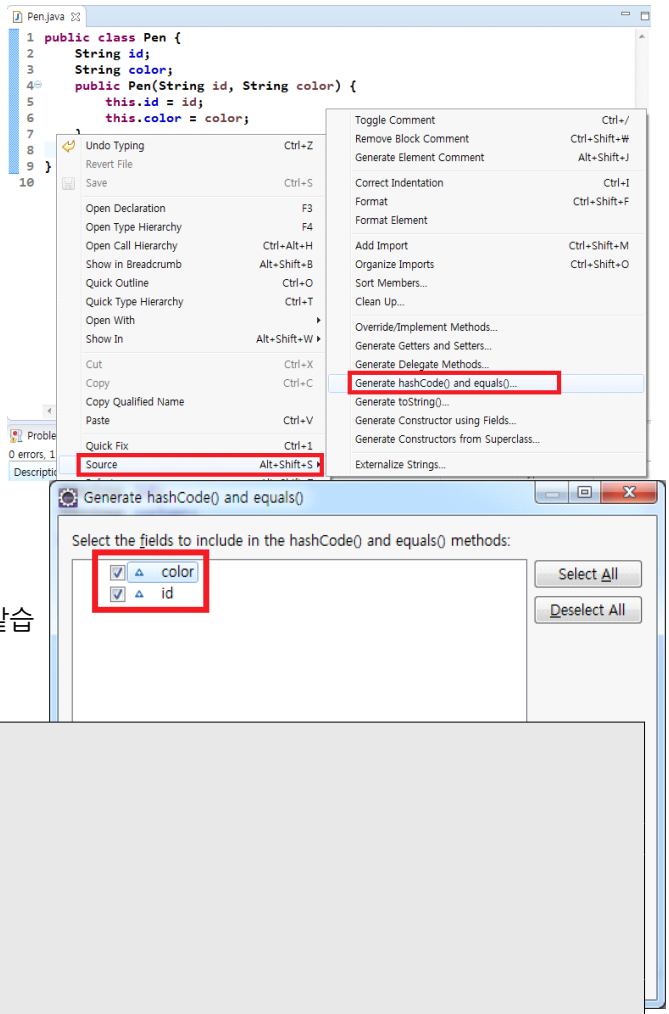
```
47: public class PenEqualsExample {
48:     public static void main(String[] args) {
49:         Pen p1 = new Pen("p0001", "빨간팬");
50:         Pen p2 = new Pen("p0001", "빨간팬");
51:         System.out.println("p1 == p2 : " + (p1 == p2));
52:         System.out.println("p1.equals(p2) : " + p1.equals(p2));
53:     }
54: }
```

```
Console
<terminated> PenEqualsExample [Jav
p1 == p2 : false
p1.equals(p2) : false
```

Pen 객체 p1, p2는 아이디(id)와 색상(color)이 같습니다. 만일 아이디와 색상이 같을 경우 두 객체는 같은 객체임을 보장하고 싶습니다. 그런데 실행 결과에는 == 연산자와 equals() 메서드 모두 false를 출력하고 있습니다. 이 코드는 아직 Pen 클래스에 equals() 메서드와 hashCode() 메서드를 구현하지 않았으므로 실행 결과는 모두 false가 출력됩니다. 그래서 Pen 클래스에 equals() 메서드와 hashCode() 메서드를 추가합니다.

이클립스를 사용한다면 소스코드에서 마우스 오른쪽 버튼을 클릭한 다음 Source > Generate hashCode() and equals()... 메뉴를 선택하면 equals() 메서드와 hashCode() 메서드를 쉽게 재정의 할 수 있습니다.

Generate hashCode() and equals() 창이 뜨면 객체 동등비교에 사용할 변수들을 체크 한 다음 [OK] 버튼만 클릭해 주면 됩니다.

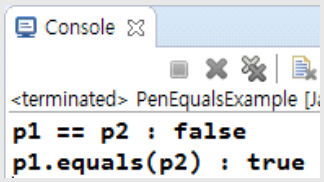


완성된 Pen 클래스는 다음과 같습니다.

Pen.java( )

```
1:  public class Pen {
2:      String id;
3:      String color;
4:      public Pen(String id,
    String color) {
5:          this.id = id;
6:          this.color = color;
7:      }
8:      @Override
9:      public int hashCode() {
10:         final int prime = 31;
11:         int result = 1;
12:         result = prime * result + ((color == null) ? 0 : color.hashCode());
13:         result = prime * result + ((id == null) ? 0 : id.hashCode());
14:         return result;
15:     }
16:     @Override
17:     public boolean equals(Object obj) {
18:         if (this == obj)
19:             return true;
20:         if (obj == null)
21:             return false;
22:         if (getClass() != obj.getClass())
23:             return false;
24:         Pen other = (Pen) obj;
25:         if (color == null) {
26:             if (other.color != null)
```

```
27:             return false;
28:         } else if (!color.equals(other.color))
29:            return false;
30:         if (id == null) {
31:            if (other.id != null)
32:               return false;
33:         } else if (!id.equals(other.id))
34:            return false;
35:         return true;
36:     }
37: }
```



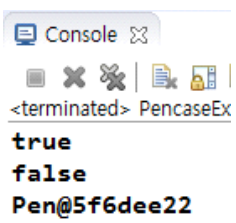
PenEqualsExample.java 파일을 다시 실행하면 equals() 메서드를 이용한 객체 동등 비교시 true가 출력됩니다. 그런데 Pen 클래스 hashCode() 메서드는 필요 없을 것 같습니다. 사실 위 예제에서 hashCode는 필요 없습니다. 그런데 만일 같은 객체는 한 개만 저장하는 것을 보장하는 자료구조에 Pen 객체를 저장할 때는 어떨까요?

다음 PencaseExample 클래스에서는 HashSet 클래스를 이용해 Pen 객체를 저장하는 예를 보여주고 있습니다. HashSet은 동일객체를 저장하지 않도록 하는 자료구조 클래스 중 하나입니다.

