KB금융그룹 | 국민의 평생 금융파트너

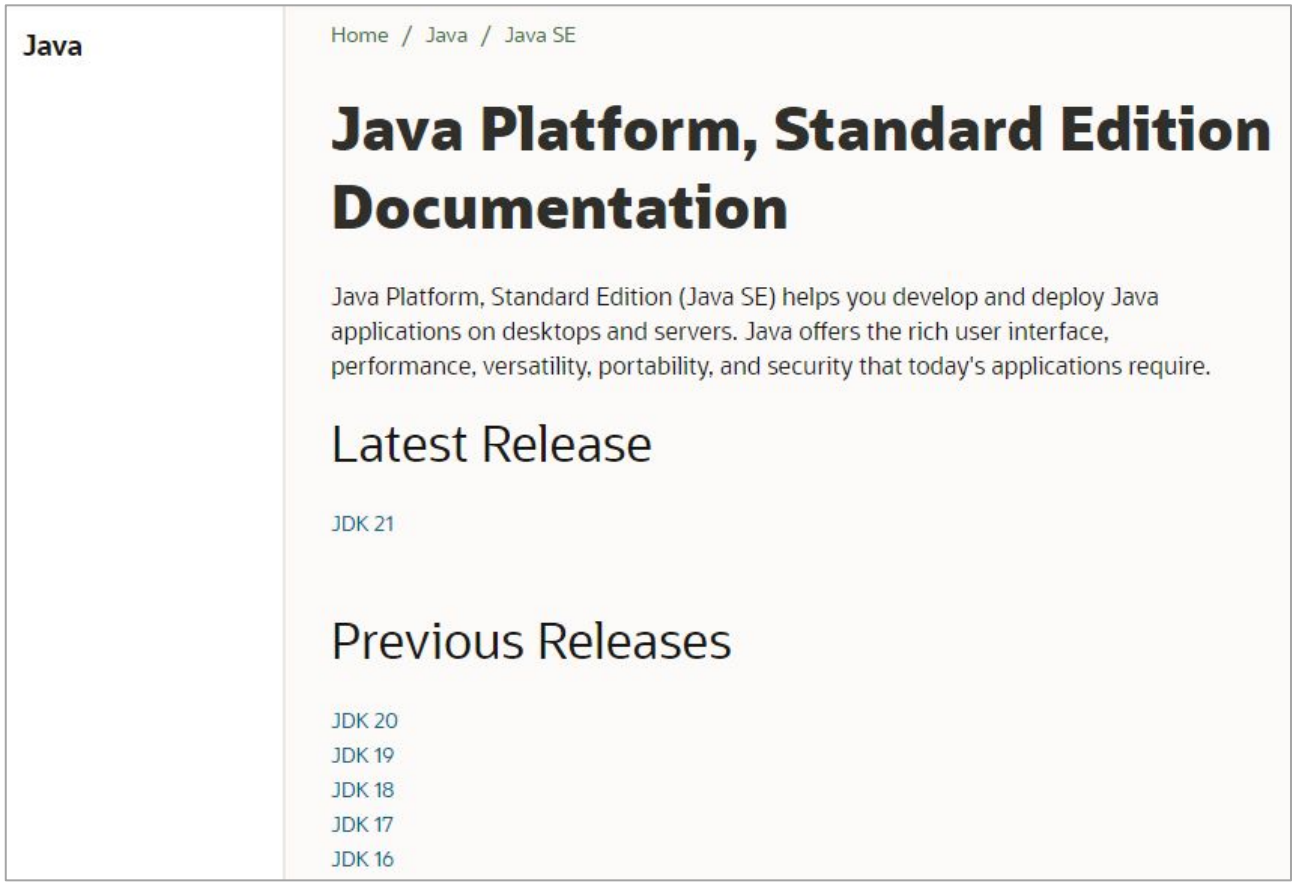
2025년 상반기 K-디지털 트레이닝

# java.base 모듈

[KB] IT’s Your Life

KB 국민은행

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



# 2 java.base 모듈

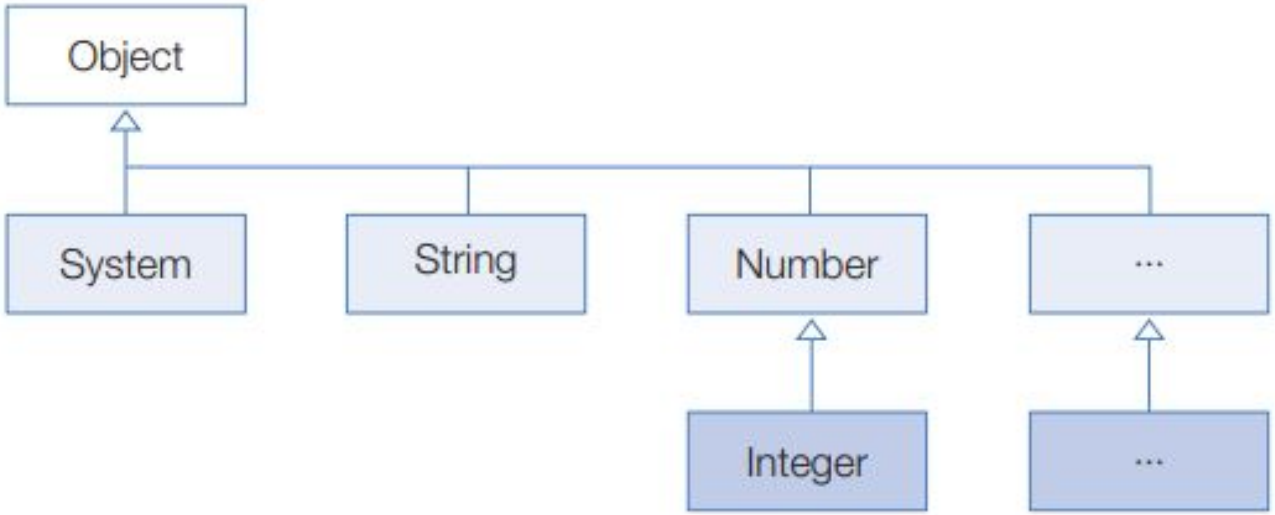
- **java.base**
  - 모든 모듈이 의존하는 기본 모듈로, 모듈 중 유일하게 requires하지 않아도 사용할 수 있음

