

10 . java.base 모듈

10.1 API 도큐먼트

10.2 java.base 모듈

10.3 Object 클래스

10.4 System 클래스

10.5 문자열 클래스

10.6 포장 클래스

10.7 수학 클래스

10.8 날짜와 시간 클래스

10.9 형식 클래스

10.10 정규 표현식 클래스

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

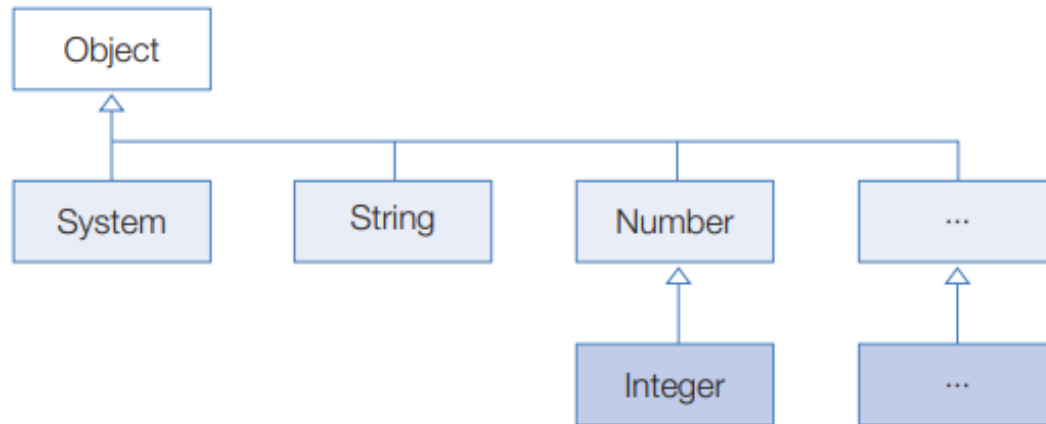
- 자바 언어의 기본적인 클래스를 담고 있는 패키지. 이 패키지에 있는 클래스와 인터페이스는 import 없이 사용할 수 있음

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

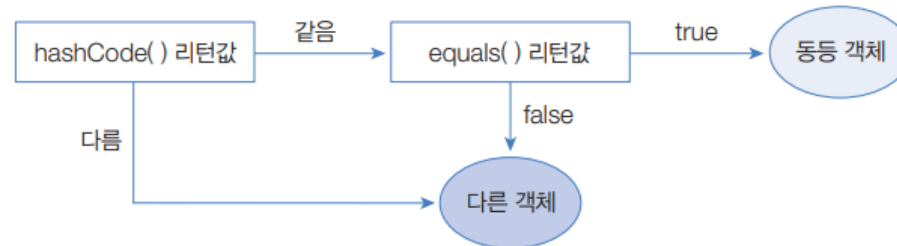
```
public boolean equals(Object obj)
```

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) {  
}
```

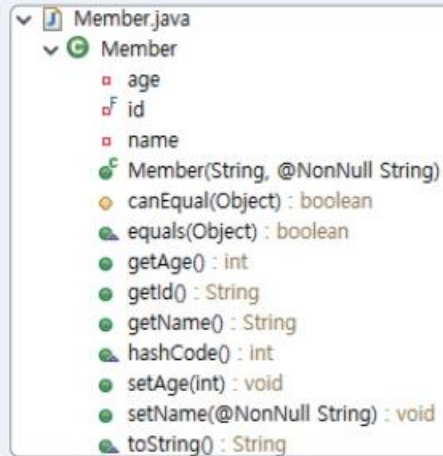
롬복 사용하기

- 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);
```

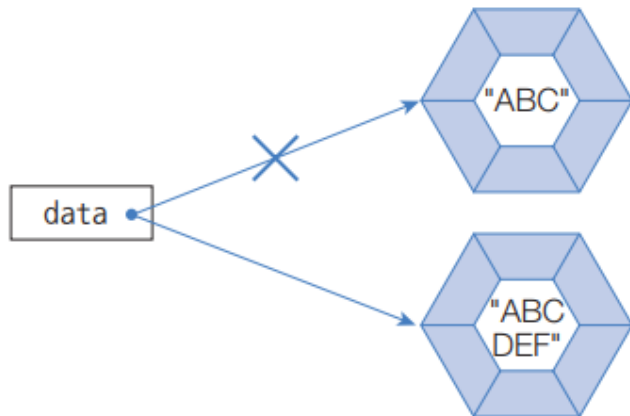
```
//특정 문자셋으로 byte 배열을 디코딩해서 String 객체로 생성  
String str = new String(byte[] bytes, String charsetName);
```

- 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

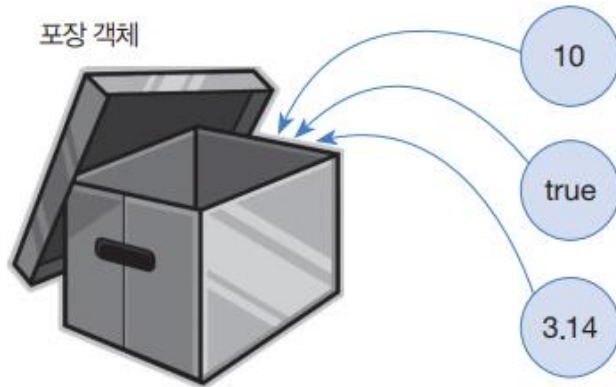
- 문자열에 여러 종류가 아닌 한 종류의 구분자만 있다면 StringTokenizer를 사용할 수도 있음
StringTokenizer 객체를 생성 시 첫 번째 매개값으로 전체 문자열을 주고, 두 번째 매개값으로 구분자를 줌. 구분자를 생략하면 공백이 기본 구분자가 됨

```
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
```

11 . 제너릭

11.1 제네릭이란?