PencaseExample.java

```
1:  import java.util.HashSet;
2:
3:  public class PencaseExample {
4:     public static void main(String[] args) {
5:        Pen p1 = new Pen("p0001", "     ");
6:        Pen p2 = new Pen("p0001", "빨간팬");
7:
8:        HashSet<Pen> pens = new HashSet<Pen>();
9:        System.out.println(pens.add(p1));
10:       System.out.println(pens.add(p2));
11:       for(Pen pen : pens) {
12:          System.out.println(pen);
13:       }
14:    }
15: }
```

위 코드를 실행시키면 오른쪽 결과처럼 add(p1)의 결과는 true 이지만 add(p2)는 false 가 출력됩니다. 이것은 두 객체가 같은 객체로 판별되고 그래서 두 번째 add 한 객체는 HashSet 엘리먼트에 추가가 되지 않음을 의미합니다. 실제 pens 세트에 저장되어 있는 엘리먼트를 출력할 때에도 Pen 객체가 하나만 출력되는 것을 확인할 수 있습니다.



그러면, Pen.java 파일의 hashCode() 메서드를 주석처리 하고 실행해 보세요. 여러 줄 주석처리는 블록을 설정한 다음 Ctrl+/를 누르면 쉽게 주석을 생성할 수 있습니다.

Pen.java | PenEqualsExample.java | PencaseExample.java

```
1  public class Pen {
2      String id;
3      String color;
4      public Pen(String id, String color) {
5          this.id = id;
6          this.color = color;
7      }
8  //  @Override
9  //  public int hashCode() {
10 //      final int prime = 31;
11 //      int result = 1;
12 //      result = prime * result + ((color == null
13 //      result = prime * result + ((id == null) ?
14 //      return result;
15 //  }
16     @Override
17     public boolean equals(Object obj) {
18         if (this == obj)
19             return true;
```

Pen 클래스에서 hashCode() 메서드를 주석처리 하고 PencaseExample 클래스를 실행시키면 HashSet에 Pen 객체가 둘 다 저장되는 것을 확인할 수 있습니다. HashSet 클래스의 add() 메서드는 객체 저장을 성공하면 true, 그렇지 않다면 falsue를 반환합니다. 즉, 동일 객체가 이미 HashSet에 저장되어 있어 같은 객체가 저장되지 않을 경우 false를 반환합니다.

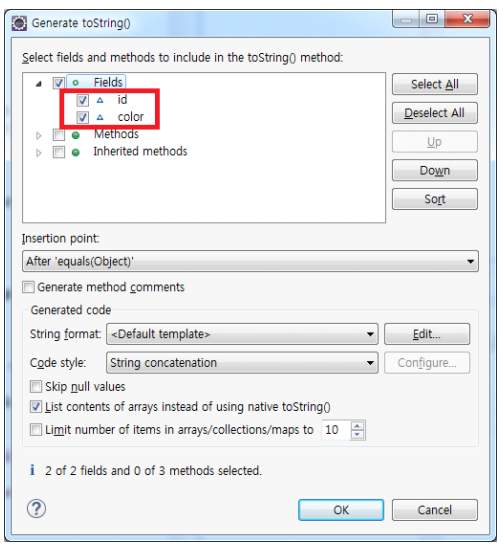
Console

<terminated> PencaseEx

```
true
true
Pen@15db9742
Pen@6d06d69c
```

14.3.3. toString()

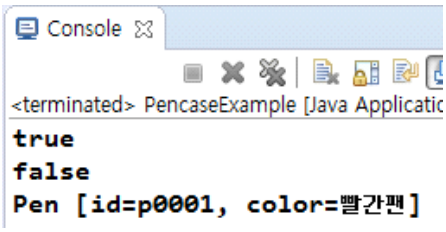
가끔 프로그램을 작성하다보면 객체가 가진 값을 화면에 출력해 봐야 할 경우가 있습니다. 그럴 경우에 println() 메서드를 사용할 수 있는데 println()메서드 인자가 객체일 경우 자동으로 .toString()을 붙여 실행시켜 줍니다. toString() 메서드도 Object에 정의되어 있는 메서드 이며 모든 클래스에서 재정의 할 수 있습니다. 이클립스에서는 toString() 메서드도 자동을 생성시킬 수 있습니다. 소스코드 화면에서 마우스 오른쪽 버튼을 클릭한 다음 Source > Generate toString()...을 선택하면 다음과 같이 Generate toString() 창이 보입니다. 이곳에서 화면에 출력하고 싶은 필드들을 선택한 다음 [OK] 버튼을 클릭하면 toString() 메서드가 자동으로 만들어집니다.



다음은 Pen.java에 추가된 toString() 메서드입니다.

```
@Override
public String toString() {
    return "Pen [id=" + id + ", color=" + color + "]";
}
```

Pen 클래스에 toString() 메서드를 추가 한 다음 PencaseExample.java 파일을 실행하면 다음과 같이 Pen의 정보가 출력됩니다.



14.4. 클래스

프로그램에서 날짜와 시간을 다루는 것은 매우 중요한 일입니다. 자바는 시스템의 날짜와 시간을 알 수 있도록 java.util 패키지에 Date 클래스와 Calendar 클래스를 제공합니다.

14.4.1. Date

Date 클래스는 컴퓨터의 날짜 정보를 읽어 Date 객체로 만들어 줍니다. Date 클래스의 생성자는 현재 날짜 정보를 Date 객체로 만들어주는 Date() 기본 생성자와 1970년 1월 1일 00:00:00 초 이후의 시간을 밀리초 단위로 입력해서 Date 객체로 만들어주는 Date(long date) 생성자가 있습니다. Date 클래스의 주요 메서드는 toString()과 getTime()입니다. toString()은 현재 날짜와 시간 정보를 문자열로 반환하며, getTime() 메서드는 현재 날짜 시간을 1970년 1월 1일 0시를 기준으로 현재시점을 밀리초 값으로 반환합니다.

DateExample.java

```
1:  import java.util.Date;
2:
3:  public class DateExample {
4:      public static void main(String[] args) {
5:          Date now = new Date();
6:          System.out.println(now);
7:          System.out.println(now.getTime());
8:          Date after = new Date();
9:          System.out.println(after.getTime());
10:     }
11: }
```

```
Console
<terminated> DateExample [Java Application] C
Sun Jul 17 21:12:20 KST 2016
1468757540361
1468757540371
```

Date 클래스는 주로 객체들 사이에 날짜 및 시간 정보를 주고받기 위해 사용합니다. 현재 시간 정보를 출력하기 위해서라면 Calendar 클래스를 사용할 것을 권합니다.

현재 날짜 및 시간을 조회할 때마다 Date 객체를 생성해야 하기 때문에 날짜 시간 정보를 반복적/주기적으로 조회하는 경우라면 Calendar 클래스가 더 좋습니다. 그리고 Date 클래스는 java.util 패키지에만 있는 것은 아닙니다. java.sql 패키지에도 Date 클래스가 있습니다. JDBC 프로그래밍 할 때에는 java.sql 패키지의 Date 클래스가 사용될 수 있습니다.

