

제13장

자바API클래스

구디아카데미 ▷ 민경태 강사

학습목표

1. java.lang 패키지에 저장된 클래스에 대해서 알 수 있다.
2. java.util 패키지에 저장된 클래스에 대해서 알 수 있다.
3. java.text 패키지에 저장된 클래스에 대해서 알 수 있다.
4. java.time 패키지에 저장된 클래스에 대해서 알 수 있다.

```
each: function(e, t, n) {  
    i = 0,  
    o = e.length,  
    a = M(e);  
    if (n) {  
        if (a) {  
            for (; o > i; i++)  
                if (r = t.apply(e[i], n), r ===  
        } else  
            for (i in e)  
                if (r = t.apply(e[i], n), r ===  
    } else if (a) {  
        for (; o > i; i++)  
            if (r = t.call(e[i], i, e[i])  
    } else  
        for (i in e)  
            if (r = t.call(e[i], i, e[i])  
    return e  
},  
trim: b && !b.call("\uffff\u00a0") ?  
    return null == e ? "" : b.call(  
} : function(e) {  
    return null == e ? "" : (e +  
},  
makeArray: function(e, t) {  
    var n = t || [];  
    return null != e && (M(Obj  
},  
isArray: function(e, t, n) {  
    var r;  
    if (t) {  
        if (m) return m.c  
        for (r = t.length  
            if (n in t  
    }  
}
```

목차

1. java.lang 패키지

- 1) System 클래스
- 2) String 클래스
- 3) StringBuilder 클래스
- 4) Math 클래스

2. java.util 패키지

- 1) Calendar 클래스
- 2) Date 클래스
- 3) Optional 클래스
- 4) Scanner 클래스
- 5) UUID 클래스

목차

3. java.text 패키지

- 1) SimpleDateFormat 클래스
- 2) DecimalFormat 클래스

4. java.time 패키지

- 1) LocalDate 클래스
- 2) LocalTime 클래스
- 3) LocalDateTime 클래스
- 4) DateTimeFormatter 클래스

```

each: function(e, t, n) {
    var r, i = 0,
        o = e.length,
        a = M(e);
    if (n) {
        if (a) {
            for (; o > i; i++)
                if (r = t.apply(e[i], n), r === !1) break;
        } else
            for (i in e)
                if (r = t.apply(e[i], n), r === !1) break;
    } else if (a) {
        for (; o > i; i++)
            if (r = t.call(e[i], , e[i]), r === !1) break;
    } else
        for (i in e)
            if (r = t.call(e[i], , e[i]), r === !1) break;
    return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
    return null == e ? "" : b.call(e);
} : function(e) {
    return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
    var n = t || [];
    return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
    var r;
    if (t) {
        if (n) return m.call(t, e, n);
        for (r = t.length, r = n ? 0 > n ? Math.max(0, r + n) : r; r-- && t[r] !== e) return n;
    }
}

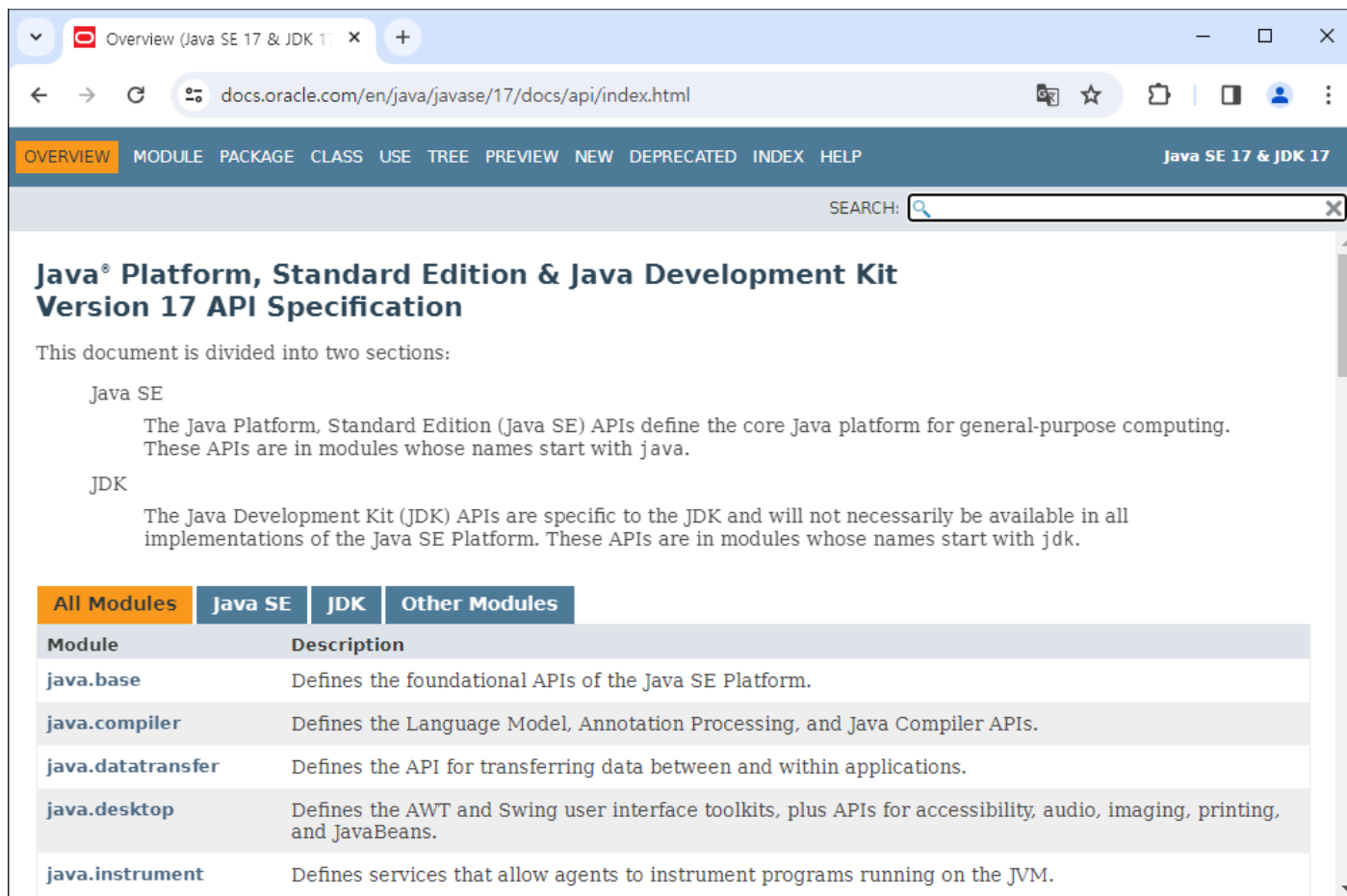
```

1. java.lang 패키지

온라인 자바 API 문서

- 오라클에서 제공하는 자바 API 문서

<https://docs.oracle.com/en/java/javase/17/docs/api/index.html>



Overview (Java SE 17 & JDK 17) x +

docs.oracle.com/en/java/javase/17/docs/api/index.html

OVERVIEW MODULE PACKAGE CLASS USE TREE PREVIEW NEW DEPRECATED INDEX HELP Java SE 17 & JDK 17

SEARCH:

Java® Platform, Standard Edition & Java Development Kit Version 17 API Specification

This document is divided into two sections:

Java SE
The Java Platform, Standard Edition (Java SE) APIs define the core Java platform for general-purpose computing. These APIs are in modules whose names start with `java`.

JDK
The Java Development Kit (JDK) APIs are specific to the JDK and will not necessarily be available in all implementations of the Java SE Platform. These APIs are in modules whose names start with `jdk`.

All Modules	Java SE	JDK	Other Modules
Module	Description		
<code>java.base</code>	Defines the foundational APIs of the Java SE Platform.		
<code>java.compiler</code>	Defines the Language Model, Annotation Processing, and Java Compiler APIs.		
<code>java.datatransfer</code>	Defines the API for transferring data between and within applications.		
<code>java.desktop</code>	Defines the AWT and Swing user interface toolkits, plus APIs for accessibility, audio, imaging, printing, and JavaBeans.		
<code>java.instrument</code>	Defines services that allow agents to instrument programs running on the JVM.		