| 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 |

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

| 클래스 | | 용도 |
|---|---|---|
| Object | | – 자바 클래스의 최상위 클래스로 사용 |
| System | | – 키보드로부터 데이터를 입력받을 때 사용<br>– 모니터(콘솔)로 출력하기 위해 사용<br>– 프로세스를 종료시킬 때 사용<br>– 진행 시간을 읽을 때 사용<br>– 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<br>Integer, Float, Double<br>Boolean | – 기본 타입의 값을 포장할 때 사용<br>– 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)
```

```
Object obj1 = new Object();
Object obj2 = obj1;
boolean result = obj1.equals(obj2);
boolean result = (obj1 == obj2)
```

결과가 동일

- **Member.java**

```
package ch12.sec03.exam01;

public class Member {
	public String id;

	public Member(String id) {
		this.id = id;
	}

	@Override
	public boolean equals(Object obj) {
		if(obj instanceof Member target) {		// 타입 감사
			if(id.equals(target.id)) {			// id 문자열이 같은지 비교
				return true;
			}
		}
		return false;
	}
}
```

- **EqualsExample.java**

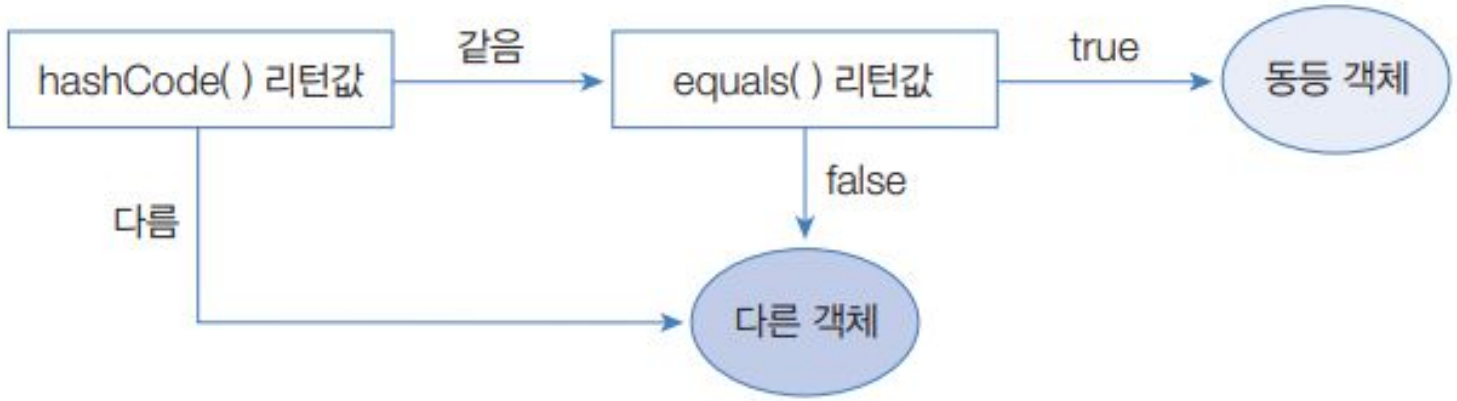
```
package ch12.sec03.exam01;

public class EqualsExample {
	public static void main(String[] args) {
		Member obj1 = new Member("blue");
		Member obj2 = new Member("blue");
		Member obj3 = new Member("red");

		if(obj1.equals(obj2)) {    // 같은 Member 타입이며 id 동일 → true
			System.out.println("obj1과 obj2는 동등합니다.");
		} else {
			System.out.println("obj1과 obj2는 동등하지 않습니다.");
		}

		if(obj1.equals(obj3)) {    // 같은 Member 타입이지만 id가 다름 → true
			System.out.println("obj1과 obj3은 동등합니다.");
		} else {
			System.out.println("obj1과 obj3은 동등하지 않습니다.");
		}
	}
}
```

```
obj1과 obj2는 동등합니다.
obj1과 obj3은 동등하지 않습니다.
```

- **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



- **Student.java**

```
package ch12.sec03.exam02;

public class Student {
	private int no;
	private String name;

	public Student(int no, String name) {
		this.no = no;
		this.name = name;
	}

	public int getNo() { return no; }

	public String getName() { return name; }
```

- **Student.java**

```
    @Override
    public int hashCode() {
        int hashCode = no + name.hashCode();
        return hashCode;
    }

    @Override
    public boolean equals(Object obj) {
        if(obj instanceof Student target) {
            if(no == target.getNo() && name.equals(target.getName())) {
                return true;
            }
        }
        return false;
    }
}
```

- **HashCodeExample.java**

```
package ch12.sec03.exam02;

public class HashCodeExample {

	public static void main(String[] args) {
		Student s1 = new Student(1, "홍길동");
		Student s2 = new Student(1, "홍길동");

		if(s1.hashCode() == s2.hashCode()) {    // 해시코드가 동일한지 검사
			if(s1.equals(s2)) {                                    // 데이터가 동일한지 검사
				System.out.println("동등 객체입니다.");
			} else {
				System.out.println("데이터가 다르므로 동등 객체가 아닙니다.");
			}
		} else {
			System.out.println("해시코드가 다르므로 동등 객체가 아닙니다.");
		}
	}
}
```