11.2 제네릭 타입

11.3 제네릭 메소드

11.4 제한된 타입 파라미터

11.5 와일드카드 타입 파라미터

제네릭

- 결정되지 않은 타입을 파라미터로 처리하고 실제 사용할 때 파라미터를 구체적인 타입으로 대체시키는 기능
- <T>는 T가 타입 파라미터임을 뜻하는 기호. 타입이 필요한 자리에 T를 사용할 수 있음을 알려줌

```
public class Box <T> {  
    public T content;  
}
```

```
Box<String> box = new Box<String>();
```



```
Box<String> box = new Box<>();
```

```
Box<Integer> box = new Box<Integer>();
```



```
Box<Integer> box = new Box<>();
```

제네릭 타입

- 결정되지 않은 타입을 파라미터로 가지는 클래스와 인터페이스
- 선언부에 '<>' 부호가 붙고 그 사이에 타입 파라미터들이 위치

```
public class 클래스명<A, B, ...> { ... }  
public interface 인터페이스명<A, B, ...> { ... }
```

- 타입 파라미터는 일반적으로 대문자 알파벳 한 글자로 표현
- 외부에서 제네릭 타입을 사용하려면 타입 파라미터에 구체적인 타입을 지정. 지정하지 않으면 Object 타입이 암묵적으로 사용

제네릭 메소드

- 타입 피라미터를 가지고 있는 메소드. 타입 파라미터가 메소드 선언부에 정의
- 리턴 타입 앞에 <> 기호 추가하고 타입 파라미터 정의 후 리턴 타입과 매개변수 타입에서 사용

```
public <A, B, ...> 리턴타입 메소드명(매개변수, ...) { ... }
```

↑
타입 파라미터 정의

- 타입 파라미터 T는 매개값의 타입에 따라 컴파일 과정에서 구체적인 타입으로 대체

```
public <T> Box<T> boxing(T t) { ... }
```

- ① `Box<Integer> box1 = boxing(100);`
- ② `Box<String> box2 = boxing("안녕하세요");`

제한된 타입 파라미터

- 모든 타입으로 대체할 수 없고, 특정 타입과 자식 또는 구현 관계에 있는 타입만 대체할 수 있는 타입 파라미터

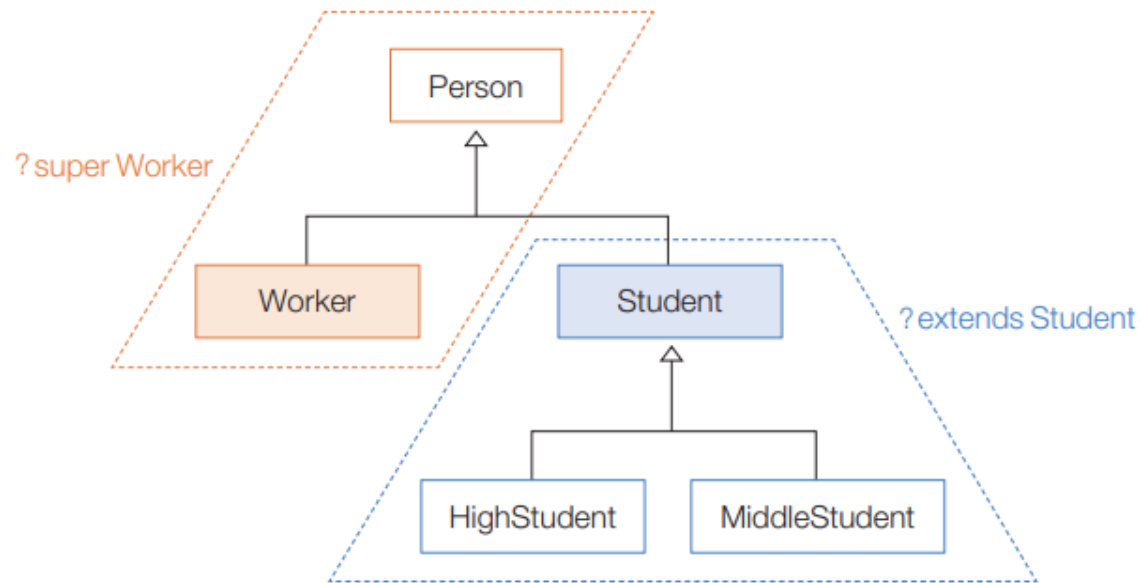
```
public <T extends 상위타입> 리턴타입 메소드(매개변수, ...) { ... }
```

- 상위 타입은 클래스뿐만 아니라 인터페이스도 가능

```
public <T extends Number> boolean compare(T t1, T t2) {  
    double v1 = t1.doubleValue(); //Number의 doubleValue() 메소드 사용  
    double v2 = t2.doubleValue(); //Number의 doubleValue() 메소드 사용  
    return (v1 == v2);  
}
```