JDK 17 기준

java.lang 패키지

- Java 개발에 필요한 가장 기본적인 클래스들이 모인 패키지
- java.lang 패키지에 포함된 클래스와 인터페이스는 import 없이 사용
- java.lang 패키지의 주요 클래스
 - Object
 - System
 - Integer
 - String
 - StringBuilder
 - Math
 - Exception

System 클래스


- 운영체제 시스템과 관련된 기능을 제공하는 클래스
- 모든 멤버가 static 처리되어 있어 "System.멤버" 형식으로 호출함
- 표준 입출력에 필요한 입출력 스트림 객체를 가지고 있음
- 배열 복사, 타임스탬프* 확인 등 유용한 메소드를 가지고 있음

* 타임스탬프 : 1970년 1월 1일 0시부터 1/1000초마다 1씩 증가시킨 값

System 클래스 - 필드

■ System 클래스에 선언된 필드

필드	역할	호출 형식
static final PrintStream err	표준 에러 스트림	System.err
static final InputStream in	표준 입력 스트림	System.in
static final PrintStream out	표준 출력 스트림	System.out



```
System.out.println()  
System.out.print()  
System.out.printf()
```

System 클래스 - 메소드

■ System 클래스 주요 메소드

메소드	역할
<code>static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)</code>	배열 <code>src[srcPos]</code> 요소부터 <code>length</code> 개의 요소를 배열 <code>dest[destPos]</code> 요소로 복사
<code>long currentTimeMillis()</code>	현재 타임스탬프 값을 반환
<code>void gc()</code>	Garbage Collector 호출
<code>String getProperty(String name)</code>	시스템 변수 <code>name</code> 의 값 반환
<code>long nanoTime()</code>	실행중인 JVM의 Time Source 값(나노초)을 반환

System 클래스 - arraycopy() 메소드

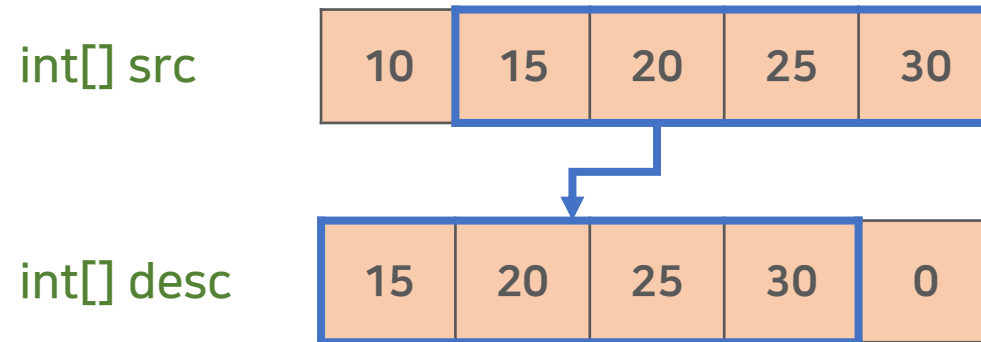
- 배열 src → 배열 desc로 복사하기

```
int[] src = {10, 15, 20, 25, 30};
```

```
int[] desc = new int[5];
```

```
System.arraycopy(src, 1, desc, 0, 4);
```

src[1]부터 4개 요소를 dest[0]으로 복사하시오.



System 클래스 - currentTimeMillis() 메소드

■ 시스템의 현재 타임스탬프 값 반환

```
long timestamp = System.currentTimeMillis();  
System.out.println(timestamp);
```

실행결과

1701162520603

타임스탬프 값

(1970-01-01부터 밀리초 단위로 카운트한 값)

실행할때마다 값이 달라진다.

■ 시간 단위

단위	해석	의미
ms	밀리초	1/1,000초 (천분의 일)
μs	마이크로초	1/1,000,000초 (백만분의 일)
ns	나노초	1/1,000,000,000초 (십억분의 일)
ps	피코초	1/1,000,000,000,000초 (조분의 일)

System 클래스 - nanoTime() 메소드

- 2개의 nanoTime() 사이의 차이값을 구하는 경우에만 의미가 있음
- 어떤 작업의 동작 시간을 측정하고자 할 때 사용함

```
long startTime = System.nanoTime();  
// ... 동작 시간을 측정해야 할 코드 ... //  
long elapsedTime = System.nanoTime() - startTime;  
System.out.println(elapsedTime);
```

실행결과

경과시간이
나노초 단위로 출력

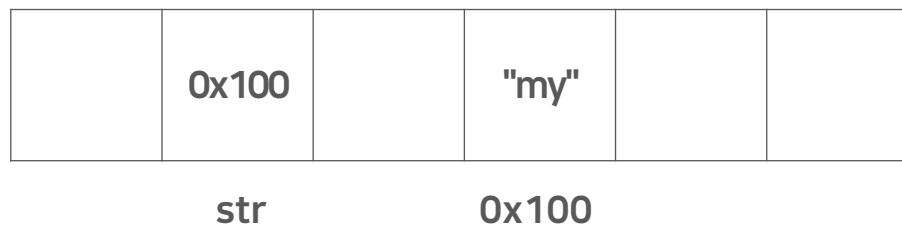
String 클래스

- 문자열을 처리하는 클래스
- String 클래스 내부의 final char[] 배열에 문자열을 저장함
 - final ▶ 한 번 생성된 문자열은 수정할 수 없음(Immutable Object)
 - char[] ▶ 글자마다 인덱스가 부여되어 있음(배열과 동일)
- 2가지 문자열 생성 방법이 있음
 - 문자열 리터럴
 - 문자열 객체

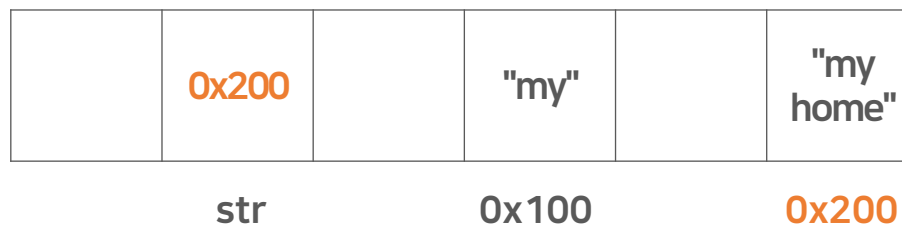
String 클래스 - 불변 객체

■ 불변 객체(Immutable Object) 이해하기

```
String str = "my";
```



```
str += "home";
```



문자열 "my"가 "myhome"으로 바뀌는 것이 아니라
새로운 "myhome"이 생기는 방식으로 동작한다.

String 클래스 - 문자열 리터럴

- 문자열 리터럴
 - 큰 따옴표("")를 이용해 나타낸 문자열 정보
- 문자열 리터럴은 JVM이 관리함
- 문자열 리터럴은 String Constant Pool 영역에 저장됨
- JVM은 이전 문자열 리터럴과 동일한 문자열 리터럴이 발생하면 기존 문자열 리터럴을 재사용함

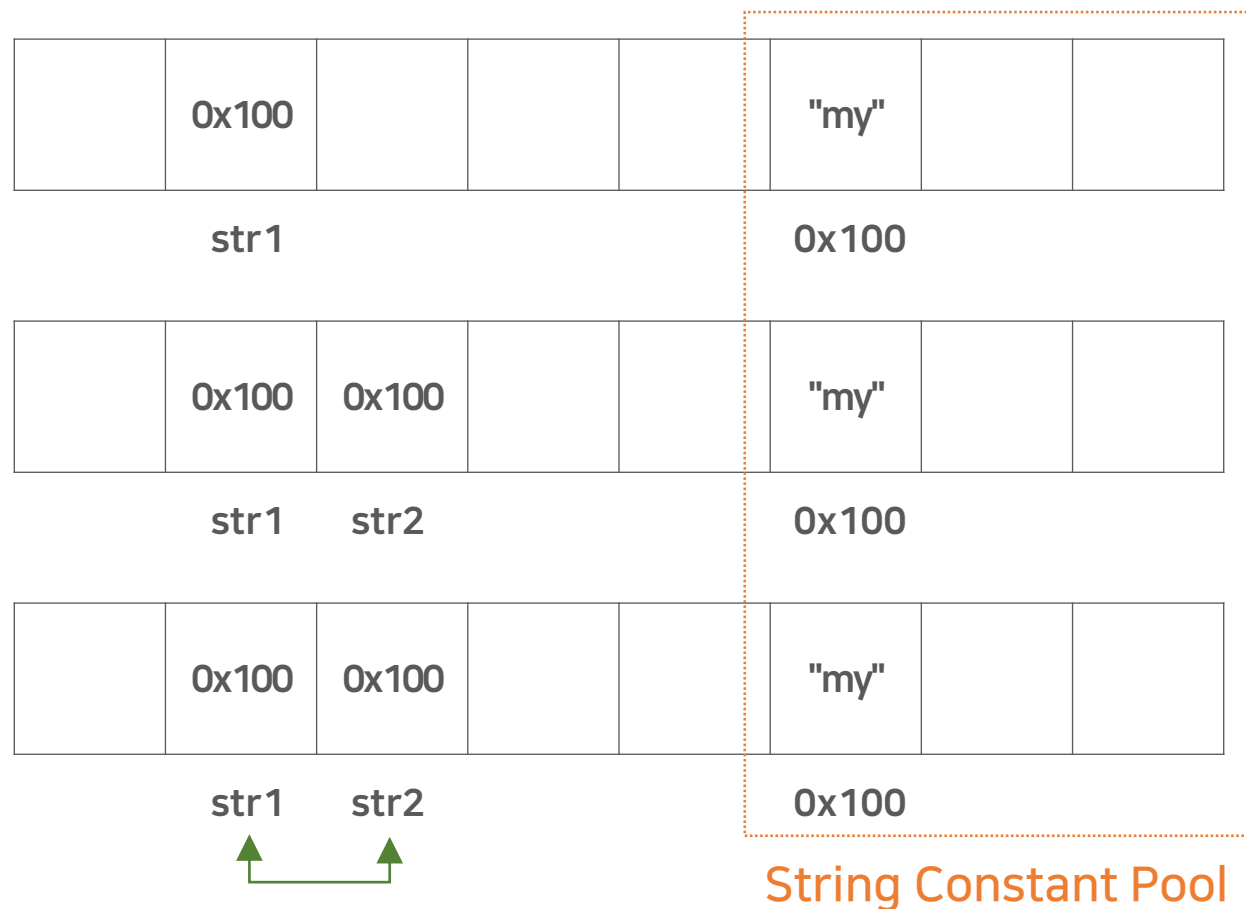
String 클래스 - 문자열 리터럴 메모리 구조

■ 문자열 리터럴 동등 비교하기

```
String str1 = "my";
```

```
String str2 = "my";
```

```
System.out.println(str1 == str2);
```