```
동등 객체입니다.
```

- **HashSetExample.java**

```
package ch12.sec03.exam02;

import java.util.HashSet;

public class HashSetExample {
	public static void main(String[] args) {
		HashSet hashSet = new HashSet();

		Student s1 = new Student(1, "홍길동");
		hashSet.add(s1);
		System.out.println("저장된 객체 수: " + hashSet.size());

		Student s2 = new Student(1, "홍길동");
		hashSet.add(s2);
		System.out.println("저장된 객체 수: " + hashSet.size());

		Student s3 = new Student(2, "홍길동");
		hashSet.add(s3);
		System.out.println("저장된 객체 수: " + hashSet.size());
	}
}
```

동등 객체는
중복 저장되지 않음

```
저장된 객체 수: 1
저장된 객체 수: 1
저장된 객체 수: 2
```

- **hashCode() 없는 경우**
  - 디폴트 hashCode()는 객체 인스턴스의 메모리 주소를 생성 → 내용이 같아도 다른 객체로 인식

```
/*
@Override
public int hashCode() {
  int hashCode = no + name.hashCode();
  return hashCode;
}
*/
```

```
저장된 객체 수: 1
저장된 객체 수: 2
저장된 객체 수: 2
```

- **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
```

- **SmartPhone.java**

```
package ch12.sec03.exam03;

public class SmartPhone {
	private String company;
	private String os;

	public SmartPhone(String company, String os) {
		this.company = company;
		this.os = os;
	}

	@Override
	public String toString() {
		return company + ", " + os;
	}
}
```

- **ToStringExample.java**

```
package ch12.sec03.exam03;

public class ToStringExample {
	public static void main(String[] args) {
		SmartPhone myPhone = new SmartPhone("삼성전자", "안드로이드");

		String strObj = myPhone.toString();
		System.out.println(strObj);

		System.out.println(myPhone);
	}
}
```

```
삼성전자, 안드로이드
삼성전자, 안드로이드
```

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

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

  - 변수의 타입과 이름을 이용해서 private final 필드가 자동 생성
  - 생성자 및 Getter 메소드가 자동 추가
  - hashCode(), equals(), toString() 메서드 자동 추가

- **Member.java**

```
package ch12.sec03.exam04;

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

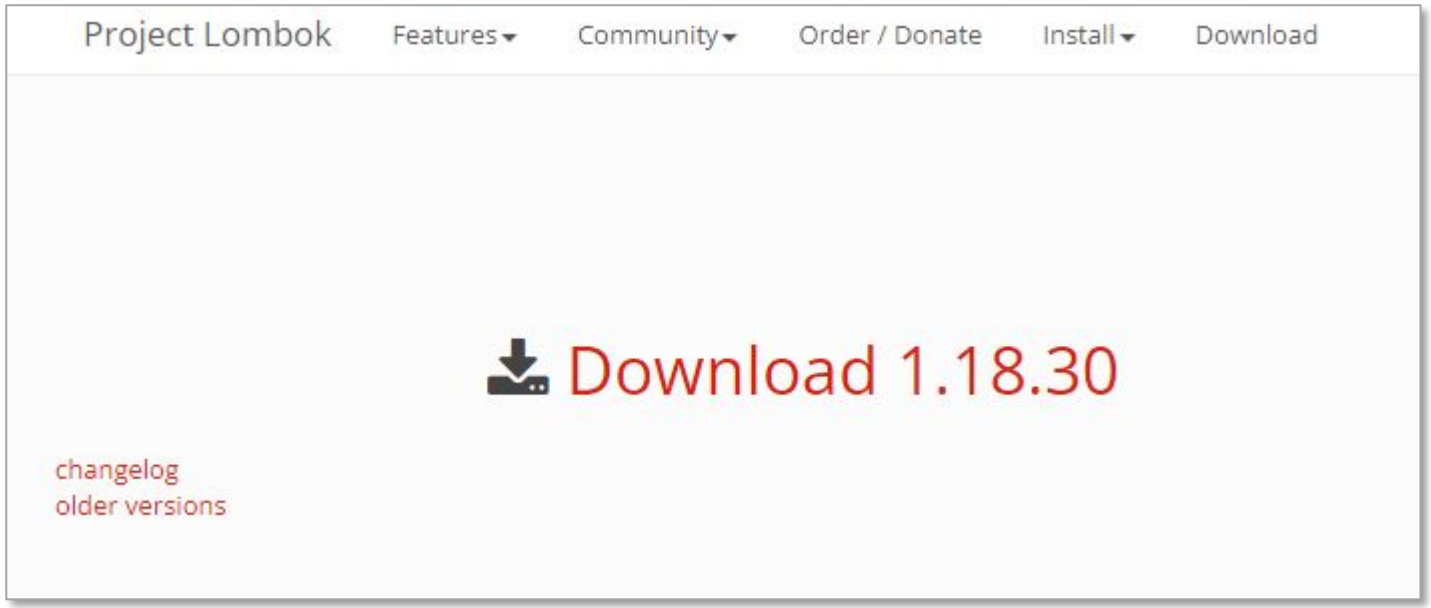
- **RecordExample.java**

```
package ch12.sec03.exam04;

public class RecordExample {
	public static void main(String[] args) {
		Member m = new Member("winter", "눈송이", 25);
		System.out.println(m.id());
		System.out.println(m.name());
		System.out.println(m.age());
		System.out.println(m.toString());
		System.out.println();