14.4.2. Calendar

Calendar 클래스는 달력을 표현한 추상 클래스입니다. Date 클래스처럼 new를 이용하여 직접 객체를 생성할 수 없고 getInstance() 메서드를 이용하여 Locale 또는 TimeZone 에 따른 Calendar 객체를 얻을 수 있습니다.

다음은 Calendar 클래스의 getInstance() 메서드들입니다.

	메서드와 설명
static Calendar	getInstance() Gets a calendar using the default time zone and locale.
static Calendar	getInstance(Locale aLocale) Gets a calendar using the default time zone and specified locale.
static Calendar	getInstance(TimeZone zone) Gets a calendar using the specified time zone and default locale.
static Calendar	getInstance(TimeZone zone, Locale aLocale) Gets a calendar with the specified time zone and locale.

기본적으로 getInstance() 메서드를 이용하면 현재 운영체제에 설정되어 있는 TimeZone(시간대)를 기준으로 Calendar 객체를 얻을 수 있습니다. Calendar 객체를 얻었다면 get() 메서드와 Calendar 클래스의 여러 상수변수들을 이용하여 날짜 및 시간 정보를 얻을 수 있습니다.

다음 코드는 Calendar 클래스를 이용하여 날짜 및 시간 정보를 출력하는 프로그램입니다.

CalendarExample.java

```
1:  import java.util.Calendar;
2:  import java.util.TimeZone;
3:
4:  public class CalendarExample {
5:     public static void main(String[] args) {
6:        Calendar cal = Calendar.getInstance();
7:        System.out.print(TimeZone.getDefault().getID() + " : ");
8:        System.out.print(cal.get(Calendar.YEAR) + "   ");
9:        System.out.print(cal.get(Calendar.MONTH)+1 + "월 ");
10:       System.out.print(cal.get(Calendar.DATE) + "일 ");
11:       System.out.print(cal.get(Calendar.AM_PM) == 0 ? "오전 " : "오후 " );
12:       System.out.print(cal.get(Calendar.HOUR) + "시 ");
13:       System.out.print(cal.get(Calendar.MINUTE) + "분 ");
14:       System.out.println(cal.get(Calendar.SECOND) + "초");
15:       System.out.println(cal.getTimeInMillis());
16:
17:       TimeZone timeZone = TimeZone.getTimeZone("America/Los_Angeles");
18:       cal = Calendar.getInstance(timeZone);
```

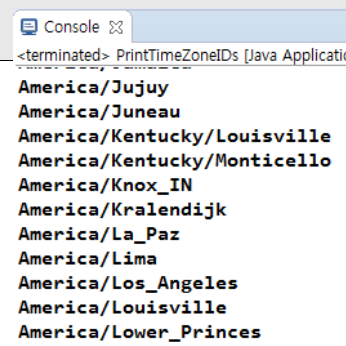
```
19:         System.out.print(timeZone.getID() + " : ");
20:         System.out.print(cal.get(Calendar.YEAR) + "   ");
21:         System.out.print(cal.get(Calendar.MONTH)+1 + "월 ");
22:         System.out.print(cal.get(Calendar.DATE) + "일 ");
23:         System.out.print(cal.get(Calendar.AM_PM) == 0 ? "오전 " : "오후 " );
24:         System.out.print(cal.get(Calendar.HOUR) + "시 ");
25:         System.out.print(cal.get(Calendar.MINUTE) + "분 ");
26:         System.out.println(cal.get(Calendar.SECOND) + "초");
27:         System.out.println(cal.getTimeInMillis());
28:     }
29: }
```

연도와 날짜, 시간, 분, 초는 조회한 값을 그대로 사용해도 되지만 월은 1을 더해줘야 합니다. Calendar.MONTH 상수를 이용해 월을 출력하면 1월일 경우 0, 2월일 경우 1이 출력됩니다.

사용가능한 시간대를 알고 싶다면 다음 TimeZone.getAvailableIDs() 메서드를 이용하면 됩니다.

PrintTimeZoneIDs.java

```
1:  import java.util.TimeZone;
2:
3:  public class PrintTimeZoneIDs {
4:     public static void main(String[] args) {
5:        for(String id : TimeZone.getAvailableIDs()) {
6:           System.out.println(id);
7:        }
8:     }
9:  }
```



14.5. 시간대 클래스

14.5.1. Locale

Locale 객체는 특정 지리적, 정치적 또는 문화적 영역을 나타냅니다. 로케일이 작업을 수행하는 데 필요한 작업을 로케일 구분이라고 하며 로케일을 사용하여 사용자 정보를 조정합니다.

다음 표는 Locale 클래스의 주요 생성자입니다.

설명
Locale(String language) 주어진 언어코드로 로케일 객체를 생성합니다.
Locale(String language, String country) 주어진 언어와 국가코드로 로케일 객체를 생성합니다.

다음 코드는 Locale을 지정하는 예입니다. Locale은 단독으로 사용되는 경우는 많지 않습니다. 뒤에서 설명하는 형식화 예에서 NumberFormat 클래스 등과 같이 사용되는 경우가 많습니다.

LocaleExample.java

```
1. import java.text.DecimalFormat;
2. import java.text.NumberFormat;
3. import java.util.Locale;
4.
5. public class LocaleExample {
6.
7.  static public void localizedFormat(String pattern, double value, Locale
   loc ) {
8.     NumberFormat nf = NumberFormat.getNumberInstance(loc);
9.     DecimalFormat df = (DecimalFormat)nf;
10.    df.applyPattern(pattern);
11.    String output = df.format(value);
12.    System.out.println(pattern + "  " + output + "  " + loc.toString());
13. }
14.
15. public static void main(String[] args) {
16.    Locale locale = new Locale("ko", "KR");
17.    System.out.println(locale.getLanguage());
18.    System.out.println(locale.getCountry());
19.    System.out.println(locale.getDisplayCountry());
20.
21.    Locale locale2 = new Locale("en", "US");
22.    System.out.println(locale2.getLanguage());
23.    System.out.println(locale2.getCountry());
```

```
24.    System.out.println(locale2.getDisplayCountry());
25.
26.    System.out.println();
27.    Locale[] locales = {
28.          new Locale("en", "US"),
29.          new Locale("de", "DE"),
30.          new Locale("fr", "FR"),
31.          new Locale("ko", "KR")
32.    };
33.
34.    for (int i = 0; i < locales.length; i++) {
35.       localizedFormat("###,###.###", 123456.789, locales[i]);
36.    }
37. }//end main
38.
39.}
```

```
Console
<terminated> LocaleExample [Java Application] C:₩Program
ko
KR
대한민국
en
US
미국
###,###.###   123,456.789   en_US
###,###.###   123.456,789   de_DE
###,###.###   123 456,789   fr_FR
###,###.###   123,456.789   ko_KR
```

