

Chapter 12 java.base 모듈

12.1 API 도큐먼트

12.2 java.base 모듈

12.3 Object 클래스

12.4 System 클래스

12.5 문자열 클래스

12.6 포장 클래스

12.7 수학 클래스

12.8 날짜와 시간 클래스

12.9 형식 클래스

12.10 정규 표현식 클래스

12.11 리플렉션

12.12 어노테이션

API 문서

- 자바 표준 모듈에서 제공하는 라이브러리를 쉽게 찾아서 사용할 수 있도록 도와주는 문서
- JDK 버전별로 사용할 수 있는 API 문서:
<https://docs.oracle.com/en/java/javase/index.html>

String 문서를 찾는 3가지 방법

방법1: 웹 사이트 메뉴 이용

- ① [All Modules] 탭에서 java.base 모듈을 클릭한다.
- ② java.base의 Packages 목록에서 java.lang 패키지를 클릭한다.
- ③ java.lang의 [All Classes and Interfaces] 탭에서 String 클래스를 클릭한다.



방법2: 웹 사이트 검색 이용

- ① 오른쪽 상단의 Search 검색란에 'String'을 입력한다.
- ② 드롭다운 목록에서 java.lang.String 항목을 선택한다.

방법3: 이클립스 이용

- ① 자바 코드에서 String 클래스를 마우스로 선택한 다음 (F1) 키를 누르면 Help 뷰가 나타난다.
- ② Help 뷰에서 Javadoc for 'java.lang.String' 링크를 클릭하면 String 문서로 이동한다.

java.base

- 모든 모듈이 의존하는 기본 모듈로, 모듈 중 유일하게 requires하지 않아도 사용할 수 있음

java.lang

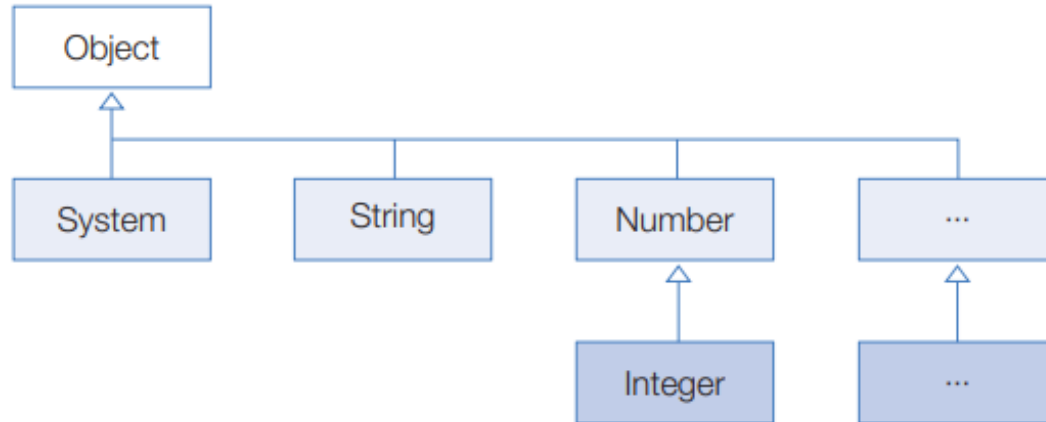
- 자바 언어의 기본적인 클래스를 담고 있는 패키지. 이 패키지에 있는 클래스와 인터페이스는 import 없이 사용할 수 있음

패키지	용도
java.lang	자바 언어의 기본 클래스를 제공
java.util	자료 구조와 관련된 컬렉션 클래스를 제공
java.text	날짜 및 숫자를 원하는 형태의 문자열로 만들어 주는 포맷 클래스를 제공
java.time	날짜 및 시간을 조작하거나 연산하는 클래스를 제공
java.io	입출력 스트림 클래스를 제공
java.net	네트워크 통신과 관련된 클래스를 제공
java.nio	데이터 저장을 위한 Buffer 및 새로운 입출력 클래스 제공