와일드카드 타입 파라미터

- 제네릭 타입을 매개값이나 리턴 타입으로 사용할 때 범위에 있는 모든 타입으로 대체할 수 있는 타입 파라미터. ?로 표시



리턴타입 메소드명(제네릭타입<? extends Student> 변수) { ... }

리턴타입 메소드명(제네릭타입<? super Worker> 변수) { ... }

리턴타입 메소드명(제네릭타입<?> 변수) { ... }

12 . 컬렉션 프레임워크

12.1 컬렉션 프레임워크

12.2 List 컬렉션

12.3 Set 컬렉션

12.4 Map 컬렉션

12.5 검색 기능을 강화시킨 컬렉션

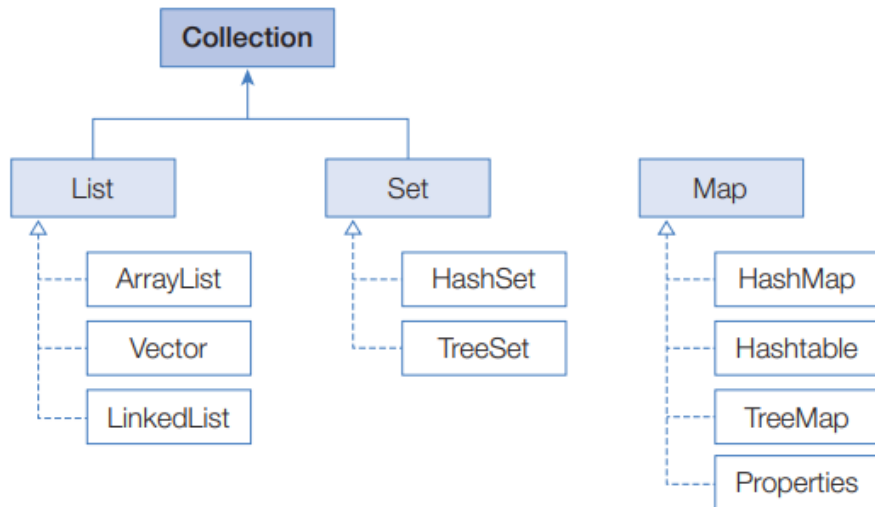
12.6 LIFO와 FIFO 컬렉션

12.7 동기화된 컬렉션

12.8 수정할 수 없는 컬렉션

컬렉션 프레임워크

- 널리 알려진 자료구조를 바탕으로 객체들을 효율적으로 추가, 삭제, 검색할 수 있도록 관련 인터페이스와 클래스들을 포함시켜 놓은 java.util 패키지
- 주요 인터페이스: List, Set, Map



인터페이스 분류		특징	구현 클래스
Collection	List	- 순서를 유지하고 저장 - 중복 저장 가능	ArrayList, Vector, LinkedList
	Set	- 순서를 유지하지 않고 저장 - 중복 저장 안됨	HashSet, TreeSet
Map		- 키와 값으로 구성된 엔트리 저장 - 키는 중복 저장 안됨	HashMap, Hashtable, TreeMap, Properties

List 컬렉션

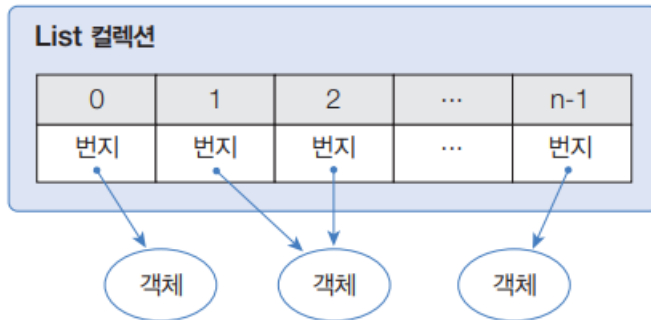
- 객체를 인덱스로 관리하기 때문에 객체를 저장하면 인덱스가 부여되고 인덱스로 객체를 검색, 삭제할 수 있는 기능을 제공

기능	메소드	설명
객체 추가	boolean add(E e)	주어진 객체를 맨 끝에 추가
	void add(int index, E element)	주어진 인덱스에 객체를 추가
	set(int index, E element)	주어진 인덱스의 객체를 새로운 객체로 바꿈
객체 검색	boolean contains(Object o)	주어진 객체가 저장되어 있는지 여부
	E get(int index)	주어진 인덱스에 저장된 객체를 리턴
	isEmpty()	컬렉션이 비어 있는지 조사
	int size()	저장되어 있는 전체 객체 수를 리턴
객체 삭제	void clear()	저장된 모든 객체를 삭제
	E remove(int index)	주어진 인덱스에 저장된 객체를 삭제
	boolean remove(Object o)	주어진 객체를 삭제

12.2 List 컬렉션

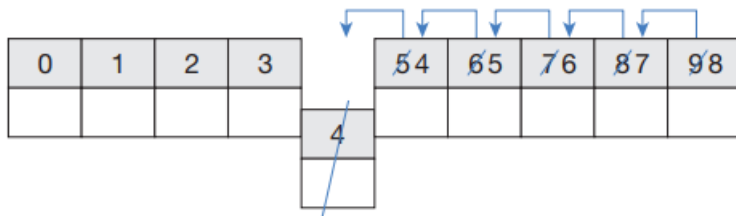
ArrayList

- ArrayList에 객체를 추가하면 내부 배열에 객체가 저장되고 제한 없이 객체를 추가할 수 있음
- 객체의 번지를 저장. 동일한 객체를 중복 저장 시 동일한 번지가 저장. null 저장 가능



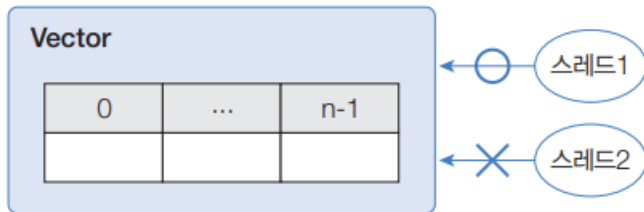
```
List<E> list = new ArrayList<E>(); //E에 지정된 타입의 객체만 저장  
List<E> list = new ArrayList<>(); //E에 지정된 타입의 객체만 저장  
List list = new ArrayList(); //모든 타입의 객체를 저장
```

- ArrayList 컬렉션에 객체를 추가 시 인덱스 0번부터 차례대로 저장
- 특정 인덱스의 객체를 제거하거나 삽입하면 전체가 앞/뒤로 1씩 당겨지거나 밀림
- 빈번한 객체 삭제와 삽입이 일어나는 곳에선 바람직하지 않음