		Member m1 = new Member("winter", "눈송이", 25);
		Member m2 = new Member("winter", "눈송이", 25);
		System.out.println("m1.hashCode(): " + m1.hashCode());
		System.out.println("m2.hashCode(): " + m2.hashCode());
		System.out.println("m1.equals(m2): " + m1.equals(m2) );
	}
}
```

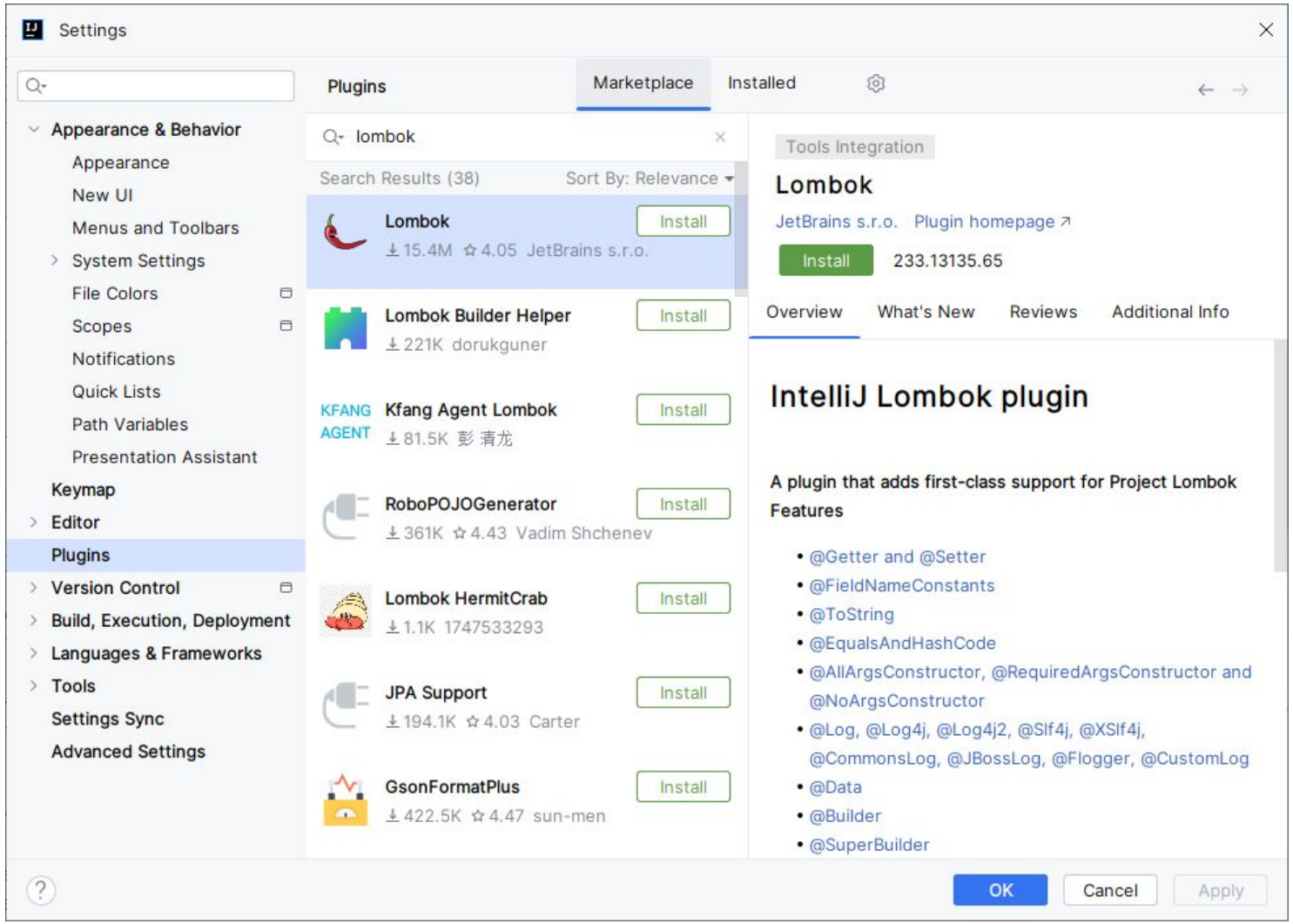
```
winter
눈송이
25
Member[id=winter, name=눈송이, age=25]