두 값이 동일하므로 true 반환

String 클래스 - 문자열 객체

- 문자열 객체
 - new String()을 이용해 생성한 문자열 정보
- 문자열 객체는 일반 객체처럼 힙 영역(Heap)에 저장됨
- new 명령은 항상 새로운 객체를 생성하므로 new String()을 이용하면 동일한 문자열 객체도 여러 개가 생성될 수 있음

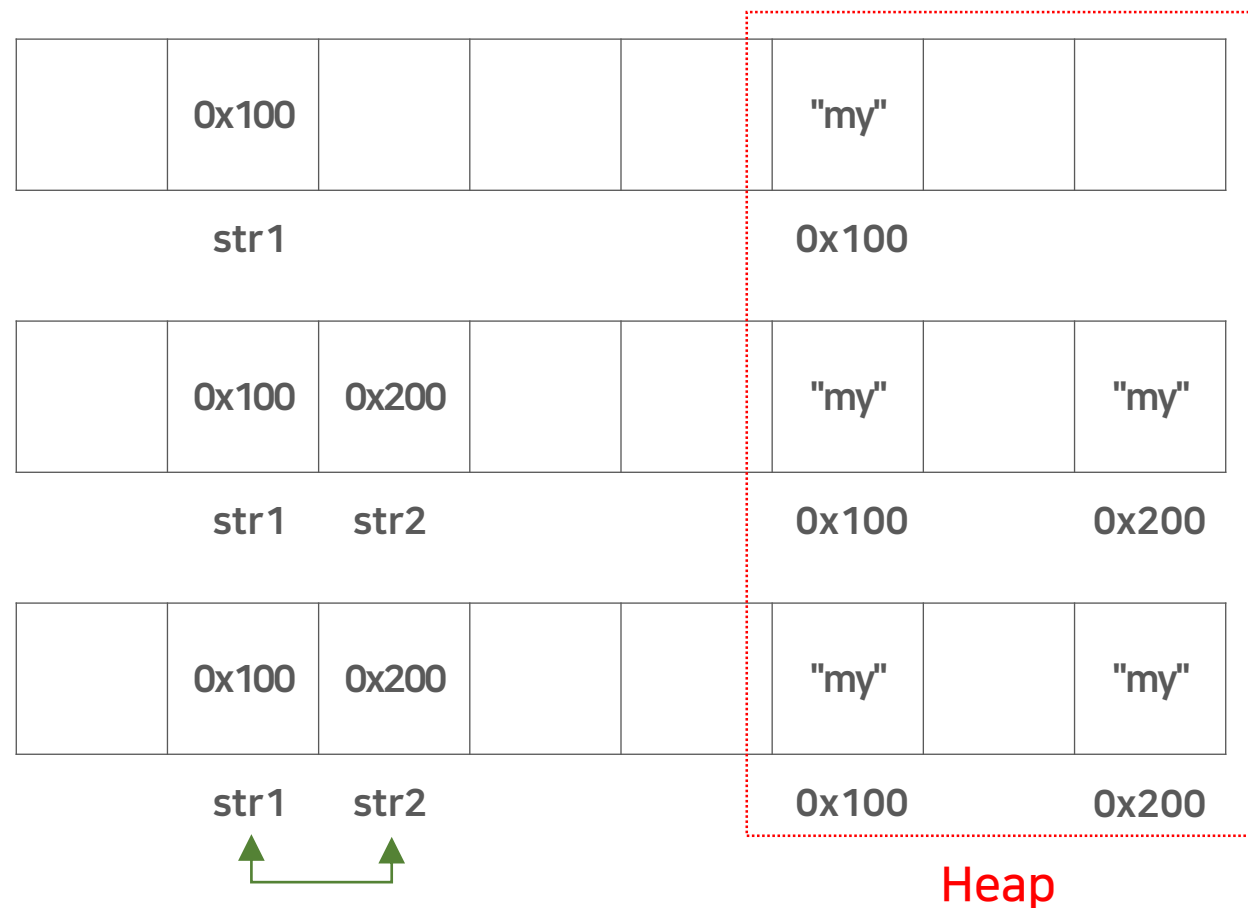
String 클래스 - 문자열 객체 메모리 구조

■ 문자열 객체 동등 비교하기

```
String str1 = new String("my");
```

```
String str2 = new String("my");
```

```
System.out.println(str1 == str2);
```



두 값이 다르므로 false 반환

String 클래스 - 메소드

■ String 클래스 주요 메소드-1

메소드	역할
char charAt(int index)	전달된 인덱스에 해당하는 문자 반환
int compareTo(String anotherString)	전달된 문자열 anotherString과 사전 편찬순으로 비교한 결과 반환 - 0 : 문자열이 동일한 경우 - 양수 : 현재 문자열이 anotherString보다 큰 경우 - 음수 : 현재 문자열이 anotherString보다 작은 경우
boolean contains(CharSequence s)	전달된 CharSequence s*가 포함되어 있으면 true 아니면 false 반환 * CharSequence 인터페이스: String, StringBuilder, StringBuffer가 속함
boolean endsWith(String suffix)	전달된 문자열 suffix로 끝나면 true 아니면 false 반환
boolean equals(Object anObject)	전달된 문자열 anObject와 같으면 true 아니면 false 반환
static String format(String format, Object... args)	전달된 데이터들을 주어진 format 형식의 문자열로 반환
byte[] getBytes()	현재 문자열을 바이트 배열로 변환하여 반환
int indexOf(String str)	전달된 문자열을 전체 검색. 검색 결과 중 첫 번째 결과의 인덱스 반환. 없으면 -1 반환
int indexOf(String str, int fromIndex)	전달된 문자열을 인덱스 fromIndex 이후에서 검색. 검색 결과 중 첫 번째 결과의 인덱스 반환. 없으면 -1 반환

String 클래스 - 메소드

■ String 클래스 주요 메소드-2

메소드	역할
<code>boolean isBlank()</code>	현재 빈 문자열이면 true 아니면 false 반환. white space는 빈 문자열로 취급
<code>boolean isEmpty()</code>	현재 문자열의 글자수가 0이면 true 아니면 false 반환
<code>static String join(CharSequence delimiter, CharSequence... elements)</code>	전달된 모든 elements들을 delimiter로 연결한 문자열을 반환
<code>int lastIndexOf(String str)</code>	전달된 문자열을 전체 검색. 검색 결과 중 마지막 결과의 인덱스 반환. 없으면 -1 반환
<code>int lastIndexOf(String str, int fromIndex)</code>	전달된 문자열을 인덱스 fromIndex 이전에서 검색. 검색 결과 중 마지막 결과의 인덱스 반환. 없으면 -1 반환
<code>int length()</code>	현재 문자열의 글자수 반환
<code>String repeat(int count)</code>	현재 문자열을 count만큼 반복해서 연결한 문자열 반환
<code>String replace(CharSequence target, CharSequence replacement)</code>	모든 target을 replacement로 변환한 문자열을 반환
<code>String replaceAll(String regex, String replacement)</code>	전달된 정규식 regex를 만족하는 모든 패턴을 찾아 replacement로 변환한 문자열을 반환

String 클래스 - 메소드

■ String 클래스 주요 메소드-3

메소드	역할
String[] split(String regex)	전달된 정규식 regex을 만족하는 모든 패턴으로 문자열을 분리한 결과들을 문자열 배열로 만들어서 반환
boolean startsWith(String prefix)	전달된 문자열 prefix로 시작하면 true 아니면 false 반환
String strip()	현재 문자열의 시작과 끝에 있는 공백 문자*를 제거한 결과 반환 * 공백 문자 : 유니코드에 포함된 모든 공백 문자
String substring(int beginIndex)	인덱스 beginIndex부터 끝까지 문자열을 반환
String substring(int beginIndex, int endIndex)	인덱스 beginIndex부터 인덱스 endIndex 이전까지 문자열을 반환
char[] toCharArray()	현재 문자열의 모든 문자를 char[] 배열의 요소로 만들어 반환
String toLowerCase()	현재 문자열을 소문자로 변환한 결과 반환
String toUpperCase()	현재 문자열을 대문자로 변환한 결과 반환
String trim()	현재 문자열의 시작과 끝에 있는 공백 문자**를 제거한 결과 반환 ** 공백 문자 : Space(U+0020, 코드값 32) 이하의 코드 값을 가진 공백 문자

String 클래스 - 메소드 예시1

■ char charAt(int index)

```
String str = "hello";  
System.out.println(str.charAt(0));  
System.out.println(str.charAt(1));  
System.out.println(str.charAt(2));
```

실행결과

'h'
'e'
'l'

■ char[] toCharArray()

```
String str = "USA";  
char[] array = str.toCharArray();  
for(char ch : array)  
    System.out.println(ch);
```

실행결과

'U'
'S'
'A'

String 클래스 - 메소드 예시2

- boolean startsWith(String prefix)

```
String str = "https://www.google.com/";  
System.out.println(str.startsWith("https"));  
System.out.println(str.startsWith("ftp"));
```

실행결과

true
false

- boolean endsWith(String suffix)

```
String str = "flower.jpg";  
System.out.println(str.endsWith("jpg"));  
System.out.println(str.endsWith("png"));
```

실행결과

true
false

String 클래스 - 메소드 예시3

- boolean contains(CharSequence s)

```
String str = "java python c#";  
System.out.println(str.contains("java"));  
System.out.println(str.contains("c++"));
```