Vector

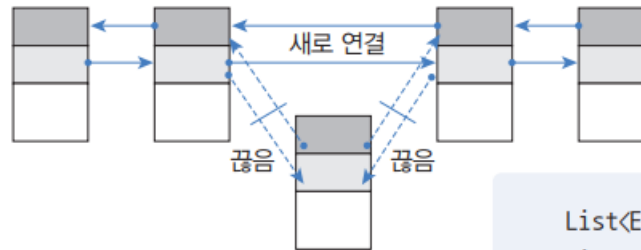
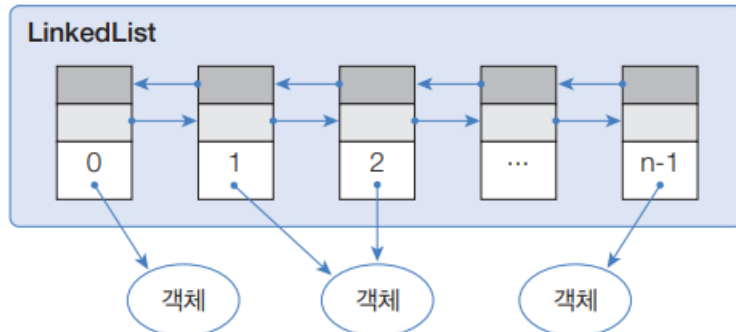
- 동기화된 메소드로 구성되어 있어 멀티 스레드가 동시에 Vector() 메소드를 실행할 수 없음
- 멀티 스레드 환경에서는 안전하게 객체를 추가 또는 삭제할 수 있음



```
List<E> list = new Vector<E>(); //E에 지정된 타입의 객체만 저장  
List<E> list = new Vector<>(); //E에 지정된 타입의 객체만 저장  
List list = new Vector(); //모든 타입의 객체를 저장
```

LinkedList

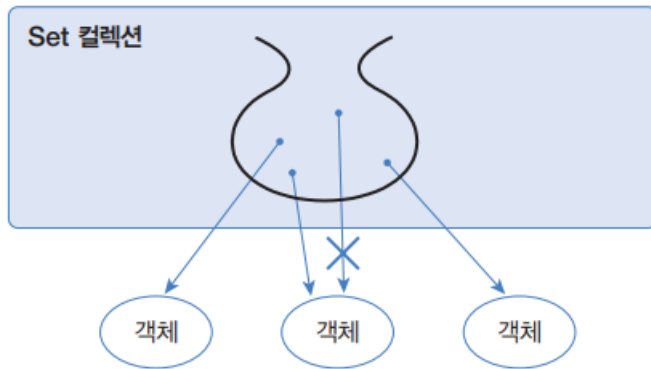
- 인접 객체를 체인처럼 연결해서 관리. 객체 삭제와 삽입이 빈번한 곳에서 ArrayList보다 유리



```
List<E> list = new LinkedList<E>(); //E에 지정된 타입의 객체만 저장  
List<E> list = new LinkedList<>(); //E에 지정된 타입의 객체만 저장  
List list = new LinkedList(); //모든 타입의 객체를 저장
```

Set 컬렉션

- Set 컬렉션은 저장 순서가 유지되지 않음
- 객체를 중복해서 저장할 수 없고, 하나의 null만 저장할 수 있음(수학의 집합 개념)

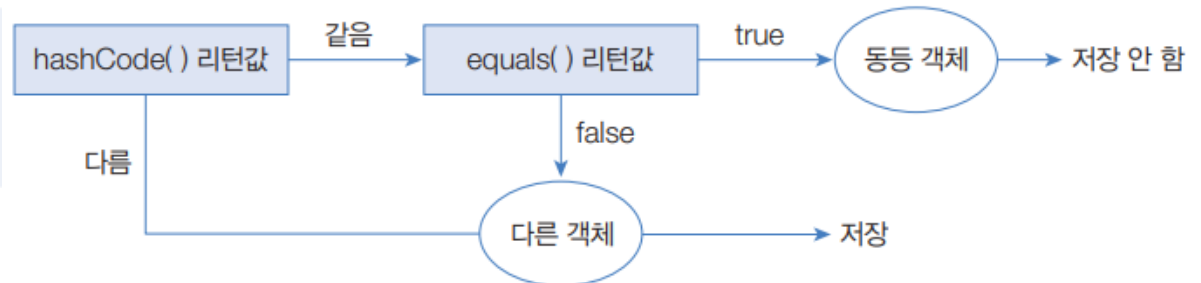


기능	메소드	설명
객체 추가	<code>boolean add(E e)</code>	주어진 객체를 성공적으로 저장하면 <code>true</code> 를 리턴하고 중복 객체면 <code>false</code> 를 리턴
	<code>boolean contains(Object o)</code>	주어진 객체가 저장되어 있는지 여부
객체 검색	<code>isEmpty()</code>	컬렉션이 비어 있는지 조사
	<code>Iterator<E> iterator()</code>	저장된 객체를 한 번씩 가져오는 반복자 리턴
	<code>int size()</code>	저장되어 있는 전체 객체 수 리턴
	<code>void clear()</code>	저장된 모든 객체를 삭제
객체 삭제	<code>boolean remove(Object o)</code>	주어진 객체를 삭제

HashSet

- 동등 객체를 중복 저장하지 않음
- 다른 객체라도 hashCode() 메소드의 리턴값이 같고, equals() 메소드가 true를 리턴하면 동일한 객체라고 판단하고 중복 저장하지 않음

```
Set<E> set = new HashSet<E>(); //E에 지정된 타입의 객체만 저장
Set<E> set = new HashSet<>(); //E에 지정된 타입의 객체만 저장
Set set = new HashSet(); //모든 타입의 객체를 저장
```