m1.hashCode(): 306065155
m2.hashCode(): 306065155
m1.equals(m2): true
```

- 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

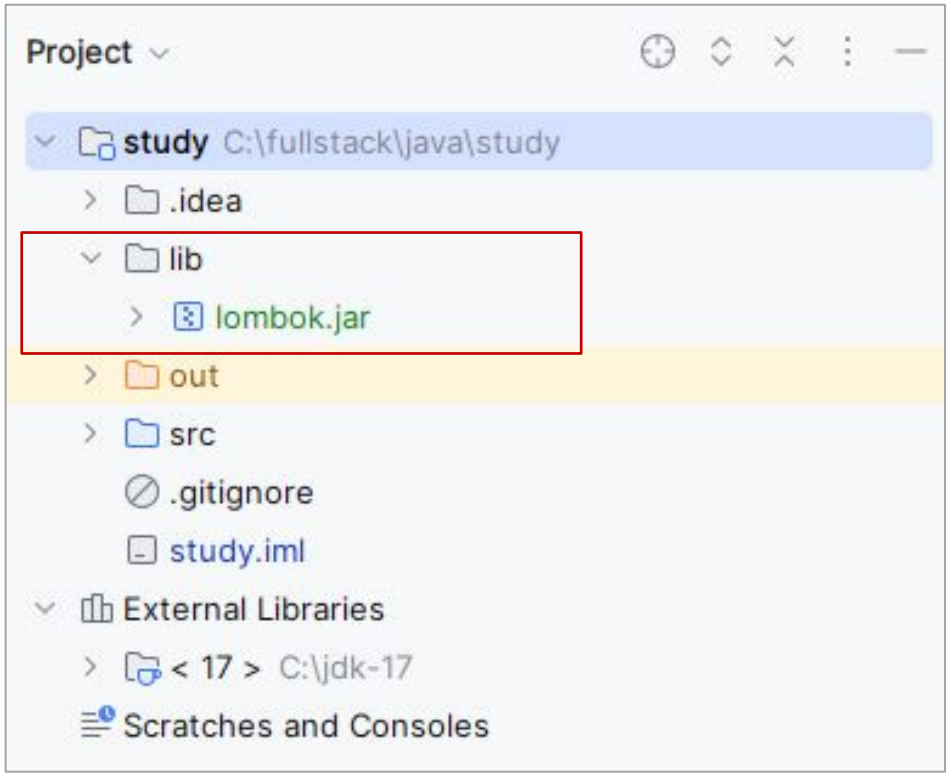
    - https://projectlombok.org/download



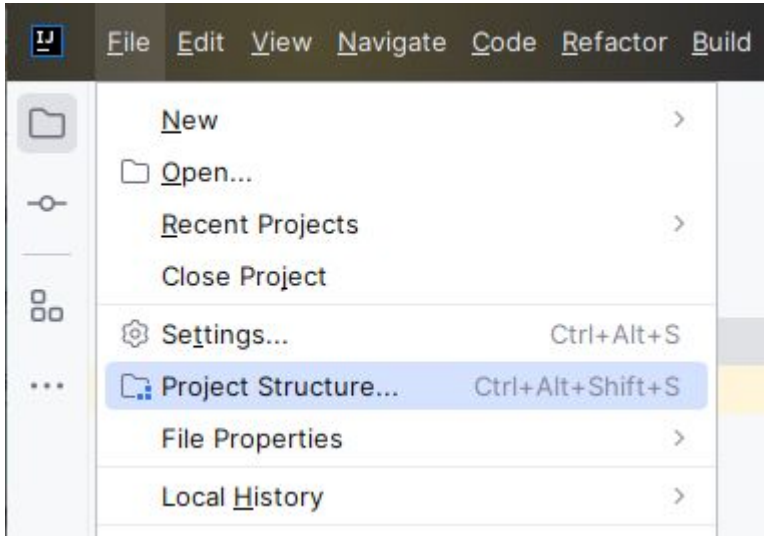
- 롬복 플러그 인 설치
  - File → Settings → Plugins → Marketplace
  - lombok 검색

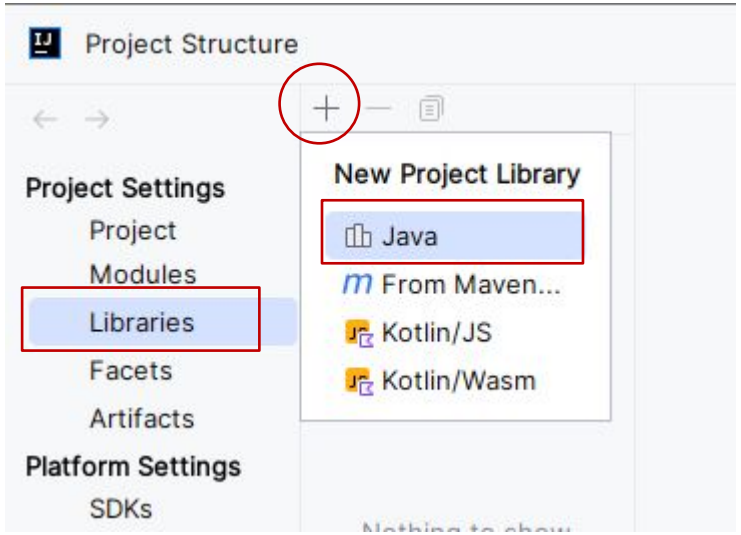


- 롬복 라이브러리 사용하기