14.5.2. Timezone

Timezone 객체는 시간대 오프셋을 나타냅니다. 일반적으로 프로그램이 실행되는 시간대를 기반으로 TimeZone을 만드는 getDefault() 메서드를 사용하여 TimeZone을 얻습니다. 예를 들면 한국에서 실행되는 프로그램의 경우 getDefault() 메서드는 한국 표준시를 기반으로 TimeZone 객체를 만듭니다. getTimeZone() 메서드는 지정한 시간대의 TimeZone을 얻습니다.

다음 표는 TimeZone 클래스 객체를 반환하는 메서드입니다.

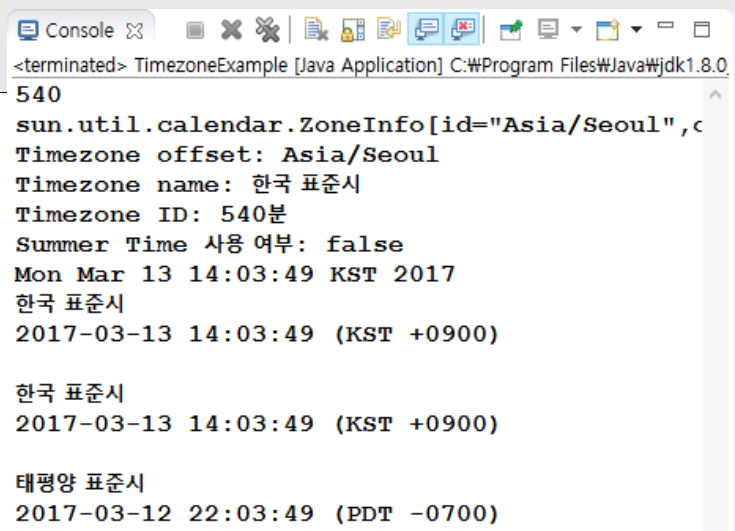
	메서드와 설명
static TimeZone	getTimeZone(String ID) 타임존 ID 문자열로 타임존 객체를 생성합니다.
static TimeZone	getTimeZone(ZoneId zoneId) 주어진 ZoneId 객체 타임존 객체를 생성합니다.

다음 코드는 TimeZone 클래스를 이용하여 표준시간대와의 오프셋을 출력합니다. 그리고 디폴트 타임존을 이용할 경우와 타임존을 지정할 경우에 출력되는 날짜 값이 어떻게 다른지를 보여주는 예입니다.

TimezoneExample.java

```
1. import java.text.DateFormat;
2. import java.text.SimpleDateFormat;
3. import java.util.Calendar;
4. import java.util.Date;
5. import java.util.TimeZone;
6.
```

```
7. public class TimezoneExample {
8.
9.    public static void main(String[] args) {
10.    Calendar calendar = Calendar.getInstance();
11.    System.out.println( (calendar.get(Calendar.ZONE_OFFSET) +
12.          calendar.get(Calendar.DST_OFFSET)) / (60 * 1000));
13.
14.       TimeZone tz = TimeZone.getDefault();
15.       System.out.println(tz);
16.       System.out.println("Timezone offset: " + tz.getID());
17.       System.out.println("Timezone name: " + tz.getDisplayName());
18.       System.out.println("Timezone ID: " + (tz.getRawOffset()/1000/60) +
   "분");
19.       System.out.println("Summer Time 사용 여부: " + tz.useDaylightTime());
20.
21.       Date date = new Date();
22.       System.out.println(date);
23.
24.       DateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss (z Z)");
25.       System.out.format("%s%n%s%n%n", tz.getDisplayName(),
   df.format(date));
26.
27.       TimeZone tz2 = TimeZone.getTimeZone("Asia/Seoul");
28.       df.setTimeZone(tz2);
29.       System.out.format("%s%n%s%n%n", tz2.getDisplayName(),
   df.format(date));
30.
31.       TimeZone tz3 = TimeZone.getTimeZone("America/Los_Angeles");
32.       df.setTimeZone(tz3);
33.       System.out.format("%s%n%s%n%n", tz3.getDisplayName(),
   df.format(date));
34.    }
35.
36.}
```



14.6. 클래스

숫자를 원하는 형식의 문자열로 출력하고 싶을 때 사용하는 것이 형식화(Format) 클래스들입니다. 형식화 클래스를 이용하면 숫자에 콤마(,)를 표기하거나 원하는 자리까지 출력하는 일, 날짜를 원하는 형식의 날짜로 출력하는 것을 쉽게 처리할 수 있습니다.

형식화 클래스들은 java.text 패키지에 정의되어 있습니다. 대표적인 형식화 클래스들로는 숫자 형식화를 위한 DecimalFormat과 날짜 형식화를 위한 SimpleDateFormat 그리고 문자열 형식화를 위한 MessageFormat등이 있습니다.

14.6.1. DecimalFormat

DecimalFormat 클래스는 NumberFormat 추상 클래스를 구현한 클래스들 중 하나입니다. DecimalFormat 클래스는 Western, Arabic 및 Indic 숫자에 대한 지원을 포함하여 모든 지역에서 숫자를 구문 분석하고 형식을 지정할 수 있도록 설계된 다양한 기능이 있습니다. 또한 정수(123), 고정 소수점 수(123.4), 과학 표기법(1.23E4), 백분율(12%) 및 통화 금액($123)을 비롯한 여러 가지 종류의 숫자를 지원합니다. 이들 모두는 지역화(Locale) 될 수 있습니다.

DecimalFormat 클래스는 숫자를 보기 좋게 표현해 주는 것만 아닙니다. 지역화를 적용하여 사용자의 실수를 줄일 수 있게 도와주기도 합니다. 이것은 매우 중요합니다. 만일 여러분이 독일에 123.45유로를 송금해야 한다고 가정해 봅시다. 만일 평소에 우리가 표기하던 방법으로 송금액을 표기(123.450)한다고 하면 여러분의 통장에서 12만3천450유로가 빠져 나갈 것입니다.[40]

DecimalFormat은 숫자 데이터를 원하는 형식으로 표현하기 위해 패턴을 지정해야 합니다. 다음 표는 패턴들의 예를 설명합니다.