클래스		용도
Object		- 자바 클래스의 최상위 클래스로 사용
System		- 키보드로부터 데이터를 입력받을 때 사용 - 모니터(콘솔)로 출력하기 위해 사용 - 프로세스를 종료시킬 때 사용 - 진행 시간을 읽을 때 사용 - 시스템 속성(프로퍼티)을 읽을 때 사용
문자열 관련	String	- 문자열을 저장하고 조작할 때 사용
	StringBuilder	- 효율적인 문자열 조작 기능이 필요할 때 사용
	java.util.StringTokenizer	- 구분자로 연결된 문자열을 분리할 때 사용
포장 관련	Byte, Short, Character Integer, Float, Double Boolean	- 기본 타입의 값을 포장할 때 사용 - 문자열을 기본 타입으로 변환할 때 사용
Math		- 수학 계산이 필요할 때 사용
Class		- 클래스의 메타 정보(이름, 구성 멤버) 등을 조사할 때 사용

Object 클래스

- 클래스를 선언할 때 extends 키워드로 다른 클래스를 상속하지 않으면 암시적으로 java.lang.Object 클래스를 상속 → 자바의 모든 클래스는 Object의 자식이거나 자손 클래스



메소드	용도
boolean equals(Object obj)	객체의 번지를 비교하고 결과를 리턴
int hashCode()	객체의 해시코드를 리턴
String toString()	객체 문자 정보를 리턴

객체 동등 비교

- Object의 equals() 메소드는 객체의 번지를 비교하고 boolean 값을 리턴

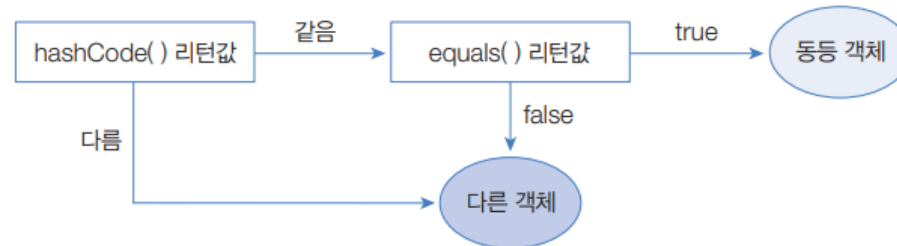
```
public boolean equals(Object obj)
```

객체 해시코드

- 객체를 식별하는 정수. Object의 hashCode() 메소드는 객체의 메모리 번지를 이용해서 해시코드를 생성하기 때문에 객체마다 다른 정수값을 리턴

```
public int hashCode()
```

- hashCode()가 리턴하는 정수값이 같은지 확인하고, equals() 메소드가 true를 리턴하는지 확인해서 동등 객체임을 판단



객체 문자 정보

- 객체를 문자열로 표현한 값. Object의 toString() 메소드는 객체의 문자 정보를 리턴
- 기본적으로 Object의 toString() 메소드는 '클래스명@16진수해시코드'로 구성된 문자열을 리턴

```
Object obj = new Object();  
System.out.println(obj.toString());
```



```
java.lang.Object@de6ced
```

레코드 선언

- 데이터 전달을 위한 DTO 작성 시 반복적으로 사용되는 코드를 줄이기 위해 도입

```
public record Person(String name, int age) {  
}
```

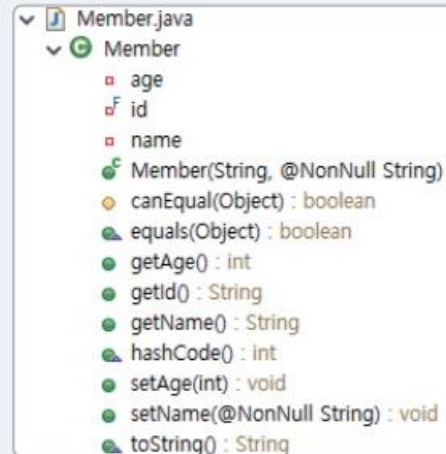
롬복 사용하기

- DTO 클래스를 작성할 때 Getter, Setter, hashCode(), equals (), toString() 메소드를 자동 생성
- 필드가 final이 아니며, 값을 읽는 Getter는 getXxx(또는 isXxx)로, 값을 변경하는 Setter는 setXxx로 생성됨

```
java -jar lombok.jar
```