실행결과

true
false

- int compareTo(String anotherString)

```
String str = "c";  
System.out.println(str.compareTo("a"));  
System.out.println(str.compareTo("c"));  
System.out.println(str.compareTo("e"));
```

실행결과

2 ("c" - "a")
0 ("c" - "c")
-2 ("c" - "e")

String 클래스 - 메소드 예시4

■ boolean equals(Object anObject)

```
String str = "java";  
System.out.println(str.equals("java"));  
System.out.println(str.equals("c++"));
```

실행결과

true
false

■ byte[] getBytes()

```
String str = "abc";  
byte[] bytes = str.getBytes();  
for(byte b : bytes)  
    System.out.println(b);
```

실행결과

97 ('a' 코드값)
98 ('b' 코드값)
99 ('c' 코드값)

String 클래스 - 메소드 예시5

■ static String format(String format, Object... args)

format	의미	format	의미	format	의미
%d	10진수(0~9)	%f	실수	숫자	출력할 자리수 지정
%o	8진수(0~7)	%c	유니코드 문자	마이너스(-) 기호	오른쪽 정렬
%x	16진수(0~f)	%t	날짜시간	마침표(.) 기호	소수점을 의미
%X	16진수(0~F)	%s	문자열	coma(,) 기호	천 단위 구분기호를 의미

```
int n = 12345;  
System.out.println("(" + String.format("%d", n) + ")");  
System.out.println("(" + String.format("%9d", n) + ")");  
System.out.println("(" + String.format("%-9d", n) + ")");  
System.out.println("(" + String.format("%09d", n) + ")");  
System.out.println("(" + String.format("%9.2f", (double)n) + ")");
```

실행결과

```
"(12,345)"  
"(      12345)"  
"(12345      )"  
"(000012345)"  
"(  12345.00)"
```

String 클래스 - 메소드 예시6

■ int indexOf(String str)

```
String str = "hello hello";  
System.out.println(str.indexOf("hello"));  
System.out.println(str.indexOf("bonjour"));
```

실행결과

0
-1

■ int indexOf(String str, int fromIndex)

```
String str = "hello hello";  
System.out.println(str.indexOf("hello", 1));  
System.out.println(str.indexOf("bonjour", 1));
```

실행결과

6
-1

String 클래스 - 메소드 예시7

■ int lastIndexOf(String str)

```
String str = "hehe";  
System.out.println(str.lastIndexOf("he"));  
System.out.println(str.lastIndexOf("she"));
```

실행결과

2
-1

■ int lastIndexOf(String str, int fromIndex)

```
String str = "hehe";  
System.out.println(str.lastIndexOf("he", 0));  
System.out.println(str.lastIndexOf("he", 1));  
System.out.println(str.lastIndexOf("he", 2));  
System.out.println(str.lastIndexOf("he", 3));
```

실행결과

0
0
2
2

String 클래스 - 메소드 예시8

■ boolean isBlank()

```
String str1 = "";  
String str2 = "  ";  
System.out.println(str1.isBlank());  
System.out.println(str2.isBlank());
```

실행결과

true
true

■ boolean isEmpty()

```
String str1 = "";  
String str2 = "  ";  
System.out.println(str1.isEmpty());  
System.out.println(str2.isEmpty());
```

실행결과

true
false

String 클래스 - 메소드 예시9

- static String join(CharSequence delimiter, CharSequence... elements)

```
String[] seasons = {"봄", "여름", "가을", "겨울"};  
String str = String.join("→", seasons);  
System.out.println(str);
```

실행결과

"봄→여름→가을→겨울"

- String[] split(String regex)

```
String str = "봄→여름→가을→겨울";  
String[] seasons = str.split("→");  
for(String season : seasons)  
    System.out.println(season);
```

실행결과

"봄"
"여름"
"가을"
"겨울"

String 클래스 - 메소드 예시 10

■ int length()

```
String str = "hello world";  
System.out.println(str.length());
```

실행결과

11

■ String repeat(int count)

```
String str = "만세";  
System.out.println(str.repeat(3));
```

실행결과

"만세만세만세"

String 클래스 - 메소드 예시 11

- String replace(CharSequence target, CharSequence replacement)

```
String str = "192.168.0.5";  
System.out.println(str.replace(".", "_"));
```

실행결과

"192_168_0_5"

- String replaceAll(String regex*, String replacement)

```
String str = "192.168.0.5";  
System.out.println(str.replaceAll(".", "_"));  
System.out.println(str.replaceAll("[.]", "_"));
```

실행결과

"_____"
"192_168_0_5"

* regex : 정규 표현식. 정규 표현식에서 마침표(.)는 모든 문자를 의미하므로 문자 클래스 []로 묶어서 표현한다.

String 클래스 - 메소드 예시12

■ String strip()

```
String str = " hello ";  
System.out.println("(" + str.strip() + ")");  
System.out.println(str.strip().length());
```

실행결과

“(hello)”
5

■ String trim()

```
String str = " hello ";  
System.out.println("(" + str.trim() + ")");  
System.out.println(str.trim().length());
```

실행결과

“(hello)”
5

String 클래스 - 메소드 예시 13

- String substring(int beginIndex)

```
String str = "hello world";  
System.out.println(str.substring(6));
```

실행결과

"world"

- String substring(int beginIndex, int endIndex)

```
String str = "hello world";  
System.out.println(str.substring(0, 5));
```

실행결과

"hello"

String 클래스 - 메소드 예시 14

■ String toLowerCase()

```
String str = "HELLO WORLD";  
System.out.println(str.toLowerCase());
```

실행결과

"hello world"

■ String toUpperCase()

```
String str = "hello world";  
System.out.println(str.toUpperCase());
```

실행결과

"HELLO WORLD"

String 클래스 - 단점

■ String 클래스의 단점

- 불변 객체(Immutable Object)이므로 + 연산을 수행할때마다 새로운 String 이 계속 생성됨

```
String str1 = "a";
```

```
String str2 = "b";
```

```
String str3 = "c";
```



```
str1 + str2 + str3
```

결과1

결과2



결과2

결과1

최종결과

메모리 낭비로 이어진다.

StringBuilder 클래스

- String 클래스의 + 연산자 사용으로 발생하는 메모리 낭비를 막기 위한 클래스
- 변경 가능한 문자 시퀀스 형식의 클래스
- StringBuffer 클래스와 동일한 목적을 가짐
 - StringBuffer 클래스 목적 ▶ 멀티 스레드* 환경에서 동작
 - StringBuilder 클래스 목적 ▶ 단일 스레드 환경에서 동작
 - 일반적인 단일 스레드 환경이라면 StringBuilder의 사용이 권장됨

* 멀티 스레드는 여러 작업이나 프로그램이 동시에 실행되는 것을 의미한다.

StringBuilder 클래스 - 생성자

■ StringBuilder 클래스 생성자

생성자	의미
StringBuilder()	비어 있는 StringBuilder 객체 생성. 초기 문자열 길이는 기본 16자로 설정
StringBuilder(int capacity)	비어 있는 StringBuilder 객체 생성. 초기 문자열 길이는 capacity로 설정
StringBuilder(CharSequence seq)	전달된 CharSequence seq와 동일한 값을 가지는 StringBuilder 객체 생성
StringBuilder(String str)	전달된 String str과 동일한 값을 가지는 StringBuilder 객체 생성

StringBuilder 클래스 - 메소드

■ StringBuilder 클래스 주요 메소드

메소드	역할
StringBuilder append(Type arg)	전달된 arg의 문자열 표현을 문자열 시퀀스 마지막에 추가함 - 사용 가능한 Type : boolean, char, char[], double, float, int, long, CharSequence, Object, String, StringBuffer
StringBuilder append(char[] str, int offset, int len)	전달된 char[] str 배열의 인덱스 offset부터 len개 요소를 문자열 시퀀스에 추가함
StringBuilder append(CharSequence s, int offset, int len)	전달된 CharSequence s의 인덱스 offset부터 len개 문자를 문자열 시퀀스에 추가함
StringBuilder delete(int start, int end)	문자열 시퀀스의 인덱스 start부터 end 이전까지 삭제함
StringBuilder deleteCharAt(int index)	문자열 시퀀스의 인덱스 index 요소를 삭제함
StringBuilder insert(int offset, Type arg)	전달된 arg의 문자열 표현을 문자열 시퀀스의 인덱스 offset에 추가함 - 사용 가능한 Type : boolean, char, char[], double, float, int, long, CharSequence, Object, String
int length()	문자열 시퀀스에 포함된 글자수 반환
String toString()	문자열 시퀀스를 String으로 변환한 뒤 반환