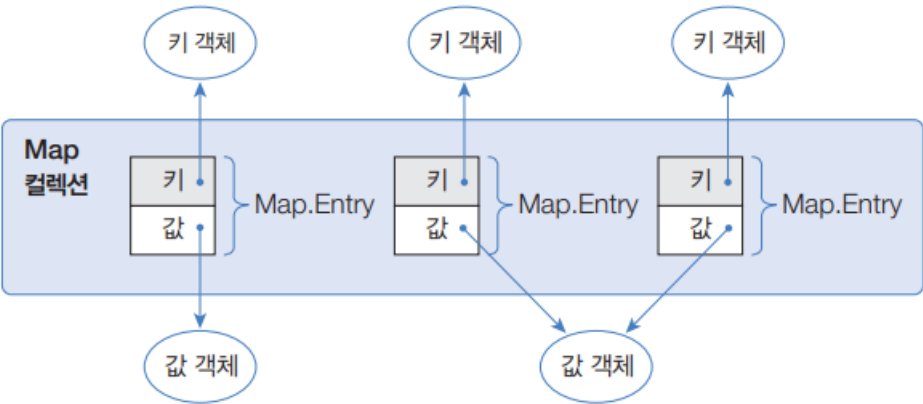
- iterator() 메소드: 반복자를 얻어 Set 컬렉션의 객체를 하나씩 가져옴

```
Set<E> set = new HashSet<>();
Iterator<E> iterator = set.iterator();
```

리턴 타입	메소드명	설명
boolean	hasNext()	가져올 객체가 있으면 true를 리턴하고 없으면 false를 리턴한다.
E	next()	컬렉션에서 하나의 객체를 가져온다.
void	remove()	next()로 가져온 객체를 Set 컬렉션에서 제거한다.

Map 컬렉션

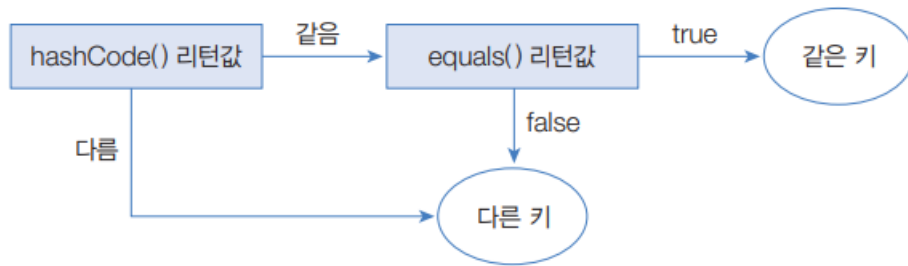
- 키와 값으로 구성된 엔트리 객체를 저장
- 키는 중복 저장할 수 없지만 값은 중복 저장할 수 있음. 기존에 저장된 키와 동일한 키로 값을 저장하면 새로운 값으로 대체



기능	메소드	설명
객체 추가	V put(K key, V value)	주어진 키와 값을 추가, 저장이 되면 값을 리턴
	boolean containsKey(Object key)	주어진 키가 있는지 여부
	boolean containsValue(Object value)	주어진 값이 있는지 여부
객체 검색	Set<Map.Entry<K,V>> entrySet()	키와 값의 쌍으로 구성된 모든 Map.Entry 객체를 Set에 담아서 리턴
	V get(Object key)	주어진 키의 값을 리턴
	boolean isEmpty()	컬렉션이 비어있는지 여부
	Set<K> keySet()	모든 키를 Set 객체에 담아서 리턴
	int size()	저장된 키의 총 수를 리턴
	Collection<V> values()	저장된 모든 값 Collection에 담아서 리턴
객체 삭제	void clear()	모든 Map.Entry(키와 값)를 삭제
	V remove(Object key)	주어진 키와 일치하는 Map.Entry 삭제, 삭제가 되면 값을 리턴

HashMap

- 키로 사용할 객체가 hashCode() 메소드의 리턴값이 같고 equals() 메소드가 true를 리턴할 경우 동일 키로 보고 중복 저장을 허용하지 않음

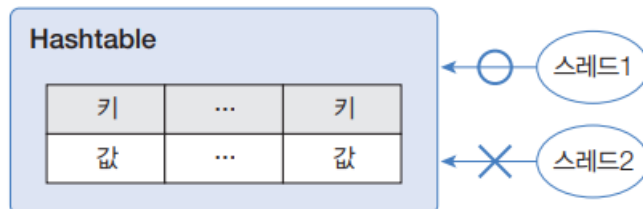


```
Map<K, V> map = new HashMap<K, V>();
```

↑ ↑ ↑ ↑
키 타입 값 타입 키 타입 값 타입

Hashtable

- 동기화된 메소드로 구성되어 있어 멀티 스레드가 동시에 Hashtable의 메소드들을 실행 불가
- 멀티 스레드 환경에서도 안전하게 객체를 추가, 삭제할 수 있다.



```
Map<String, Integer> map = new Hashtable<String, Integer>();  
Map<String, Integer> map = new Hashtable<>();
```


Properties

- Properties는 Hashtable의 자식 클래스. 키와 값을 String 타입으로 제한한 컬렉션
- 주로 확장자가 .properties인 프로퍼티 파일을 읽을 때 사용
- 프로퍼티 파일은 키와 값이 = 기호로 연결된 텍스트 파일(ISO 8859-1 문자셋, 한글은 \u+유니코드)
- Properties 객체를 생성하고, load() 메소드로 프로퍼티 파일의 내용을 메모리로 로드