>>> Member.java

```
1 package ch12.sec03.exam05;
2
3 import lombok.Data;
4 import lombok.NonNull;
5
6 @Data
7 public class Member {
8     private final String id;
9     @NonNull private String name;
10    private int age;
11 }
```



```
Member.java
└─ Member
   ├── age
   ├── id
   ├── name
   ├── Member(String, @NonNull String)
   ├── canEqual(Object) : boolean
   ├── equals(Object) : boolean
   ├── getAge() : int
   ├── getId() : String
   ├── getName() : String
   ├── hashCode() : int
   ├── setAge(int) : void
   ├── setName(@NonNull String) : void
   └── toString() : String
```


System 클래스

- System 클래스의 정적 static 필드와 메소드를 이용하면 프로그램 종료, 키보드 입력, 콘솔(모니터) 출력, 현재 시간 읽기, 시스템 프로퍼티 읽기 등이 가능

정적 멤버		용도
필드	out	콘솔(모니터)에 문자 출력
	err	콘솔(모니터)에 에러 내용 출력
	in	키보드 입력
메소드	exit(int status)	프로세스 종료
	currentTimeMillis()	현재 시간을 밀리초 단위의 long 값으로 리턴
	nanoTime()	현재 시간을 나노초 단위의 long 값으로 리턴
	getProperty()	운영체제와 사용자 정보 제공
	getenv()	운영체제의 환경 변수 정보 제공

메소드	용도
long currentTimeMillis()	1/1000 초 단위로 진행된 시간을 리턴
long nanoTime()	1/10 ⁹ 초 단위로 진행된 시간을 리턴

속성 이름(key)	설명	값(value)
java.specification.version	자바 스펙 버전	17
java.home	JDK 디렉토리 경로	C:\Program Files\Java\jdk-17.0.3
os.name	운영체제	Windows 10
user.name	사용자 이름	xxx
user.home	사용자 홈 디렉토리 경로	C:\Users\xxx
user.dir	현재 디렉토리 경로	C:\ThisIsJavaSecondEdition\workspace\thisisjava

String 클래스

- String 클래스는 문자열을 저장하고 조작할 때 사용
- 문자열 리터럴은 자동으로 String 객체로 생성. String 클래스의 다양한 생성자를 이용해서 직접 객체를 생성할 수도 있음

```
//기본 문자셋으로 byte 배열을 디코딩해서 String 객체로 생성  
String str = new String(byte[] bytes);
```

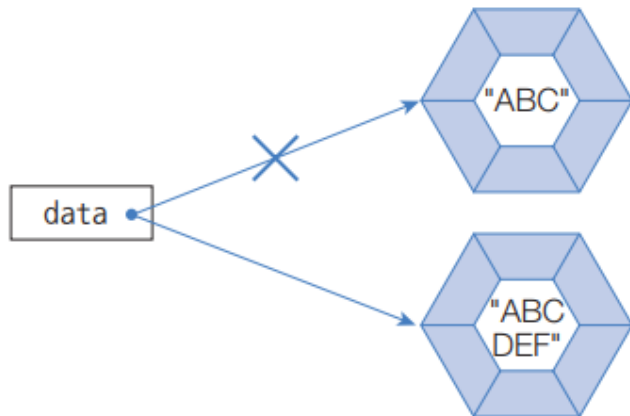
```
//특정 문자셋으로 byte 배열을 디코딩해서 String 객체로 생성  
String str = new String(byte[] bytes, String charsetName);
```

- 한글 1자를 UTF-8로 인코딩하면 3바이트가 되고, EUC-KR로 인코딩하면 2바이트가 됨

StringBuilder 클래스

- 작은 문자열 변경 작업을 해야 한다면 String보다는 StringBuilder가 좋음
- StringBuilder는 내부 버퍼에 문자열을 저장해두고 그 안에서 추가, 수정, 삭제 작업을 하도록 설계

```
String data = "ABC";  
data += "DEF";
```



리턴 타입	메소드(매개변수)	설명
StringBuilder	append(기본값 문자열)	문자열을 끝에 추가
StringBuilder	insert(위치, 기본값 문자열)	문자열을 지정 위치에 추가
StringBuilder	delete(시작 위치, 끝 위치)	문자열 일부를 삭제
StringBuilder	replace(시작 위치, 끝 위치, 문자열)	문자열 일부를 대체
String	toString()	완성된 문자열을 리턴