  - lib 디렉토리 생성 후 Lombok.jar 추가

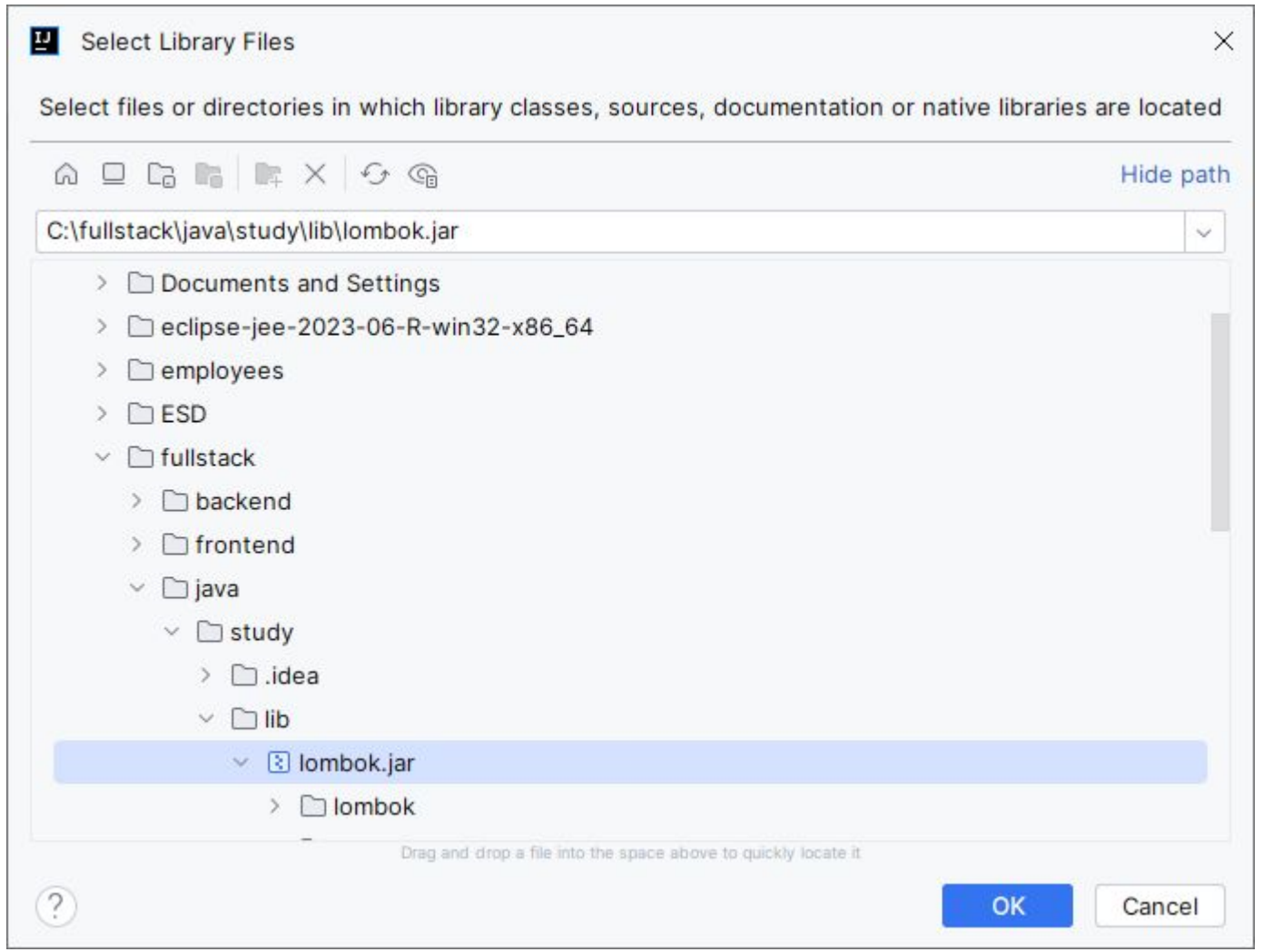


- 롬복 라이브러리 사용하기
  - 프로젝트에 Lombok.jar을 라이브러리로 등록
    - File > Project Structure

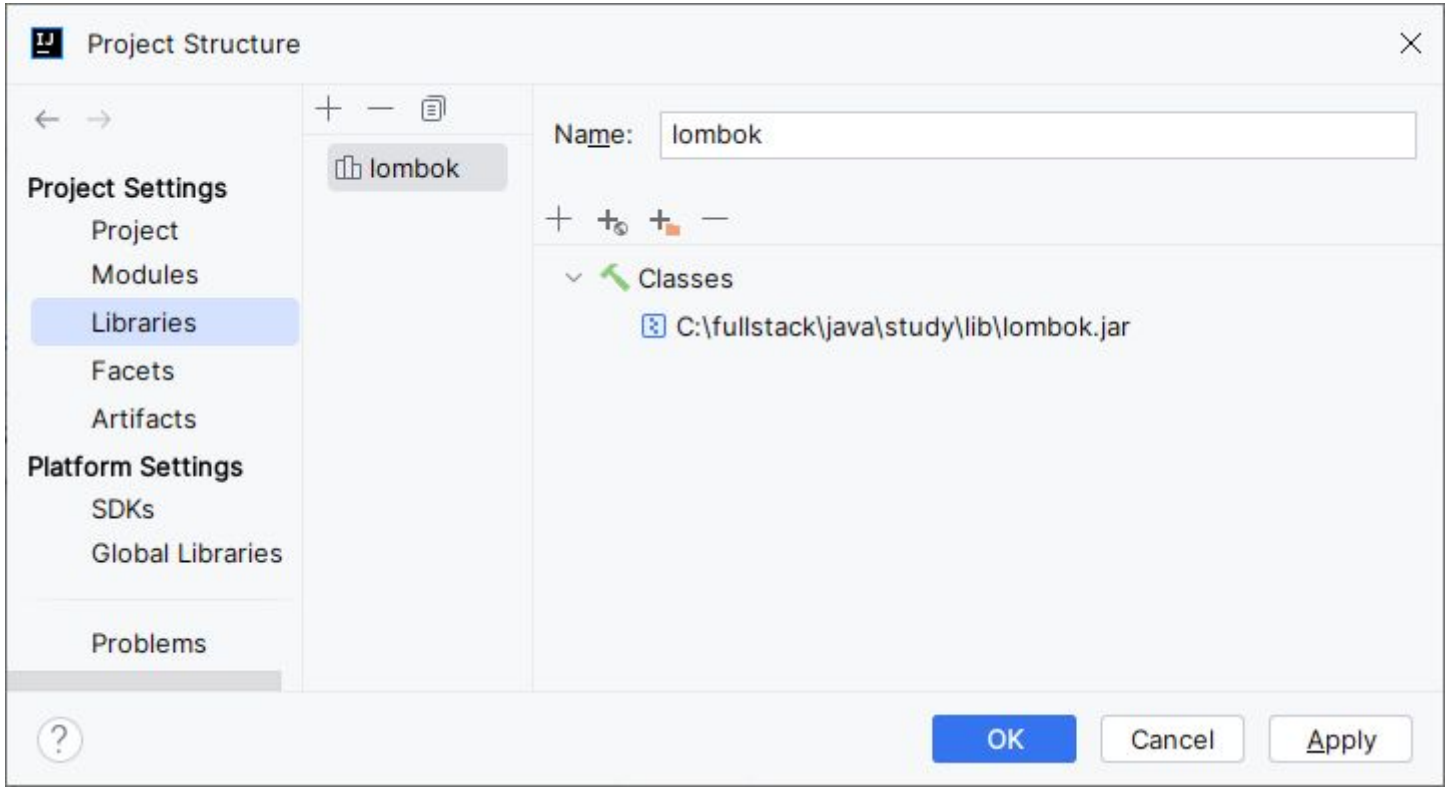


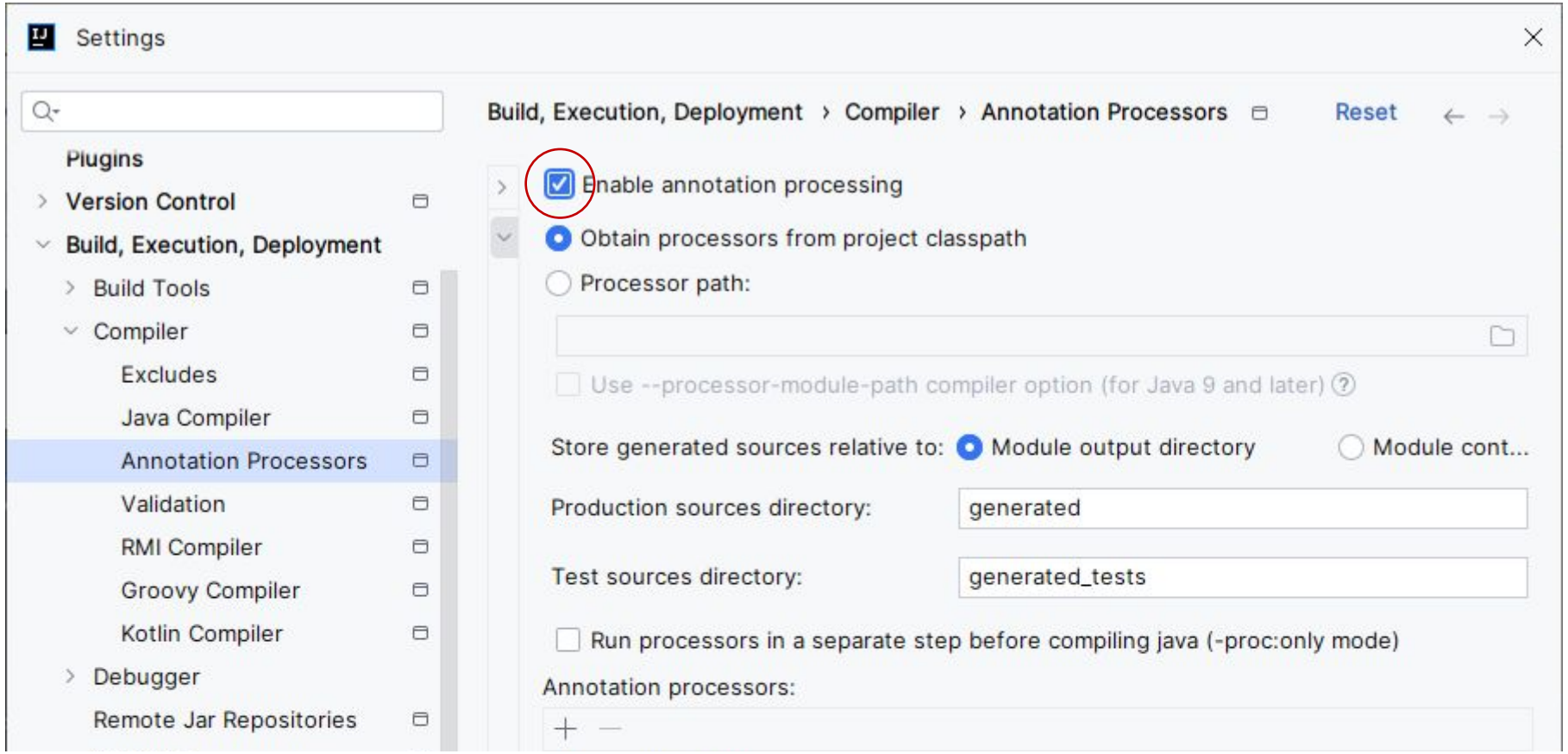
- 롬복 라이브러리 사용하기
  - 프로젝트에 Lombok.jar을 라이브러리로 등록



- 롬복 라이브러리 사용하기
  - 프로젝트에 Lombok.jar을 라이브러리로 등록



- 롬복 라이브러리 사용하기
  - Annotation Processors 설정