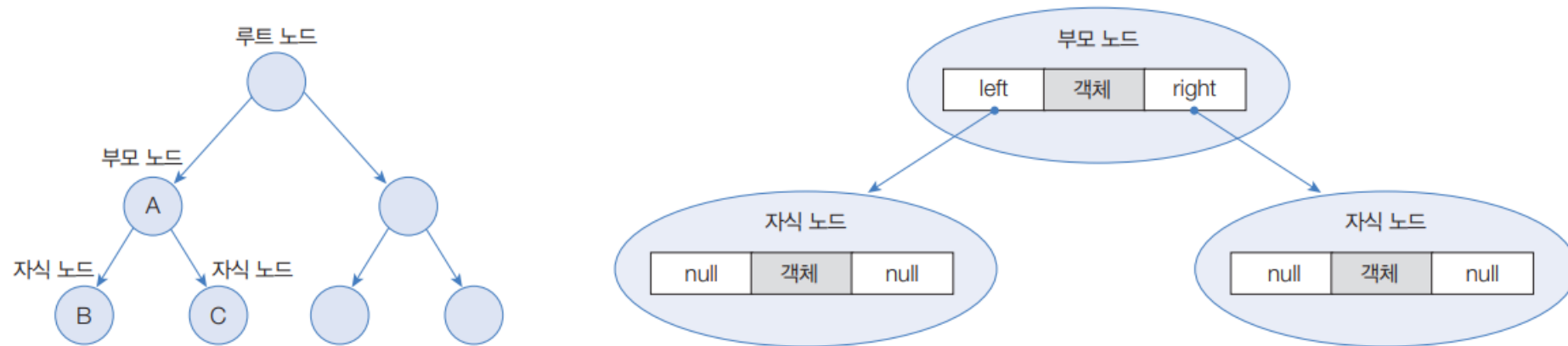
```
>>> database.properties
```

```
1 driver=oracle.jdbc.OracleDriver
2 url=jdbc:oracle:thin:@localhost:1521:orcl
3 username=scott
4 password=tiger
5 admin=\uD64D\uAE38\uB3D9
```

```
Properties properties = new Properties();
properties.load(Xxx.class.getResourceAsStream("database.properties"));
```

TreeSet

- 이진 트리를 기반으로 한 Set 컬렉션
- 여러 개의 노드가 트리 형태로 연결된 구조. 루트 노드에서 시작해 각 노드에 최대 2개의 노드를 연결할 수 있음
- TreeSet에 객체를 저장하면 부모 노드의 객체와 비교해서 낮은 것은 왼쪽 자식 노드에, 높은 것은 오른쪽 자식 노드에 저장



TreeSet 컬렉션을 생성하는 방법

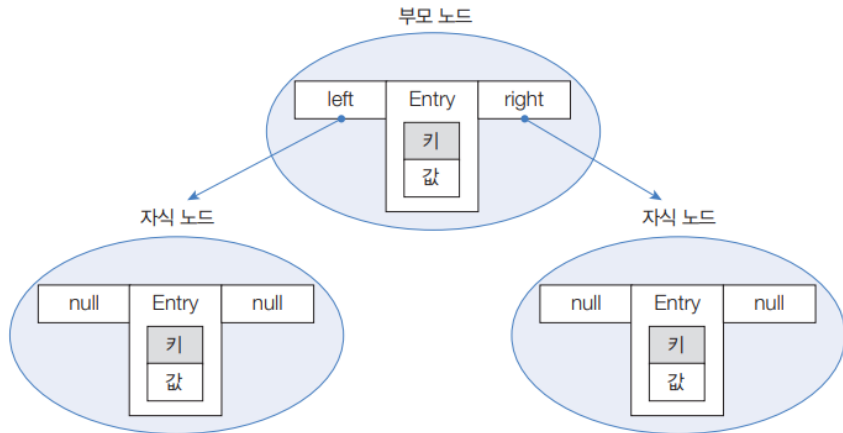
```
TreeSet<E> treeSet = new TreeSet<E>();  
TreeSet<E> treeSet = new TreeSet<>();
```

- Set 타입 변수에 대입해도 되지만 TreeSet 타입으로 대입한 이유는 검색 관련 메소드가 TreeSet에만 정의되어 있기 때문

리턴 타입	메소드	설명
E	first()	제일 낮은 객체를 리턴
E	last()	제일 높은 객체를 리턴
E	lower(E e)	주어진 객체보다 바로 아래 객체를 리턴
E	higher(E e)	주어진 객체보다 바로 위 객체를 리턴
E	floor(E e)	주어진 객체와 동등한 객체가 있으면 리턴, 만약 없다면 주어진 객체의 바로 아래의 객체를 리턴
E	ceiling(E e)	주어진 객체와 동등한 객체가 있으면 리턴, 만약 없다면 주어진 객체의 바로 위의 객체를 리턴
E	pollFirst()	제일 낮은 객체를 꺼내고 컬렉션에서 제거함
E	pollLast()	제일 높은 객체를 꺼내고 컬렉션에서 제거함
Iterator<E>	descendingIterator()	내림차순으로 정렬된 Iterator를 리턴
NavigableSet<E>	descendingSet()	내림차순으로 정렬된 NavigableSet을 리턴
NavigableSet<E>	headSet(E toElement, boolean inclusive)	주어진 객체보다 낮은 객체들을 NavigableSet으로 리턴. 주어진 객체 포함 여부는 두 번째 매개값에 따라 달라짐
NavigableSet<E>	tailSet(E fromElement, boolean inclusive)	주어진 객체보다 높은 객체들을 NavigableSet으로 리턴. 주어진 객체 포함 여부는 두 번째 매개값에 따라 달라짐
NavigableSet<E>	subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive)	시작과 끝으로 주어진 객체 사이의 객체들을 NavigableSet으로 리턴. 시작과 끝 객체의 포함 여부는 두 번째, 네 번째 매개값에 따라 달라짐

TreeMap

- 이진 트리를 기반으로 한 Map 컬렉션. 키와 값이 저장된 엔트리 저장
- 부모 키 값과 비교해서 낮은 것은 왼쪽, 높은 것은 오른쪽 자식 노드에 Entry 객체를 저장



```
TreeMap<K, V> treeMap = new TreeMap<K, V>();
TreeMap<K, V> treeMap = new TreeMap<>();
```