	위치	지역화	의미
0	숫자	가능	숫자
#	숫자	가능	숫자, 0은 표시하지 않습니다.
.	숫자	가능	소수 구분 기호 또는 화폐 소수 구분 기호
-	숫자	가능	음수 기호
,	숫자	가능	그룹화 구분 기호

40) 독일은 소수점을 ,로 표기하고 정수 자릿수 구분자를 .을 이용합니다.

E		가능	과학 표기법으로 가수와 지수를 구분합니다. 접두어 또는 접미사에 따옴표를 쓸 필요가 없습니다.
;	서브 패턴 경계	가능	양수 및 음수 하위 패턴 구분
%	접두어 또는 접미어	가능	100을 곱하고 백분율로 표시하세요.
₩u2030	접두어 또는 접미어	가능	1000을 곱하고 퍼밀리 값으로 표시하세요.
¤ (₩u00A4)	접두어 또는 접미어	불가능	통화 기호, 통화 기호로 대체되었습니다. 두 배가되면 국제 통화 기호로 바뀝니다. 패턴에 있으면 소수 구분 기호 대신 화폐 소수 구분 기호가 사용됩니다.
'	접두어 또는 접미어	불가능	접두어와 접미어에 특수 문자를 사용할 때 '로 묶어줍니다.(예: 123을 "'#'#" 포맷으로 표현하면 "#123") '(a single quote) 자신을 사용하려면 '를 하나 더 표기합니다.(예: "# o''clock")

다음 코드는 DecimalFormat을 이용하여 주어진 숫자를 다양한 형식으로 표현하는 예제입니다.

DecimalFormatExample.java

```
1:  import java.text.DecimalFormat;
2:  import java.text.NumberFormat;
3:  import java.util.Locale;
4:
5:  public class DecimalFormatExample {
6:
7:     static public void customFormat(String pattern, double value ) {
8:        DecimalFormat myFormatter = new DecimalFormat(pattern);
9:        String output = myFormatter.format(value);
10:       System.out.println(value + "\t" + pattern + "\t" + output);
11:    }
12:
13:    static public void main(String[] args) {
14:       customFormat("###,###.###", 123456.789);
15:       customFormat("###.##", 123456.789);
16:       customFormat("000000.000", 12345.67);
17:       customFormat("$###,###.###", 12345.67);
18:       customFormat("\u00a5###,###.###", 12345.67);
19:       customFormat("\u20a9###,###.###", 12345.67);
20:
21:       DecimalFormat deFormatter =
    (DecimalFormat)NumberFormat.getNumberInstance(new Locale("de", "DE"));
22:       deFormatter.applyPattern("###,###.###");
23:       String output = deFormatter.format(123.450);
24:       System.out.println(123.450  \t###,###.### \t" + output);
25:    }
26: }
```

```
Console
<terminated> DecimalFormatExample [Java Application] C:₩Program Files₩Java₩
123456.789      ###,###.###     123,456.789
123456.789      ###.##          123456.79
12345.67        000000.000      012345.670
12345.67        $###,###.###    $12,345.67
12345.67        ¥###,###.###    ¥12,345.67
12345.67        ₩###,###.###    ₩12,345.67
123.450         ###,###.###     123,45
```

14.6.2. SimpleDateFormat

SimpleDateFormat은 로케일에 따라 날짜 형식을 지정하고 구문 분석하는 데 필요한 구체적인 클래스입니다. SimpleDateFormat을 이용하면 날짜를 텍스트로 형식화하는 것, 텍스트를 날짜 객체로 구문 분석하는 것, 그리고 정규화를 할 수 있습니다.

다음은 날짜와 시간 패턴 문자들에 대한 설명입니다.

	Date 또는 Time Component	표현	예
G	지정자	텍스트	AD
y	연도	연도	1996; 96
Y	연도	연도	2009; 09
M	월	월	July; Jul; 07
w	연도에서 주	숫자	27
W	월에서 주	숫자	2
D	연도에서 날짜	숫자	189
d	월에서 날짜	숫자	10
F	월에서 요일	숫자	2
E	주에서 요일 이름	텍스트	Tuesday; Tue
u	주에서 요일의 수(1 = Monday, ..., 7 = Sunday)	숫자	1
a	오전/오후 표시	텍스트	PM
H	시간(0-23)	숫자	0
k	시간(1-24)	숫자	24
K	오전/오후 시간(0-11)	숫자	0
h	오전/오후 시간(1-12)	숫자	12
m	분	숫자	30
s	초	숫자	55
S	밀리초	숫자	978
z	타임존	일반 타임존	Pacific Standard Time; PST; GMT-08:00
Z	타임존	RFC 822 타임존	-0800
X	타임존	ISO 8601 타임존	-08; -0800; -08:00

다음 표는 날짜와 시간 패턴이 kr_KR로케일과 en_US 로케일에서 각각 어떻게 변환되는지 보여주고 있습니다. 주어진 날짜와 시간은 Asia/Seoul 타임존에서 한국 표준시 2017년 3월 13일 오후 10시 16분 55초입니다.

시간 패턴	결과(ko_KR 로케일)
"yyyy.MM.dd G 'at' HH:mm:ss z"	2017.03.13 at 22:16:55 KST
"EEE, MMM d, ''yy"	월, 3월 13, '17
"h:mm a"	10:16 오후
"hh 'o''clock' a, zzzz"	10 o'clock 오후, 한국 표준시
"K:mm a, z"	10:16 오후, KST
"yyyyy.MMMMM.dd GGG hh:mm aaa"	02017.3월.13 서기 10:16 오후
"EEE, d MMM yyyy HH:mm:ss Z"	월, 13 3월 2017 22:16:55 +0900
"yyMMddHHmmssZ"	170313221655+0900
"yyyy-MM-dd'T'HH:mm:ss.SSSZ"	2017-03-13T22:16:55.266+0900
"yyyy-MM-dd'T'HH:mm:ss.SSSXXX"	2017-03-13T22:16:55.266+09:00
"YYYY-'W'ww-u"	2017-W11-1