StringTokenizer 클래스

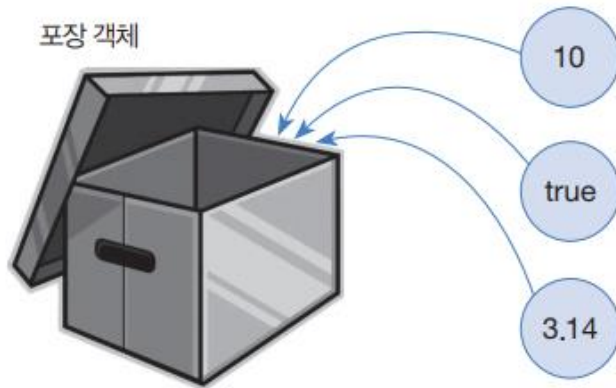
- 문자열에 여러 종류가 아닌 한 종류의 구분자만 있다면 StringTokenizer를 사용할 수도 있음
StringTokenizer 객체를 생성 시 첫 번째 매개값으로 전체 문자열을 주고, 두 번째 매개값으로 구분자를 줌. 구분자를 생략하면 공백이 기본 구분자가 됨

```
String data = "홍길동/이수홍/박연수";  
StringTokenizer st = new StringTokenizer(data, "/");
```

리턴 타입	메소드(매개변수)	설명
int	countTokens()	분리할 수 있는 문자열의 총 수
boolean	hasMoreTokens()	남아 있는 문자열이 있는지 여부
String	nextToken()	문자열을 하나씩 가져옴

포장 객체

- 기본 타입(byte, char, short, int, long, float, double, boolean)의 값을 갖는 객체
- 포장하고 있는 기본 타입의 값을 변경할 수 없고, 단지 객체로 생성하는 목적



기본 타입	포장 클래스
byte	Byte
char	Character
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

박싱과 언박싱

- 박싱: 기본 타입 값을 포장 객체로 만드는 과정. 포장 클래스 변수에 기본 타입 값이 대입 시 발생
- 언박싱: 포장 객체에서 기본 타입 값을 얻어내는 과정. 기본 타입 변수에 포장 객체가 대입 시 발생

```
Integer obj = 100;    //박싱  
int value = obj;      //언박싱
```

```
int value = obj + 50;  //언박싱 후 연산
```

문자열을 기본 타입 값으로 변환

- 포장 클래스는 문자열을 기본 타입 값으로 변환할 때도 사용.
- 대부분의 포장 클래스에는 'parse+기본타입' 명으로 되어 있는 정적 메소드 있음

포장 값 비교

- 포장 객체는 번지를 비교하므로 내부 값을 비교하기 위해 ==와 != 연산자를 사용할 수 없음

```
Integer obj1 = 300;  
Integer obj2 = 300;  
System.out.println(obj1 == obj2);
```

- 다음 범위의 값을 갖는 포장 객체는 ==와 != 연산자로 비교할 수 있지만, 내부 값을 비교하는 것이 아니라 객체 번지를 비교하는 것

타입	값의 범위
boolean	true, false
char	\u0000 ~ \u007f
byte, short, int	-128 ~ 127

- 대신 포장 클래스의 equals() 메소드로 내부 값을 비교할 수 있음

Math 클래스

- 수학 계산에 사용할 수 있는 정적 메소드 제공

구분	코드	리턴값
절대값	int v1 = Math.abs(-5); double v2 = Math.abs(-3.14);	v1 = 5 v2 = 3.14
올림값	double v3 = Math.ceil(5.3); double v4 = Math.ceil(-5.3);	v3 = 6.0 v4 = -5.0
버림값	double v5 = Math.floor(5.3); double v6 = Math.floor(-5.3);	v5 = 5.0 v6 = -6.0
최대값	int v7 = Math.max(5, 9); double v8 = Math.max(5.3, 2.5);	v7 = 9 v8 = 5.3
최소값	int v9 = Math.min(5, 9); double v10 = Math.min(5.3, 2.5);	v9 = 5 v10 = 2.5
랜덤값	double v11 = Math.random();	0.0 ≤ v11 < 1.0
반올림값	long v14 = Math.round(5.3); long v15 = Math.round(5.7);	v14 = 5 v15 = 6