리턴 타입	메소드	설명
Map.Entry<K,V>	firstEntry()	제일 낮은 Map.Entry를 리턴
Map.Entry<K,V>	lastEntry()	제일 높은 Map.Entry를 리턴
Map.Entry<K,V>	lowerEntry(K key)	주어진 키보다 바로 아래 Map.Entry를 리턴
Map.Entry<K,V>	higherEntry(K key)	주어진 키보다 바로 위 Map.Entry를 리턴
Map.Entry<K,V>	floorEntry(K key)	주어진 키와 동등한 키가 있으면 해당 Map.Entry를 리턴. 없다면 주어진 키 바로 아래의 Map.Entry를 리턴
Map.Entry<K,V>	ceilingEntry(K key)	주어진 키와 동등한 키가 있으면 해당 Map.Entry를 리턴. 없다면 주어진 키 바로 위의 Map.Entry를 리턴
Map.Entry<K,V>	pollFirstEntry()	제일 낮은 Map.Entry를 꺼내고 컬렉션에서 제거함
Map.Entry<K,V>	pollLastEntry()	제일 높은 Map.Entry를 꺼내고 컬렉션에서 제거함
NavigableSet<K>	descendingKeySet()	내림차순으로 정렬된 키의 NavigableSet을 리턴
NavigableMap<K,V>	descendingMap()	내림차순으로 정렬된 Map.Entry의 NavigableMap을 리턴
NavigableMap<K,V>	headMap(K toKey, boolean inclusive)	주어진 키보다 낮은 Map.Entry들을 NavigableMap으로 리턴. 주어진 키의 Map.Entry 포함 여부는 두 번째 매개값에 따라 달라짐
NavigableMap<K,V>	tailMap(K fromKey, boolean inclusive)	주어진 객체보다 높은 Map.Entry들을 NavigableSet으로 리턴. 주어진 객체 포함 여부는 두 번째 매개값에 따라 달라짐
NavigableMap<K,V>	subMap(K fromKey, boolean fromInclusive, K toKey, boolean toInclusive)	시작과 끝으로 주어진 키 사이의 Map.Entry들을 NavigableMap 컬렉션으로 반환. 시작과 끝 키의 Map.Entry 포함 여부는 두 번째, 네 번째 매개값에 따라 달라짐

Comparable과 Comparator

- TreeSet에 저장되는 객체와 TreeMap에 저장되는 키 객체를 정렬
- Comparable 인터페이스에는 compareTo() 메소드가 정의. 사용자 정의 클래스에서 이 메소드를 재정의해서 비교 결과를 정수 값으로 리턴

리턴 타입	메소드	설명
int	compareTo(T o)	주어진 객체와 같으면 0을 리턴 주어진 객체보다 적으면 음수를 리턴 주어진 객체보다 크면 양수를 리턴

- 비교 기능이 없는 Comparable 비구현 객체를 저장하려면 비교자 Comparator를 제공
- 비교자는 compare() 메소드를 재정의해서 비교 결과를 정수 값으로 리턴

```
TreeSet<E> treeSet = new TreeSet<E>( new ComparatorImpl() );
```

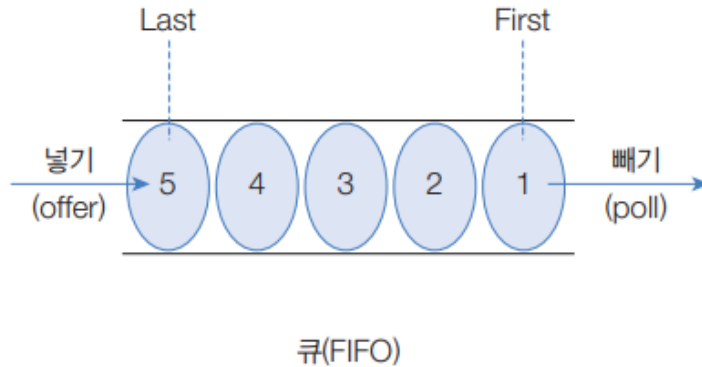
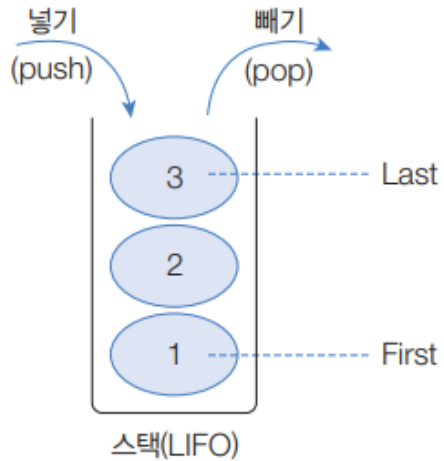
비교자

```
TreeMap<K,V> treeMap = new TreeMap<K,V>( new ComparatorImpl() );
```

리턴 타입	메소드	설명
int	compare(T o1, T o2)	o1과 o2가 동등하다면 0을 리턴 o1이 o2보다 앞에 오게 하려면 음수를 리턴 o1이 o2보다 뒤에 오게 하려면 양수를 리턴

후입선출과 선입선출

- 후입선출(LIFO): 나중에 넣은 객체가 먼저 빠져나가는 구조
- 선입선출(FIFO): 먼저 넣은 객체가 먼저 빠져나가는 구조
- 컬렉션 프레임워크는 LIFO 자료구조를 제공하는 스택 클래스와 FIFO 자료구조를 제공하는 큐 인터페이스를 제공



Stack

- Stack 클래스: LIFO 자료구조를 구현한 클래스

```
Stack<E> stack = new Stack<E>();  
Stack<E> stack = new Stack<>();
```

리턴 타입	메소드	설명
E	push(E item)	주어진 객체를 스택에 넣는다.
E	pop()	스택의 맨 위 객체를 빼낸다.