날짜와 시간 패턴	결과(en_US 로케일)
"yyyy.MM.dd G 'at' HH:mm:ss z"	2017.03.13 AD at 22:16:55 KST
"EEE, MMM d, ''yy"	Mon, Mar 13, '17
"h:mm a"	10:16 PM
"hh 'o''clock' a, zzzz"	10 o'clock PM, Korea Standard Time
"K:mm a, z"	10:16 PM, KST
"yyyyy.MMMMM.dd GGG hh:mm aaa"	02017.March.13 AD 10:16 PM
"EEE, d MMM yyyy HH:mm:ss Z"	Mon, 13 Mar 2017 22:16:55 +0900
"yyMMddHHmmssZ"	170313221655+0900
"yyyy-MM-dd'T'HH:mm:ss.SSSZ"	2017-03-13T22:16:55.266+0900
"yyyy-MM-dd'T'HH:mm:ss.SSSXXX"	2017-03-13T22:16:55.266+09:00
"YYYY-'W'ww-u"	2017-W11-1

다음 코드는 날짜와 시간 패턴 예를 보여주고 있습니다. 실행 시 날짜와 시간은 Asia/Seoul 타임존 에서 한국 표준시 2017년 3월 13일 오후 10시 16분 55초입니다.

DateFormatExample.java

```
1:  import java.text.DateFormat;
2:  import java.text.SimpleDateFormat;
3:  import java.util.Calendar;
4:  import java.util.Date;
5:  import java.util.Locale;
6:  import java.util.TimeZone;
7:
8:  public class DateFormatExample {
9:
```

```
10:     static void displayDateAndTime(String format, Locale locale, Date date) {
11:         DateFormat df = new SimpleDateFormat(format, locale);
12:         System.out.format("%s : %s%n", format, df.format(date));
13:     }
14:     public static void main(String[] args) {
15:         Date date = Calendar.getInstance().getTime();
16:         Locale koLocale = new Locale("ko", "KR");
17:         TimeZone tz = TimeZone.getTimeZone("Asia/Seoul");
18:         System.out.println(tz.getDisplayName() + " : " + date);
19:         displayDateAndTime("yyyy.MM.dd G 'at' HH:mm:ss z", koLocale, date);
20:         displayDateAndTime("EEE, MMM d, ''yy", koLocale, date);
21:         displayDateAndTime("h:mm a", koLocale, date);
22:         displayDateAndTime("hh 'o''clock' a, zzzz", koLocale, date);
23:         displayDateAndTime("K:mm a, z", koLocale, date);
24:         displayDateAndTime("yyyyy.MMMMM.dd GGG hh:mm aaa", koLocale, date);
25:         displayDateAndTime("EEE, d MMM yyyy HH:mm:ss Z", koLocale, date);
26:         displayDateAndTime("yyMMddHHmmssZ", koLocale, date);
27:         displayDateAndTime("yyyy-MM-dd'T'HH:mm:ss.SSSZ", koLocale, date);
28:         displayDateAndTime("yyyy-MM-dd'T'HH:mm:ss.SSSXXX", koLocale, date);
29:         displayDateAndTime("YYYY-'W'ww-u", koLocale, date);
30:
31:         System.out.println();
32:         Locale usLocale = new Locale("en", "US");
33:         System.out.println(tz.getDisplayName() + " : " + date);
34:         displayDateAndTime("yyyy.MM.dd G 'at' HH:mm:ss z", usLocale, date);
35:         displayDateAndTime("EEE, MMM d, ''yy", usLocale, date);
36:         displayDateAndTime("h:mm a", usLocale, date);
37:         displayDateAndTime("hh 'o''clock' a, zzzz", usLocale, date);
38:         displayDateAndTime("K:mm a, z", usLocale, date);
39:         displayDateAndTime("yyyyy.MMMMM.dd GGG hh:mm aaa", usLocale, date);
40:         displayDateAndTime("EEE, d MMM yyyy HH:mm:ss Z", usLocale, date);
41:         displayDateAndTime("yyMMddHHmmssZ", usLocale, date);
42:         displayDateAndTime("yyyy-MM-dd'T'HH:mm:ss.SSSZ", usLocale, date);
43:         displayDateAndTime("yyyy-MM-dd'T'HH:mm:ss.SSSXXX", usLocale, date);
44:         displayDateAndTime("YYYY-'W'ww-u", usLocale, date);
45:     }
46: }
```

Console

<terminated> DateFormatExample [Java Application] C:₩Program Files₩Java₩jre1.8.0_121₩bin₩javaw.e

```
한국 표준시 : Mon Mar 13 22:16:55 KST 2017
yyyy.MM.dd G 'at' HH:mm:ss z : 2017.03.13 서기 at 22:16:55 KST
EEE, MMM d, ''yy : 월, 3월 13, '17
h:mm a : 10:16 오후
hh 'o''clock' a, zzzz : 10 o'clock 오후, 한국 표준시
K:mm a, z : 10:16 오후, KST
yyyyy.MMMMM.dd GGG hh:mm aaa : 02017.3월.13 서기 10:16 오후
EEE, d MMM yyyy HH:mm:ss Z : 월, 13 3월 2017 22:16:55 +090
yyMMddHHmmssZ : 170313221655+0900
yyyy-MM-dd'T'HH:mm:ss.SSSZ : 2017-03-13T22:16:55.266+090
yyyy-MM-dd'T'HH:mm:ss.SSSXXX : 2017-03-13T22:16:55.266+0
YYYY-'W'ww-u : 2017-W11-1
```

```
한국 표준시 : Mon Mar 13 22:16:55 KST 2017
yyyy.MM.dd G 'at' HH:mm:ss z : 2017.03.13 AD at 22:16:55 KST
EEE, MMM d, ''yy : Mon, Mar 13, '17
h:mm a : 10:16 PM
hh 'o''clock' a, zzzz : 10 o'clock PM, Korea Standard Time
K:mm a, z : 10:16 PM, KST
yyyyy.MMMMM.dd GGG hh:mm aaa : 02017.March.13 AD 10:16 PM
EEE, d MMM yyyy HH:mm:ss Z : Mon, 13 Mar 2017 22:16:55 +0900
yyMMddHHmmssZ : 170313221655+0900
yyyy-MM-dd'T'HH:mm:ss.SSSZ : 2017-03-13T22:16:55.266+0900
yyyy-MM-dd'T'HH:mm:ss.SSSXXX : 2017-03-13T22:16:55.266+09:00
YYYY-'W'ww-u : 2017-W11-1
```

14.6.3. MessageFormat

MessageFormat 클래스는 언어에 중립적인 방법으로 문자열들이 연결된 메시지를 생성하는 수단을 제공합니다. MessageFormat은 최종 사용자에게 표시되는 메시지를 생성 할 때 사용합니다.