StringBuilder 클래스 - append() 메소드

■ append() 메소드

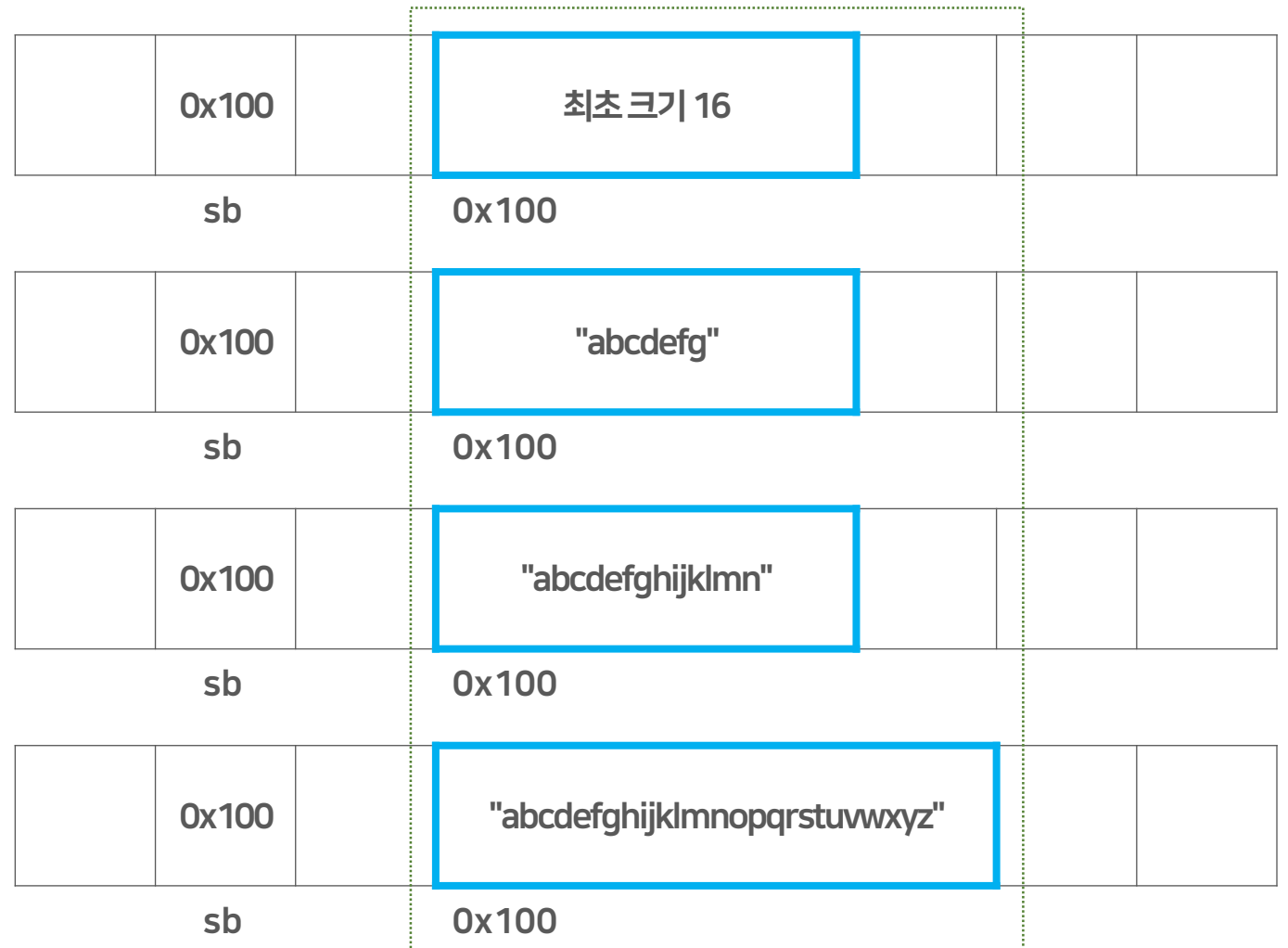
```
StringBuilder sb;  
sb = new StringBuilder();
```

```
sb.append("abcdefg");
```

```
sb.append("hijklmn");
```

```
sb.append("opqrstuvwxyz");
```

메모리 공간이 부족하면 자동으로 확장된다.



StringBuilder 클래스 - 주의사항

- StringBuilder 클래스로 생성한 문자열은 String 클래스 타입으로 변환 후 사용하는 것이 일반적임
- String 클래스 타입으로 변환하기

```
StringBuilder sb = new StringBuilder();  
// ... sb에 문자열 추가 코드 ... //  
String str = sb.toString();
```

toString() 메소드

```
StringBuilder sb = new StringBuilder();  
// ... sb에 문자열 추가 코드 ... //  
String str = new String(sb);
```

문자열 객체 생성

→ 어느 방법을 사용하던 상관 없다. ←

Math 클래스

- 수학에 관련된 기능을 제공하는 클래스
- 모든 멤버가 static 처리되어 있어 "Math.멤버" 형식으로 호출함
- 일반 연산자로 처리할 수 없는 다양한 수학 메소드를 지원함

Math 클래스 - 메소드

■ Math 클래스 주요 메소드

메소드	역할
static Type abs(Type a)	a의 절대값을 반환
static double ceil(double a)	a보다 크거나 같은 정수 중 가장 작은 값을 반환 (올림값)
static double floor(double a)	a보다 작거나 같은 정수 중 가장 큰 값을 반환 (내림값)
static Type max(Type a, Type b)	a와 b중에서 큰 값을 반환
static Type min(Type a, Type b)	a와 b중에서 작은 값을 반환
static double pow(double a, double b)	a의 b 제곱값을 반환
static double random()	0.0 이상 1.0 미만의 난수값을 반환
static double sqrt(double a)	a의 제곱근값을 반환 (루트값)
* abs, max, min 메소드에서 사용 가능한 Type : double, float, int, long	

Math 클래스 - random() 메소드

- random() 메소드를 이용해 원하는 int형 난수 생성하기

코드

`Math.random()`

`Math.random() * 5`

`Math.random() * 5 + 1`

`(int)(Math.random() * 5 + 1)`

생성범위

$0.0 \leq \text{난수} < 1.0$

$0.0 \leq \text{난수} < 5.0$

$1.0 \leq \text{난수} < 6.0$

$1 \leq \text{난수} < 6$ (1, 2, 3, 4, 5 중 하나)

일반화 코드

`(int)(Math.random() * 개수 + 시작값)`

```

each: function(e, t, n) {
    var r, i = 0,
        o = e.length,
        a = M(e);
    if (n) {
        if (a) {
            for (; o > i; i++)
                if (r = t.apply(e[i], n), r === !1) break;
        } else
            for (i in e)
                if (r = t.apply(e[i], n), r === !1) break;
    } else if (a) {
        for (; o > i; i++)
            if (r = t.call(e[i], e[i]), r === !1) break;
    } else
        for (i in e)
            if (r = t.call(e[i], e[i]), r === !1) break;
    return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
    return null == e ? "" : b.call(e);
} : function(e) {
    return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
    var n = t || [];
    return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
    var r;
    if (t) {
        if (n) return m.call(t, e, n);
        for (r = t.length, r = r ? 0 > n ? Math.max(0, r + n) : n : 0; r in t && t[r] === e) return r;
    }
}

```

2. java.util 패키지

java.util 패키지

- Java 개발에 유용한 클래스들이 모인 패키지
- 날짜/시간, 국제화 지원, 난수, 문자열 구문 분석, base64 인코딩 및 디코딩, 각종 유틸리티 클래스가 모여 있음
- 컬렉션 프레임워크 관련 클래스와 인터페이스가 모여 있음

Calendar 클래스

- 날짜/시간에 관한 정보를 제공하는 추상 클래스
- 나라마다 달력 체계가 다를 수 있기 때문에 추상 클래스로 존재함
- 추상 클래스이기 때문에 new를 이용한 객체 생성은 불가능함
 - 별도의 객체 생성 방법을 지원함
 - GregorianCalendar 클래스를 이용한 객체 생성을 기본으로 지원함
- Calendar 클래스의 모든 필드는 static이므로 "Calendar.필드" 형식으로 호출함

Calendar 클래스 - 객체 반환 메소드

■ Calendar 객체 반환 메소드

객체 반환 메소드	역할
static Calendar getInstance()	기본 TimeZone과 Locale을 이용해 현재 날짜와 시간을 반환
static Calendar getInstance(TimeZone zone)	전달된 TimeZone zone을 이용해 현재 날짜와 시간을 반환
static Calendar getInstance(Locale locale)	전달된 Locale locale을 이용해 현재 날짜와 시간을 반환
static Calendar getInstance(TimeZone zone, Locale locale)	전달된 TimeZone zone과 Locale locale을 이용해 현재 날짜와 시간을 반환

Calendar 클래스 - 객체 생성

■ Calendar 객체 생성하기

```
Calendar c1 = Calendar.getInstance();  
System.out.println(c1);
```

```
Calendar c2 = Calendar.getInstance(TimeZone.getTimeZone("Asia/Seoul"));  
System.out.println(c2);
```

```
Calendar c3 = Calendar.getInstance(Locale.KOREA);  
System.out.println(c3);
```

실행결과 (전체 결과 동일함)

```
java.util.GregorianCalendar[time=1701325022438,  
areFieldsSet=true,areAllFieldsSet=true,lenient=true  
,zone=sun.util.calendar.ZoneInfo[id="Asia/Seoul",of  
fset=32400000,dstSavings=0,useDaylight=false,tra  
nsitions=30,lastRule=null],firstDayOfWeek=1,minim  
alDaysInFirstWeek=1,ERA=1,YEAR=2023,MONTH=1  
0,WEEK_OF_YEAR=48,WEEK_OF_MONTH=5,DAY_OF  
_MONTH=30,DAY_OF_YEAR=334,DAY_OF_WEEK=5,  
DAY_OF_WEEK_IN_MONTH=5,AM_PM=1,HOUR=3,H  
OUR_OF_DAY=15,MINUTE=17,SECOND=2,MILLISEC  
OND=438,ZONE_OFFSET=32400000,DST_OFFSET=0]
```