Queue

- Queue 인터페이스: FIFO 자료구조에서 사용되는 메소드를 정의
- LinkedList: Queue 인터페이스를 구현한 대표적인 클래스

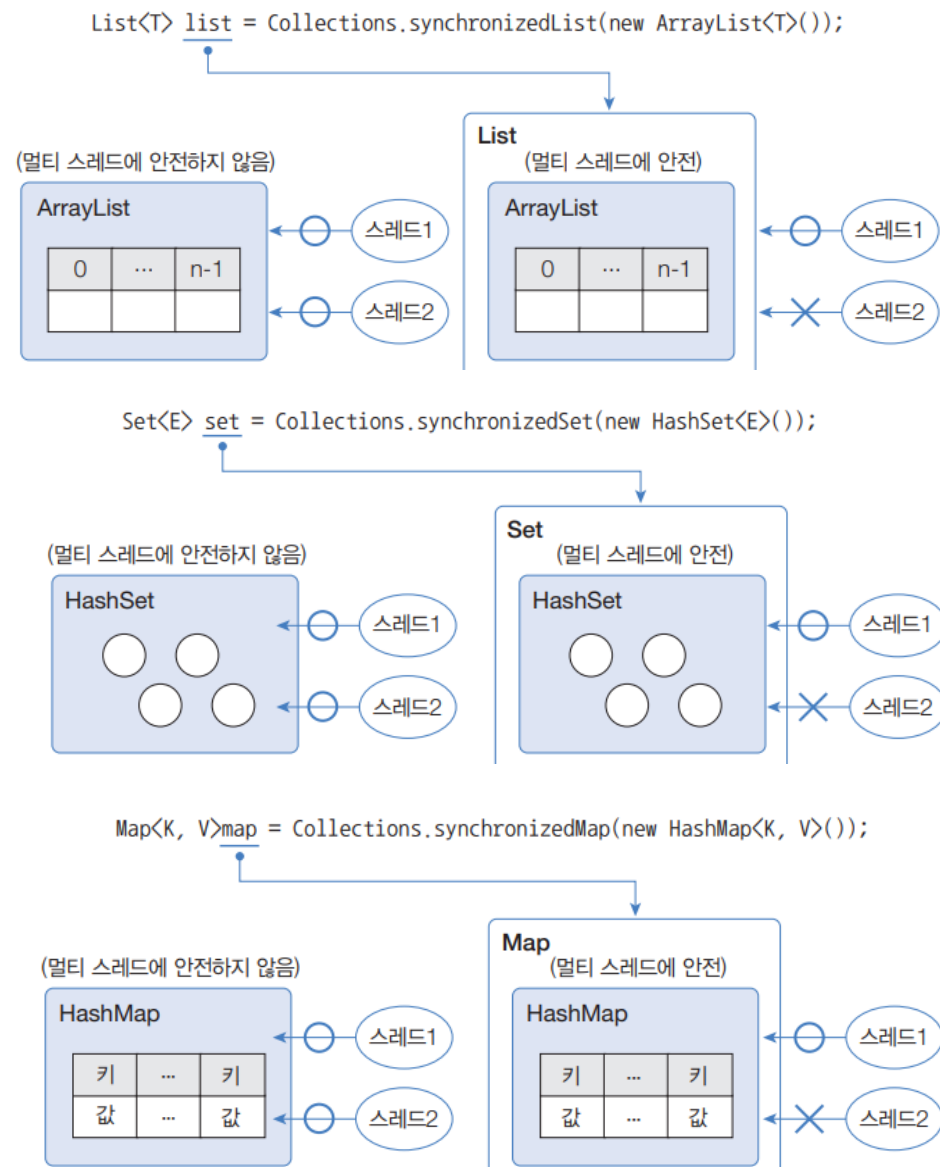
리턴 타입	메소드	설명
boolean	offer(E e)	주어진 객체를 큐에 넣는다.
E	poll()	큐에서 객체를 빼낸다.

```
Queue<E> queue = new LinkedList<E>();  
Queue<E> queue = new LinkedList<>();
```

동기화된 컬렉션

- 동기화된 메소드로 구성된 Vector와 Hashtable은 멀티 스레드 환경에서 안전하게 요소를 처리
- Collections의 synchronizedXXX() 메소드: ArrayList, HashSet, HashMap 등 비동기화된 메소드를 동기화된 메소드로 래핑

리턴 타입	메소드(매개변수)	설명
List<T>	synchronizedList(List<T> list)	List를 동기화된 List로 리턴
Map<K,V>	synchronizedMap(Map<K,V> m)	Map을 동기화된 Map으로 리턴
Set<T>	synchronizedSet(Set<T> s)	Set을 동기화된 Set으로 리턴



수정할 수 없는 컬렉션

- 요소를 추가, 삭제할 수 없는 컬렉션. 컬렉션 생성 시 저장된 요소를 변경하고 싶지 않을 때 유용
- List, Set, Map 인터페이스의 정적 메소드인 `of()`로 생성
- List, Set, Map 인터페이스의 정적 메소드인 `copyOf()`을 이용해 기존 컬렉션을 복사
- 배열로부터 수정할 수 없는 List 컬렉션을 만들

```
List<E> immutableList = List.of(E... elements);  
Set<E> immutableSet = Set.of(E... elements);  
Map<K,V> immutableMap = Map.of( K k1, V v1, K k2, V v2, ... );
```

```
List<E> immutableList = List.copyOf(Collection<E> coll);  
Set<E> immutableSet = Set.copyOf(Collection<E> coll);  
Map<K,V> immutableMap = Map.copyOf(Map<K,V> map);
```

```
String[] arr = { "A", "B", "C" };  
List<String> immutableList = Arrays.asList(arr);
```