MessageFormat은 객체 세트를 가져 와서 형식을 지정한 다음 서식이 지정된 문자열을 적절한 위치의 패턴에 삽입합니다.

MessageFormat는 다른 Format 클래스와 달리, getInstance() 스타일 팩토리 메서드[41]가 아닌 생성자 중 하나를 사용하여 MessageFormat 객체를 생성합니다. MessageFormat 자체가 로케일 고유의 동작을 구현하지 않기 때문에 팩토리 메서드가 필요하지 않습니다. 모든 로케일 특정 동작은 사용자가 제공한 패턴과 삽입 된 인수에 사용되는 하위 형식에 의해 정의됩니다.

MessageFormat은 다음 형식의 패턴을 사용합니다.

	형식 또는 값
MessageFormatPattern	String MessageFormatPattern FormatElement String
FormatElement	{ ArgumentIndex } { ArgumentIndex , FormatType } { ArgumentIndex , FormatType , FormatStyle }
FormatType	number, date, time, choice 하나
FormatStyle:	short, medium, long, full, integer, currency, percent, SubformatPattern

다음 코드는 MessageFormat을 이용하여 문자열에 형식을 지정하여 연결된 메시지를 생성하는 예입니다.

MessageFormatExample.java

```
1:  import java.text.MessageFormat;
2:  import java.util.Date;
3:
4:  public class MessageFormatExample {
5:     public static void main(String[] args) {
6:        new MessageFormatExample();
7:     }
8:
```

41) getInstance() 메서드를 갖는 대표적인 클래스는 Calendar 클래스입니다. getInstance() 메서드는 객체를 생성하여 반환하는 용도입니다. 이처럼 객체를 생성하기 위해 사용하는 메서드들을 팩토리 메서드(Factory Method)라 부릅니다.

```
9:     public MessageFormatExample() {
10:        Object[] args = {"foo", "bar", new Long(99)};
11:        MessageFormat mf = new MessageFormat("1:{0} 2:{1} 3:{2}");
12:        System.out.println(mf.format(args));
13:
14:        int meetingTime = 10;
15:        String event = "          면접 심사";
16:
17:        String result = MessageFormat.format(
18:              "오늘({0, date}) {1,number,integer}시에 {2}가 있습니다.",
19:              new Date(), meetingTime, event);
20:        System.out.println(result);
21:    }
22: }
```

```
Console
<terminated> MessageFormatExample [Java Application] C:₩Program Files₩Java₩j
1:foo 2:bar 3:99
오늘(2017. 3. 14) 10시에 대회의실에서 면접 심사가 있습니다.
```

14.6.4. ChoiceFormat

ChoiceFormat을 사용하면 숫자 범위에 형식을 첨부 할 수 있습니다. 일반적으로는 여러 개 형식을 처리하기 위해 MessageFormat에서 사용됩니다. 선택 항목은 double 유형 리스트의 오름차순으로 지정되며, 각 항목은 다음 항목까지의 반 개방(half open) 간격을 지정합니다.

ChoiceFormat는 다른 Format 클래스와 달리, getInstance 스타일 팩토리 메서드가 아닌 생성자를 사용하여 ChoiceFormat 객체를 만듭니다. ChoiceFormat는 지정된 로케일에 대해 복잡한 설정이 필요 없기 때문에 팩토리 메서드가 필요하지 않습니다.

ChoiceFormat를 작성할 때는 제한(limit) 배열과 형식(format) 배열을 지정해야합니다. 제한 배열과 형식 배열의 길이는 동일해야합니다.

다음 코드는 ChoiceFormat을 이용하여 메시지를 선택하는 예입니다.

ChoiceFormatExample1.java

```
1:  import java.text.ChoiceFormat;
2:  import java.text.ParsePosition;
3:
4:  public class ChoiceFormatExample1 {
5:
```

```
6:     public static void main(String[] args) {
7:        double[] limits = {1,2,3,4,5,6,7};
8:        String[] dayOfWeekNames = {"Sun","Mon","Tue","Wed","Thur","Fri","Sat"};
9:        ChoiceFormat choiceForm = new ChoiceFormat(limits, dayOfWeekNames);
10:       ParsePosition status = new ParsePosition(0);
11:       for (double i = 0.0; i <= 8.0; ++i) {
12:          status.setIndex(0);
13:          System.out.println(i + " -> " + choiceForm.format(i) + " -> "
14:                + choiceForm.parse(choiceForm.format(i),status));
15:       }
16:    }
17: }
```

```
<terminated> ChoiceFormatExa
0.0 -> Sun -> 1.0
1.0 -> Sun -> 1.0
2.0 -> Mon -> 2.0
3.0 -> Tue -> 3.0
4.0 -> Wed -> 4.0
5.0 -> Thur -> 5.0
6.0 -> Fri -> 6.0
7.0 -> Sat -> 7.0
8.0 -> Sat -> 7.0
```

다음 코드는 MessageFormat과 ChoiceFormat을 같이 사용한 예입니다. appleForm 객체를 이용하여 사과의 개수에 따른 선택 항목이 지정되도록 했습니다. 만일 사과의 개수가 2개 이상일 경우에는 사과의 개수가 출력되도록 MessageFormat을 같이 사용했습니다. MessageFormat 클래스의 setFormatByArgumentIndex() 메서드는 format() 메서드 인자로 전달되는 배열의 값 들 중 ChoiceFormat의 limit 값으로 전달되어야 할 항목의 인덱스를 지정합니다.

ChoiceFormatExample2.java

```
1:  import java.text.ChoiceFormat;
2:  import java.text.MessageFormat;
3:
4:  public class ChoiceFormatExample2 {
5:     public static void main(String[] args) {
6:
7:        MessageFormat msgForm = new MessageFormat("{0}  바구니에  사과가 {1}");
8:
9:        double[] appleLimits = {0,1,2};
10:       String[] applePart = {"없습니다.","1개 있습니다.","{1,number}개 있습니다."};
11:       ChoiceFormat appleForm = new ChoiceFormat(appleLimits, applePart);
12:
13:       msgForm.setFormatByArgumentIndex(1, appleForm);
14:
15:       System.out.println(msgForm.format(new Object[]{"그", new Long(2)}));
16:       System.out.println(msgForm.format(new Object[]{"그녀", new Long(0)}));
17:       System.out.println(msgForm.format(new Object[]{"나", new Long(1)}));
18:       System.out.println(msgForm.format(new Object[]{"너", new Long(5)}));
19:    }
20: }
```

```
Console
<terminated> ChoiceFormatExample2 [Java
그의 바구니에  사과가 2개 있습니다.
그녀의 바구니에  사과가 없습니다.
나의 바구니에  사과가 1개 있습니다.
너의 바구니에  사과가 5개 있습니다.
```

14.7. 관련 클래스

14.7.1. Random

난수를 발생시키기 위해 Math.random() 메서드를 사용할 수 있습니다. 그러나 Random 클래스를 이용하면 보다 쉽고 직관적으로 난수를 발생시킬 수 있습니다.