Date 클래스

- 날짜를 표현하는 클래스로 객체 간에 날짜 정보를 주고받을 때 사용
- Date() 생성자는 컴퓨터의 현재 날짜를 읽어 Date 객체로 만듦

```
Date now = new Date();
```

Calendar 클래스

- 달력을 표현하는 추상 클래스
- getInstance() 메소드로 컴퓨터에 설정된 시간대 기준으로 Calendar 하위 객체를 얻을 수 있음

```
Calendar now = Calendar.getInstance();
```

날짜와 시간 조작

- java.time 패키지의 LocalDateTime 클래스가 제공하는 메소드를 이용해 날짜와 시간을 조작 가능

날짜와 시간 비교

- LocalDateTime 클래스는 날짜와 시간을 비교할 수 있는 메소드 제공

메소드(매개변수)	설명
minusYears(long)	년 빼기
minusMonths(long)	월 빼기
minusDays(long)	일 빼기
minusWeeks(long)	주 빼기
plusYears(long)	년 더하기
plusMonths(long)	월 더하기
plusWeeks(long)	주 더하기
plusDays(long)	일 더하기
minusHours(long)	시간 빼기
minusMinutes(long)	분 빼기
minusSeconds(long)	초 빼기
minusNanos(long)	나노초 빼기
plusHours(long)	시간 더하기
plusMinutes(long)	분 더하기
plusSeconds(long)	초 더하기

리턴 타입	메소드(매개변수)	설명
boolean	isAfter(other)	이후 날짜인지?
	isBefore(other)	이전 날짜인지?
	isEqual(other)	동일 날짜인지?
long	until(other, unit)	주어진 단위(unit) 차이를 리턴

DecimalFormat

- 숫자를 형식화된 문자열로 변환

기호	의미	패턴 예	1234567.89 → 변환 결과
0	10진수(빈자리는 0으로 채움)	0 0.0 0000000000.00000	1234568 1234567.9 0001234567.89000
#	10진수(빈자리는 채우지 않음)	# #.# #####.#####	1234568 1234567.9 1234567.89
.	소수점	#.0	1234567.9
-	음수 기호	+#.0 -#.0	+1234567.9 -1234567.9
,	단위 구분	#,###.0	1,234,567.9
E	지수 문자	0.0E0	1.2E6
;	양수와 음수의 패턴을 모두 기술할 경우, 패턴 구분자	+#,### ; -#,###	+1,234,568(양수일 때) -1,234,568(음수일 때)
%	% 문자	## %	123456789 %
\u00A4	통화 기호	\u00A4 #,###	₩1,234,568

SimpleDateFormat

- 날짜를 형식화된 문자열로 변환

패턴 문자	의미	패턴 문자	의미
y	년	H	시(0~23)
M	월	h	시(1~12)
d	일	K	시(0~11)
D	월 구분이 없는 일(1~365)	k	시(1~24)
E	요일	m	분
a	오전/오후	s	초
w	년의 몇 번째 주	S	밀리세컨드(1/1000초)
W	월의 몇 번째 주		

정규 표현식

- 문자 또는 숫자와 관련된 표현과 반복 기호가 결합된 문자열

표현 및 기호	설명
[]	[abc] a, b, c 중 하나의 문자
	[^abc] a, b, c 이외의 하나의 문자
	[a-zA-Z] a~z, A~Z 중 하나의 문자
\d	한 개의 숫자, [0-9]와 동일
\s	공백
\w	한 개의 알파벳 또는 한 개의 숫자, [a-zA-Z_0-9]와 동일
\.	.
.	모든 문자 중 한 개의 문자
?	없음 또는 한 개
*	없음 또는 한 개 이상
+	한 개 이상
{n}	정확히 n개
{n,}	최소한 n개
{n, m}	n개부터 m개까지
a b	a 또는 b
()	그룹핑

Pattern 클래스로 검증

- java.util.regex 패키지의 Pattern 클래스는 정규 표현식으로 문자열을 검증하는 matches() 메소드 제공

```
boolean result = Pattern.matches("정규식", "검증할 문자열");
```