  - File > Settings > Build, Exception, Deplyment > Compiler > Annotation Processors



- 롬복 라이브러리 사용하기
  - 롬복 어노테이션

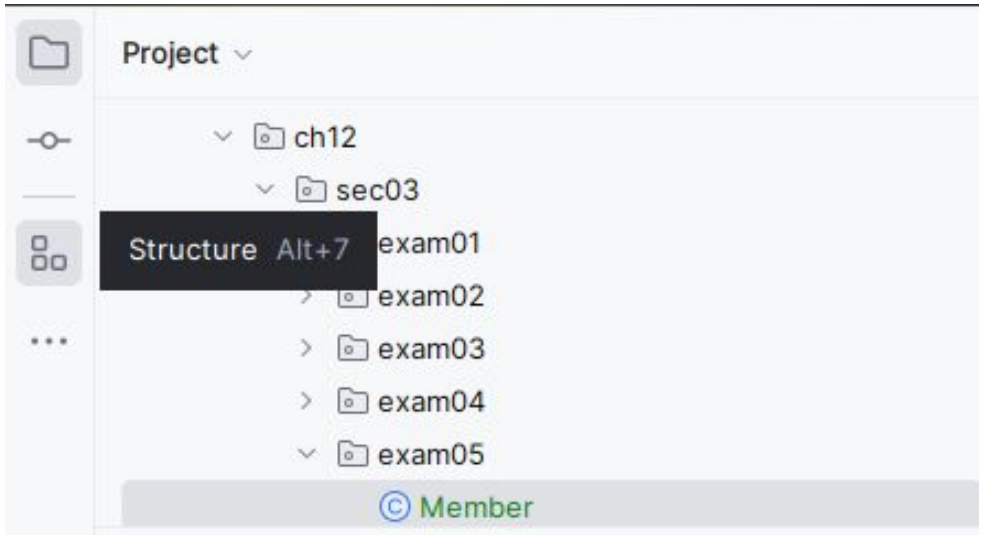
| 어노테이션 | 설명 |
|---|---|
| @NoArgsConstructor | 기본(매개변수가 없는) 생성자 포함 |
| @AllArgsConstructor | 모든 필드를 초기화시키는 생성자 포함 |
| @RequiredArgsConstructor | 기본적으로 매개변수가 없는 생성자 포함. 만약 final 또는 @NonNull이 붙은 필드가 있다면 이 필드만 초기화시키는 생성자 포함 |
| @Getter | Getter 메소드 포함 |
| @Setter | Setter 메소드 포함 |