Calendar 클래스 - 필드

■ Calendar 클래스 주요 필드

필드	역할
public static final int YEAR = 1	현재 년도 (4자리)
public static final int MONTH = 2	현재 월 (주의 0 - 11 사이 값을 반환함)
public static final int DAY_OF_MONTH = 5	현재 일 (1 - 31)
public static final int DAY_OF_WEEK = 7	현재 요일 (일:1, 월:2, 화:3, 수:4, 목:5, 금:6, 토:7)
public static final int AM_PM = 9	현재 오전/오후 여부 (오전:0, 오후:1)
public static final int HOUR = 10	현재 시간 12-hour clock (0 - 11)
public static final int HOUR_OF_DAY = 11	현재 시간 24-hour clock (0 - 23)
public static final int MINUTE = 12	현재 분 (0 - 59)
public static final int SECOND = 13	현재 초 (0 - 59)
public static final int MILLISECOND = 14	현재 밀리초 (0 - 999)

Calendar 클래스 - 메소드

■ Calendar 클래스 주요 메소드

메소드	역할
<code>void add(int field, int amount)</code>	Calendar 객체의 field를 amount만큼 증가시킴
<code>int get(int field)</code>	Calendar 객체의 field에 해당하는 값을 반환
<code>final Date getTime()</code>	Calendar 객체를 java.util.Date 객체로 변환해서 반환
<code>long getTimeInMillis()</code>	Calendar 객체를 타임스탬프 값으로 변환해서 반환
<code>final void set(int year, int month, int date, int hourOfDay, int minute, int second)</code>	Calendar 객체의 년,월,일,시,분,초 값을 변경함
<code>final void setTime(Date date)</code>	전달된 java.util.Date 객체 값으로 Calendar 객체 값을 변경함

Calendar 클래스 - 필드

- Calendar 필드와 get() 메소드를 이용해 원하는 단위 알아내기

```
Calendar calendar = Calendar.getInstance();  
int year = calendar.get(Calendar.YEAR);  
int month = calendar.get(Calendar.MONTH);  
int day = calendar.get(Calendar.DAY_OF_MONTH);  
int weekNo = calendar.get(Calendar.DAY_OF_WEEK);  
int ampm = calendar.get(Calendar.AM_PM);  
int hour12 = calendar.get(Calendar.HOUR);  
int hour24 = calendar.get(Calendar.HOUR_OF_DAY);  
int minute = calendar.get(Calendar.MINUTE);  
int second = calendar.get(Calendar.SECOND);  
int millisecond = calendar.get(Calendar.MILLISECOND);
```

변수값 (2023-10-20 금요일 14:50:30.250 기준)

2023 (년)
9 (월. 10월을 의미)
20 (일)
6 (요일. 금요일 의미)
1 (오전/오후. 오후를 의미)
2 (시간. 12시각)
14 (시간. 24시각)
50 (분)
30 (초)
250 (밀리초)

java.util.Date 클래스

- 특정 날짜와 시간을 밀리초 단위까지 나타내는 클래스
- 많은 생성자와 메소드가 Deprecated 상태임(더 이상 사용되지 않음)
- 현재 날짜와 시간을 쉽게 알 수 있다는 장점이 있음
- 지원중인 생성자

생성자	역할
Date()	현재 날짜와 시간 객체 생성
Date(long date)	전달된 타임스탬프(long date)값에 해당하는 날짜와 시간 객체 생성

java.sql.Date 클래스

- java.util.Date 클래스의 자식 클래스
- JDBC*에서 테이블의 DATE 타입 칼럼값을 처리하기 위한 Wrapper 역할
- 데이터베이스의 날짜 타입 데이터를 처리하기 위해서 존재함

* JDBC : Java DataBase Connection. 자바로 데이터베이스에 접속할 때 사용하는 자바 API

Optional<T> 클래스

- 어떤 값을 wrapping할 때 사용하는 컨테이너 클래스
- wrapping 값의 타입을 제네릭 타입으로 처리함
- wrappin한 값의 null 여부에 따라서 꺼내서 사용할 값을 지정할 수 있음
 - 감싼 값이 non-null이다 ▶ 그대로 꺼내서 사용한다
 - 감싼 값이 null이다 ▶ null 대신 꺼낸 값을 사용한다

Optional<T> 클래스 - 메소드

■ Optional<T> 클래스 주요 메소드

메소드	역할
<code>boolean isEmpty()</code>	Optional 객체에 저장한 값이 없으면 true 아니면 false 반환
<code>boolean isPresent()</code>	Optional 객체에 저장한 값이 있으면 true 아니면 false 반환
<code>static<T> Optional<T> of(T value)</code>	T 타입의 value 값을 wrapping한 Optional 객체 반환 T 타입의 value는 non-null이어야 함
<code>static<T> Optional<T> ofNullable(T value)</code>	T 타입의 value 값을 wrapping한 Optional 객체 반환 T 타입의 value가 non-null이면 null 값을 wrapping한 Optional 객체 반환
<code>T get()</code>	Optional 객체에 저장한 값을 반환 저장한 값이 null이면 NoSuchElementException 발생
<code>T orElse(T other)</code>	Optional 객체에 저장한 값을 반환 저장한 값이 null이면 T 타입의 other 값을 반환

Optional<T> 클래스 - 메소드 예시1

- non-null인 값을 wrapping한 뒤 사용하기

```
String str = "java";  
Optional<String> opt1 = Optional.of(str);  
System.out.println(opt1.get());  
  
int num = 50;  
Optional<Integer> opt2 = Optional.of(num);  
System.out.println(opt2.get());
```

실행결과

"java"
50

Optional<T> 클래스 - 메소드 예시2

- null인 값을 wrapping한 뒤 사용하기

```
String str1 = "java";  
Optional<String> opt1 = Optional.ofNullable(str1);  
System.out.println(opt1.orElse("python"));  
  
String str2 = null;  
Optional<String> opt2 = Optional.ofNullable(str2);  
System.out.println(opt2.orElse("python"));
```

실행결과

"java"
"python"

▶ opt2에 저장한 값이 null이면 "python"을 꺼내시오.

Scanner 클래스

- 어떤 값을 입력 받을 수 있도록 구성된 간단한 텍스트 입력기
- 기본적으로 공백을 기준으로 입력 값을 구분하도록 설계되어 있음
- 입력 받을 값의 타입에 따라 사용할 메소드를 선택할 수 있음
- 주로 Console View를 이용해서 어떤 값을 입력 받기 위해서 사용함

Scanner 클래스 - 객체 생성

- 표준 입력 스트림 필드인 System.in 필드를 사용해서 Scanner 객체를 생성하면 키보드 입력이 가능함

```
Scanner sc = new Scanner(System.in);
```



Scanner 클래스 - 메소드

■ Scanner 클래스 주요 메소드

메소드	역할
String next()	스캐너가 입력 받은 문자열 토큰*을 반환함
String nextLine()	스캐너가 입력 받은 문자열 토큰을 반환함 (줄 바꿈으로 문자열 토큰을 구분함)
boolean nextBoolean()	스캐너가 입력 받은 boolean 토큰을 반환함
byte nextByte()	스캐너가 입력 받은 byte 토큰을 반환함
double nextDouble()	스캐너가 입력 받은 double 토큰을 반환함
int nextInt()	스캐너가 입력 받은 int 토큰을 반환함
long nextLong()	스캐너가 입력 받은 long 토큰을 반환함
void close()	스캐너가 사용한 자원을 반납하고 닫음

* 기본적으로 모든 토큰은 공백으로 입력을 구분한다.

Scanner 클래스 - 메소드

■ 데이터 타입별 입력 예시

```
Scanner sc = new Scanner(System.in);
```

```
boolean isAlive = sc.nextBoolean();
```

```
int age = sc.nextInt();
```

```
long balance = sc.nextLong();
```

```
double height = sc.nextDouble();
```

```
String name = sc.next();
```

```
String address = sc.nextLine();
```

```
char gender = sc.next().charAt(0);
```

```
sc.close();
```

boolean 타입 입력은 nextBoolean() 메소드 사용

int 타입 입력은 nextInt() 메소드 사용

long 타입 입력은 nextLong() 메소드 사용

double 타입 입력은 nextDouble() 메소드 사용

입력 문자열에 공백이 포함되지 않으면 next() 메소드 사용

입력 문자열에 공백이 포함되면 nextLine() 메소드 사용

char 타입은 String으로 받은 뒤 첫 글자만 사용하는 방식으로 처리

입력 받기 종료

UUID 개념

- UUID
 - Universally Unique Identifier
 - 범용 고유 식별자
- 모든 리소스 중 어떤 리소스를 고유하게 식별할 때 사용하는 레이블
- 이론상 생성한 UUID는 중복이 있을 수 있지만 중복 가능성이 거의 없으므로 중복이 없는 것으로 간주함
- 형식
 - 123e4567-e89b-12d3-a456-426614174000
 - 하이픈(-)으로 구분된 5개의 16진수 문자열 형식의 36자 문자열(128비트)

UUID 버전

■ 총 5개의 UUID 버전이 있음

버전	설명	특징
UUID Version 1	현재 시간과 랜덤 MAC 주소 이용	Unique 보장되지만 보안 취약
UUID Version 2	버전1과 유사. POSIX UID(사용자 ID)를 이용	거의 사용 안 함
UUID Version 3	MD5 해시 기반으로 이름과 네임스페이스 조합을 이용	해시 기반이므로 보안이 우수함 이름과 네임스페이스 조합이 같으면 동일한 UUID 생성
UUID Version 4	난수 이용	보안이 우수하고 빠름 가장 대중적인 방식
UUID Version 5	버전3와 유사. MD5 해시 대신 SHA-1 해시 사용	SHA-1 해시는 보안 취약점이 있음