리플렉션

- Class 객체로 관리하는 클래스와 인터페이스의 메타 정보를 프로그램에서 읽고 수정하는 것
- 메타 정보: 패키지 정보, 타입 정보, 멤버(생성자, 필드, 메소드) 정보

```
① Class clazz = 클래스이름.class;
② Class clazz = Class.forName("패키지...클래스이름");
③ Class clazz = 객체참조변수.getClass();
```

□ 클래스로부터 얻는 방법
— 객체로부터 얻는 방법

패키지와 타입 정보 얻기

- 패키지과 타입(클래스, 인터페이스) 이름 정보는 다음 메소드로 얻을 수 있음

메소드	용도
Package getPackage()	패키지 정보 읽기
String getSimpleName()	패키지를 제외한 타입 이름
String getName()	패키지를 포함한 전체 타입 이름

리소스 경로 얻기

- Class 객체는 클래스 파일(~.class)의 경로 정보를 기준으로 상대 경로에 있는 다른 리소스 파일(이미지, XML, Property 파일)의 정보를 얻을 수 있음

메소드	용도
URL getResource(String name)	리소스 파일의 URL 리턴
InputStream getResourceAsStream(String name)	리소스 파일의 InputStream 리턴

```
C:\app\bin
| - Car.class
| - photo1.jpg
| - images
|   | - photo2.jpg
```

어노테이션

- 코드에서 @으로 작성되는 요소. 클래스 또는 인터페이스를 컴파일하거나 실행할 때 어떻게 처리해야 할 것인지를 알려주는 설정 정보
- ① 컴파일 시 사용하는 정보 전달
- ② 빌드 툴이 코드를 자동으로 생성할 때 사용하는 정보 전달
- ③ 실행 시 특정 기능을 처리할 때 사용하는 정보 전달

어노테이션 타입 정의와 적용

- @interface 뒤에 사용할 어노테이션 이름 작성

```
public @interface AnnotationName {  
}
```

```
@AnnotationName
```


어노테이션 적용 대상

- 어노테이션을 적용할 수 있는 대상의 종류는 ElementType 열거 상수로 정의
- @Target 어노테이션으로 적용 대상 지정. @Target의 기본 속성 value 값은 ElementType 배열

ElementType 열거 상수	적용 요소
TYPE	클래스, 인터페이스, 열거 타입
ANNOTATION_TYPE	어노테이션
FIELD	필드
CONSTRUCTOR	생성자
METHOD	메소드
LOCAL_VARIABLE	로컬 변수
PACKAGE	패키지

```
@Target( { ElementType.TYPE, ElementType.FIELD, ElementType.METHOD } )
public @interface AnnotationName {
}
```

```
@AnnotationName
public class ClassName {
    @AnnotationName
    private String fieldName;

    //@AnnotationName
    public ClassName() { }

    @AnnotationName
    public void methodName() { }
}
```

TYPE(클래스)에 적용

필드에 적용

@Target에 CONSTRUCT가 없으므로 생성자에는 적용 못함

메소드에 적용

어노테이션 유지 정책

- 어노테이션 정의 시 `@AnnotationName`을 언제까지 유지할지 지정
- 어노테이션 유지 정책은 `RetentionPolicy` 열거 상수로 정의

RetentionPolicy 열거 상수	어노테이션 적용 시점	어노테이션 제거 시점
SOURCE	컴파일할 때 적용	컴파일된 후에 제거됨
CLASS	메모리로 로딩할 때 적용	메모리로 로딩된 후에 제거됨
RUNTIME	실행할 때 적용	계속 유지됨

어노테이션 설정 정보 이용

- 애플리케이션은 리플렉션을 이용해 적용 대상에서 어노테이션의 정보를 다음 메소드로 얻어냄

리턴 타입	메소드명(매개변수)	설명
boolean	<code>isAnnotationPresent(AnnotationName.class)</code>	지정한 어노테이션이 적용되었는지 여부
Annotation	<code>getAnnotation(AnnotationName.class)</code>	지정한 어노테이션이 적용되어 있으면 어노테이션을 리턴하고, 그렇지 않다면 null을 리턴
Annotation[]	<code>getDeclaredAnnotations()</code>	적용된 모든 어노테이션을 리턴

Thank you!