Random 클래스의 인스턴스는 난수를 발생시키기 위해서 사용됩니다. 생성자는 seed를 지정하지 않고 객체를 생성하는 기본 생성자와 long 형 seed값을 인자로 갖는 생성자를 가지고 있습니다. 다음 표는 Random 클래스의 생성자입니다.

설명
Random() 난수 발생 객체를 생성합니다.
Random(long seed) 시드 값을 사용한 난수 발생 객체를 생성합니다.

만일 두 인스턴스가 동일한 시드(seed)로 만들어지고 각각에 대해 동일한 순서로 난수를 발생시키는 메서드가 호출되면 두 객체는 동일한 순서로 난수 시퀀스가 생성되고 반환됩니다.

다음 표는 Random 클래스의 주요 메서드들입니다.

	메서드와 설명
boolean	nextBoolean() boolean 타입 임의의 값을 반환합니다.(true 또는 false)
void	nextBytes(byte[] bytes) 임의의 바이트를 생성하여 주어진 bytes 배열에 저장합니다.
double	nextDouble() 0.0부터 1.0미만 임의의 값을 double 형으로 생성합니다.
float	nextFloat() 0.0부터 1.0미만 임의의 값을 float 형으로 생성합니다.
int	nextInt() int 형 임의의 값을 생성합니다.
int	nextInt(int bound) 0부터 bound 미만의 값을 int 형으로 반환합니다.
long	nextLong() long 형 임의의 값을 생성합니다.
void	setSeed(long seed) seed 값을 설정합니다.

다음 코드는 난수를 발생하는 예입니다.

RandomClassExample.java

```
1:  import java.util.Random;
2:
3:  public class RandomClassExample {
4:     public static void main(String[] args) {
5:        int numTests = 10;
6:
7:        Random random = new Random();
8:        for ( int i=0; i<numTests; i++ ) {
9:           int randomInt = random.nextInt(100);
10:          System.out.format("테스트 %2d, 발생 난수 %d\n", i+1, randomInt );
11:       }
12:    }
13: }
```

```
Console
<terminated> RandomClassExam
테스트  1, 발생 난수 8
테스트  2, 발생 난수 45
테스트  3, 발생 난수 41
테스트  4, 발생 난수 29
테스트  5, 발생 난수 87
테스트  6, 발생 난수 16
테스트  7, 발생 난수 13
테스트  8, 발생 난수 16
테스트  9, 발생 난수 52
테스트 10, 발생 난수 68
```

14.7.2. Math

Math 클래스에는 지수, 로그, 제곱근 및 삼각 함수와 같은 기본 숫자 연산을 수행하기 위한 메서드가 포함되어 있습니다. Math 클래스의 멤버들은 모두 static 으로 선언되어 있기 때문에 객체를 생성하지 않고 클래스 이름만으로 멤버를 참조할 수 있습니다.

다음 표는 Math 클래스의 주요 메서드들입니다. Math 클래스는 java.lang 패키지에 정의되어 있습니다.

	메서드와 설명
static double	abs(double a) 절대 값을 반환합니다. float, int, long 형 파라미터를 갖는 메서드도 중복 정의되어 있습니다.
static double	ceil(double a) 소수점 이하를 올림 한 값을 반환합니다.
static double	cos(double a) cos 값을 반환합니다. a 값은 라디안(radian)[42] 값으로 입력해야 합니다.
static double	floor(double a) 내림 한 값을 반환 합니다.
static double	log(double a) log 값은 반환합니다.
static double	log10(double a) base가 10인 로그 값을 반환합니다.
static double	max(double a, double b) 두 수 중에서 큰 값을 반환합니다. float, int, long 형 파라미터를 갖는 메서드도 중복 정의되어 있습니다.

static double	min(double a, double b) 수 중에서 작은 값을 반환합니다. float, int, long 형 파라미터를 갖는 메서드도 중복 정의되어 있습니다.
static double	pow(double a, double b) a의 b승수를 반환합니다.
static double	random() 0.0이상 1.0 미만의 난수를 반환합니다.
static long	round(double a) 반올림 합니다.
static int	round(float a) 반올림 합니다.
static double	sin(double a) sin 값을 반환합니다. a 값은 라디안(radian) 값으로 입력해야 합니다.
static double	tan(double a) tan 값을 반환합니다. a 값은 라디안(radian) 값으로 입력해야 합니다.

다음은 Math 클래스를 테스트하는 예입니다.

MathClassExample.java

```
1:  public class MathClassExample {
2:     public static void main(String[] args) {
3:        System.out.println(Math.random());
4:        System.out.println(Math.sin(30*(Math.PI/180)));
5:        System.out.println(Math.cos(60*(Math.PI/180)));
6:        System.out.println(Math.tan(45*(Math.PI/180)));
7:        System.out.println(Math.pow(2, 3));
8:        System.out.printf("%d %d %d %d\n", (int)-3.6, (int)-3.4, (int)3.5,
    (int)3.6);
9:        System.out.printf("%d %d %d %d\n", Math.round(-3.6), Math.round(-3.4),
    Math.round(3.5), Math.round(3.6));
10:       System.out.printf("%f %f %f %f\n", Math.ceil(-3.6), Math.ceil(-3.4),
    Math.ceil(3.5), Math.ceil(3.6));
11:       System.out.printf("%f %f %f %f\n", Math.floor(-3.6), Math.floor(-3.4),
    Math.floor(3.5), Math.floor(3.6));
12:    }
13: }
```

```
Console
<terminated> MathClassExample [Java Application] C:₩Program Files₩
0.18817897238857384
0.49999999999999994
0.5000000000000001
0.9999999999999999
8.0
-3 -3 3 3
-4 -3 4 4
-3.000000 -3.000000 4.000000 4.000000
-4.000000 -4.000000 3.000000 3.000000
```

42) Radian = Degree * PI/180