UUID 클래스

- UUID를 생성하는 클래스
- UUID 생성을 위한 3개의 정적 메소드를 제공함

메소드	역할
static UUID fromString(String name)	주어진 String name 문자열을 이용해 UUID를 생성함
static UUID nameUUIDFromBytes(byte[] name)	주어진 byte[] name 배열을 이용해 UUID를 생성함 (UUID Version 3)
static UUID randomUUID()	난수를 이용해 UUID를 생성함 (UUID Version 4)

UUID 클래스 - UUID 값 생성

- UUID Version 3(이름 이용)를 이용해 UUID 값 생성

```
String name = "john";  
String uuid = UUID.nameUUIDFromBytes(str.getBytes()).toString();  
System.out.println(uuid);
```

실행결과

```
"527bd5b5-d689-  
32c3-aae9-  
74c6229ff785"
```

- UUID Version 4(난수 이용)를 이용해 UUID 값 생성

```
String uuid = UUID.randomUUID().toString();  
System.out.println(uuid);
```

실행결과

```
"123e4567-e89b-  
12d3-a456-  
426614174000"
```

```

each: function(e, t, n) {
    var r, i = 0,
        o = e.length,
        a = M(e);
    if (n) {
        if (a) {
            for (; o > i; i++)
                if (r = t.apply(e[i], n), r === !1) break;
        } else
            for (i in e)
                if (r = t.apply(e[i], n), r === !1) break;
    } else if (a) {
        for (; o > i; i++)
            if (r = t.call(e[i], , e[i]), r === !1) break;
    } else
        for (i in e)
            if (r = t.call(e[i], , e[i]), r === !1) break;
    return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
    return null == e ? "" : b.call(e);
} : function(e) {
    return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
    var n = t || [];
    return null != e && (M(Object(e)) ? x.merge(n, "string"
    );
inArray: function(e, t, n) {
    var r;
    if (t) {
        if (m) return m.call(t, e, n);
        for (r = t.length, r = r ? 0 > n ? Math.max(0, r + n) : n; r in t && t[r] === e) return r;
    }
}

```

3. java.text 패키지

java.text 패키지

- 문자열, 날짜/시간, 숫자 그리고 메시지 등을 처리하기 위한 클래스와 인터페이스가 모인 패키지
- 날짜/시간 및 숫자의 형식 지정이 가능한 클래스를 지원함
 - DateFormat 클래스
 - NumberFormat 클래스

SimpleDateFormat 클래스

- DateFormat 클래스의 자식 클래스
- java.util.Date 클래스에 원하는 패턴을 지정할 수 있음
- 서식 지정이 가능함
 - 날짜/시간 → 문자열
- 구문 분석이 가능함
 - 문자열 → 날짜/시간

SimpleDateFormat 클래스 - Pattern Letter

■ 날짜/시간 패턴 문자

Pattern Letter	의미	Pattern Letter	의미
yy	2자리 년도	a	AMPM(오전/오후)
y/yyy/yyyy	4자리 년도	K	시(0 ~ 11)
M L	월(1 ~ 12)	H	시(0 ~ 23)
w	주(1년 중 27주차)	h	시(1 ~ 12)
D	일(1년 중 189일째)	k	시(1 ~ 24)
d	일(1 ~ 31)	m	분(0 ~ 59)
u	요일(1 ~ 7), 1:월 ~ 7:일	s	초(0 ~ 59)
E	요일(월 ~ 일)	S	밀리초(0 ~ 999)

SimpleDateFormat 클래스 - 생성자

■ SimpleDateFormat 클래스 주요 생성자

생성자	의미
SimpleDateFormat()	기본 패턴을 사용하는 SimpleDateFormat 객체 생성
SimpleDateFormat(String pattern)	전달된 String pattern을 사용하는 SimpleDateFormat 객체 생성
SimpleDateFormat(String pattern, Locale locale)	전달된 String pattern과 Locale locale을 사용하는 SimpleDateFormat 객체 생성

SimpleDateFormat 클래스 - 서식 적용

- final String format(Date date)

```
import java.util.Date;

SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMdd");
Date date = new Date();
String str = sdf.format(date);
System.out.println(str);
```

실행결과 (2023-10-25 기준)

"20231025"

SimpleDateFormat 클래스 - 구문 분석

- Date parse(String source) throws ParseException

```
import java.util.Date;

String str = "20231025";
SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMdd");
Date date = null;
try {
    date = sdf.parse(str);
} catch (ParseException e) {
    System.out.println(e.getMessage());
}
System.out.println(date);
```

실행결과 (2023-10-25 기준)

Wed Oct 25 00:00:00
KST 2023

DecimalFormat 클래스

- NumberFormat 클래스의 자식 클래스
- 숫자에 원하는 패턴을 지정할 수 있음
- 서식 지정이 가능함
 - 숫자(Number) → 문자열
- 구문 분석이 가능함
 - 문자열 → 숫자(Number)

DecimalFormat 클래스 - Symbols

■ 패턴 문자

Symbol	의미
0	숫자 (불필요한 자리에도 0을 표시)
#	숫자 (불필요한 자리에는 표시 안함)
.	마침표 (정수부와 소수부를 구분)
,	콤마 (천 단위 구분 기호 표시)
%	백분율 (값에 100을 곱한 뒤 % 기호 표시)
-	마이너스 기호 표시

■ 패턴 예시

Symbol	의미
#,##0	천 단위 구분 기호를 표시한 정수
0.00	소수부 2자리를 표시한 정수
#,##0.00	천 단위 구분 기호와 소수부 2자리를 표시한 정수

DecimalFormat 클래스 - 생성자

■ DecimalFormat 클래스 주요 생성자

생성자	의미
DecimalFormat()	기본 패턴을 사용하는 DecimalFormat 객체 생성
DecimalFormat(String pattern)	전달된 String pattern을 사용하는 DecimalFormat 객체 생성

DecimalFormat 클래스 - 서식 적용

- final String format(double number)
- final String format(long number)

```
DecimalFormat df = new DecimalFormat("#,###0");  
long number = 10000L;  
String str = df.format(number);  
System.out.println(str);
```

실행결과

"10,000"

DecimalFormat 클래스 - 구문 분석

- Number parse(String source) throws ParseException

```
String str = "10,000";  
DecimalFormat df = new DecimalFormat("#,###0");  
Number number = null;  
try {  
    number = df.parse(str);  
} catch (ParseException e) {  
    System.out.println(e.getMessage());  
}  
System.out.println(number);
```

실행결과

10000

```

each: function(e, t, n) {
    var r, i = 0,
        o = e.length,
        a = M(e);
    if (n) {
        if (a) {
            for (; o > i; i++)
                if (r = t.apply(e[i], n), r === !1) break;
        } else
            for (i in e)
                if (r = t.apply(e[i], n), r === !1) break;
    } else if (a) {
        for (; o > i; i++)
            if (r = t.call(e[i], e[i]), r === !1) break;
    } else
        for (i in e)
            if (r = t.call(e[i], e[i]), r === !1) break;
    return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
    return null == e ? "" : b.call(e);
} : function(e) {
    return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
    var n = t || [];
    return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
    var r;
    if (t) {
        if (n) return m.call(t, e, n);
        for (r = t.length, r = r ? 0 > n ? Math.max(0, r + n) : n : 0; r in t && t[r] === e) return r;
    }
}

```

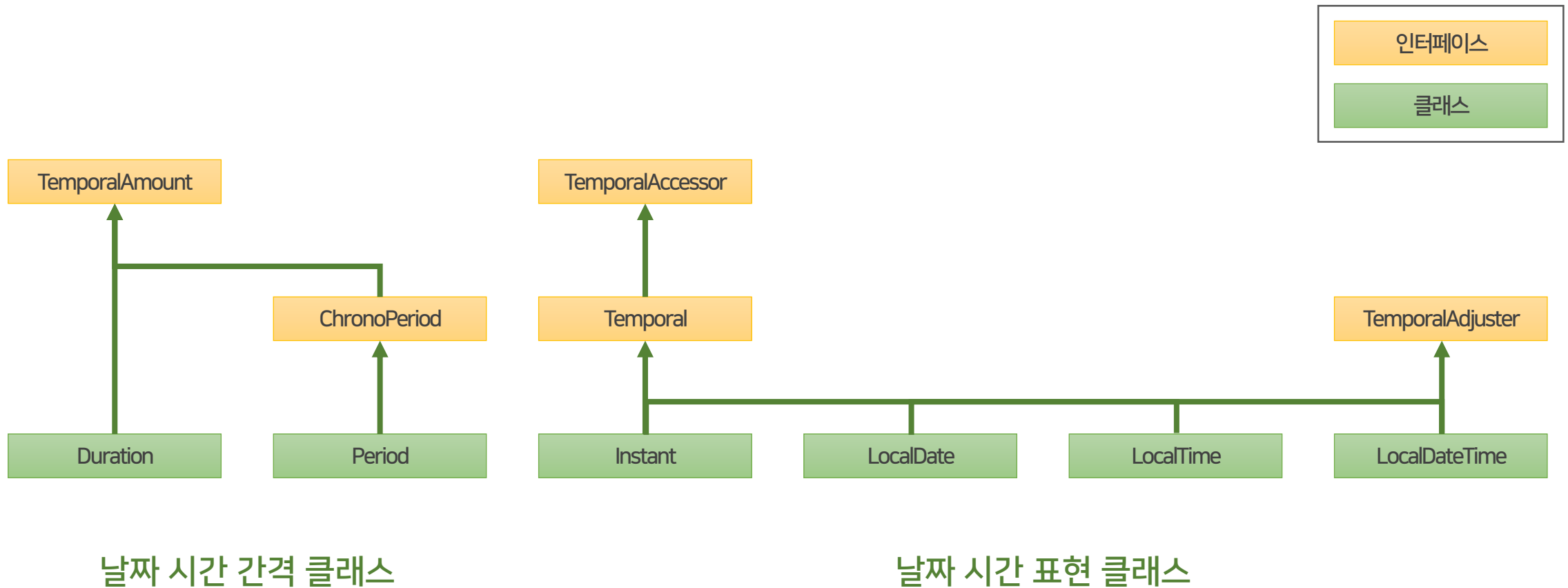
4. java.time 패키지

java.time 패키지

- 날짜, 시간, 순간, 기간에 대한 메인 API를 제공하는 패키지
- java.util 패키지에 있는 Date 클래스와 Calendar 클래스의 단점을 보완하기 위해 만든 패키지
- 불변 객체(Immutable Object) 활용
 - 날짜/시간을 변경하면 기존 객체는 그대로 두고 변경된 새로운 객체를 반환하는 방식 사용
 - 객체 값이 변하지 않으므로 여러 스레드가 객체를 공유하는 멀티 스레드 환경에서 안전하게 사용 가능함

클래스 구조도

- java.time 패키지의 주요 인터페이스와 클래스 구조도



날짜 시간 표현 클래스

- 특정 날짜와 시간을 나타내는 클래스

- 종류

- | | |
|---------|-----------------|
| 1. 날짜 | ▶ LocalDate |
| 2. 시간 | ▶ LocalTime |
| 3. 날짜시간 | ▶ LocalDateTime |

날짜 시간 표현 클래스 - 객체 반환 메소드

■ 객체 반환 메소드

반환타입	메소드	역할
static LocalDate	now()	시스템의 현재 날짜를 반환
	of (int year, int month, int dayOfMonth)	year년 month월 dayOfMonth일 날짜를 반환
	parse(CharSequence text)	문자열 형식의 text 날짜를 LocalDate 타입의 날짜로 변환 후 반환
static LocalTime	now()	시스템의 현재 시간을 반환
	of (int hour, int minute, int second)	hour시 minute분 second초 시간을 반환
	parse(CharSequence text)	문자열 형식의 text 시간을 LocalTime 타입의 시간으로 변환 후 반환
static LocalDateTime	now()	시스템의 현재 날짜와 시간을 반환
	of (int year, int month, int dayOfMonth, int hour, int minute, int second)	year년 month월 dayOfMonth일 hour시 minute분 second초 날짜와 시간을 반환
	parse(CharSequence text)	문자열 형식의 text 날짜와 시간을 LocalDateTime 타입의 날짜시간 으로 변환 후 반환

날짜 시간 표현 클래스 - 메소드

■ 주요 메소드-1

용도	클래스	메소드	역할
특정 필드 조회	LocalDate	int getYear() int getMonthValue() int getDayOfMonth() int getDayOfWeek().getValue()	년도 반환 월 반환(1-12) 일 반환(1-31) 요일 반환(월:1, 화:2, 수:3, 목:4, 금:5, 토:6, 일:7)
	LocalTime	int getHour() int getMinute() int getSecond()	시간 반환(0-23) 분 반환(0-59) 초 반환(0-59)
필드 변경	LocalDate	LocalDate withYear(int year) LocalDate withMonth(int month) LocalDate withDayOfMonth(int dayOfMonth)	년도 변경 월 변경 일 변경
	LocalTime	LocalTime withHour(int hour) LocalTime withMinute(int minute) LocalTime withSecond(int second)	시 변경 분 변경 초 변경

날짜 시간 표현 클래스 - 메소드

■ 주요 메소드-2

용도	클래스	메소드	역할
필드 증가	LocalDate	LocalDate plusYears(long yearsToAdd) LocalDate plusMonths(long monthsToAdd) LocalDate plusDays(long daysToAdd) LocalDate plusWeeks(long weeksToAdd)	년도 증가 월 증가 일 증가 주 증가
	LocalTime	LocalTime plusHours(long hoursToAdd) LocalTime plusMinutes(long minutesToAdd) LocalTime plusSeconds(long secondsToAdd)	시 증가 분 증가 초 증가
필드 감소	LocalDate	LocalDate minusYears(long yearsToSubtract) LocalDate minusMonths(long monthsToSubtract) LocalDate minusDays(long daysToSubtract) LocalDate minusWeeks(long weeksToSubtract)	년도 감소 월 감소 일 감소 주 감소
	LocalTime	LocalTime minusHours(long hoursToSubtract) LocalTime minusMinutes(long minutesToSubtract) LocalTime minusSeconds(long secondsToSubtract)	시 감소 분 감소 초 감소

LocalDate 클래스 - 날짜 생성

■ LocalDate 날짜 생성

```
LocalDate date1 = LocalDate.now();  
System.out.println(date1);  
LocalDate date2 = LocalDate.of(2023, 10, 25);  
System.out.println(date2);  
LocalDate date3 = LocalDate.parse("2023-10-25");  
System.out.println(date3);
```

실행결과 (2023-10-25 기준)

```
"2023-10-25"  
"2023-10-25"  
"2023-10-25"
```

LocalTime 클래스 - 시간 생성

■ LocalTime 시간 생성

```
LocalTime time1 = LocalTime.now();  
System.out.println(time1);  
LocalTime time2 = LocalTime.of(14, 30, 45);  
System.out.println(time2);  
LocalTime time3 = LocalTime.parse("14:30:45");  
System.out.println(time3);
```

실행결과 (14:30:45 기준)

```
"14:30:45.123456789"  
"14:30:45"  
"14:30:45"
```


LocalDateTime 클래스 - 날짜 시간 생성

■ LocalDateTime 날짜 시간 생성

```
LocalDateTime dateTime1 = LocalDateTime.now();  
System.out.println(dateTime1);  
LocalDateTime dateTime2 = LocalDateTime.of(2023, 10, 25, 14, 30, 45);  
System.out.println(dateTime2);  
LocalDateTime dateTime3 = LocalDateTime.parse("2023-10-25T14:30:45");  
System.out.println(dateTime3);
```

실행결과 (2023-10-25 14:30:45 기준)

```
"2023-10-25T14:30:45.123456789 "  
"2023-10-25T14:30:45 "  
"2023-10-25T14:30:45 "
```

DateTimeFormatter 클래스

- 날짜 시간 객체의 출력 형식을 지정하는 클래스
- LocalDate/LocalTime/LocalDateTime 클래스에 원하는 패턴을 적용할 수 있음
- 서식 지정이 가능함
 - 날짜/시간 → 문자열
- 구문 분석이 가능함
 - 문자열 → 날짜/시간

DateTimeFormatter 클래스 - Symbols

■ 날짜/시간 심볼 문자

Symbol	의미	Symbol	의미
yy uu	2자리 년도	a	AMPM(오전/오후)
y/yyy/yyyy u/uuu/uuuu	4자리 년도	K	시(0 ~ 11)
M L	월(1 ~ 12)	H	시(0 ~ 23)
w	주(1년 중 27주차)	h	시(1 ~ 12)
D	일(1년 중 189일째)	k	시(1 ~ 24)
d	일(1 ~ 31)	m	분(0 ~ 59)
e/c	요일(1 ~ 7), 1:일 ~ 7:토	s	초(0 ~ 59)
E	요일(월 ~ 일)	S	밀리초(0 ~ 999)

DateTimeFormatter 클래스 - 서식 적용

- static DateTimeFormatter ofPattern(String pattern)

```
LocalDateTime dateTime = LocalDateTime.now();  
DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy년 MM월 dd일 E요일 a hh시 mm분 ss초");  
String strDateTime = dtf.format(dateTime);  
System.out.println(strDateTime);
```

실행결과 (2023-10-25 14:30:45 기준)

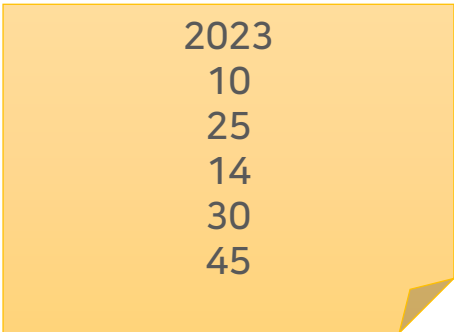
"2023년 10월 25일 수요일 오후 02시 30분 45초 "

DateTimeFormatter 클래스 - 구문 분석

- TemporalAccessor parse(CharSequence text)

```
String str = "2023-10-25 14:30:45";  
DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");  
TemporalAccessor temporalAccessor = dtf.parse(str);  
System.out.println(temporalAccessor.get(ChronoField.YEAR));  
System.out.println(temporalAccessor.get(ChronoField.MONTH_OF_YEAR));  
System.out.println(temporalAccessor.get(ChronoField.DAY_OF_MONTH));  
System.out.println(temporalAccessor.get(ChronoField.HOUR_OF_DAY));  
System.out.println(temporalAccessor.get(ChronoField.MINUTE_OF_HOUR));  
System.out.println(temporalAccessor.get(ChronoField.SECOND_OF_MINUTE));
```

실행결과



```
2023  
10  
25  
14  
30  
45
```