| @EqualsAndHashCode | equals( )와 hashCode( ) 메소드 포함 |
| @ToString | toString( ) 메소드 포함 |

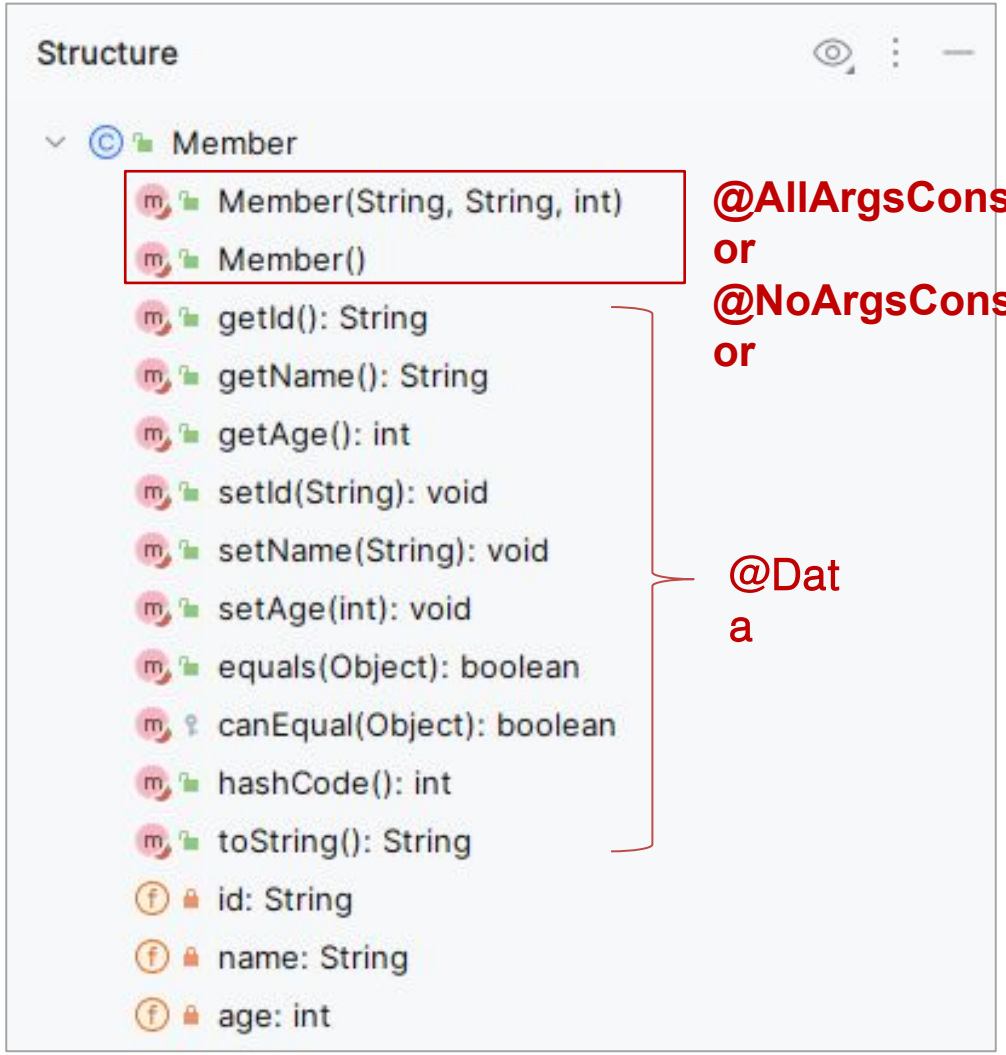
  - @Data
    - @RequiredArgsConstructor, @Getter, @Setter, @EqualsAndHashCode, @ToString 합친 것

- **Member.java**

```
package ch12.sec03.exam05;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Member {
	private String id;
	private String name;
	private int age;
}
```

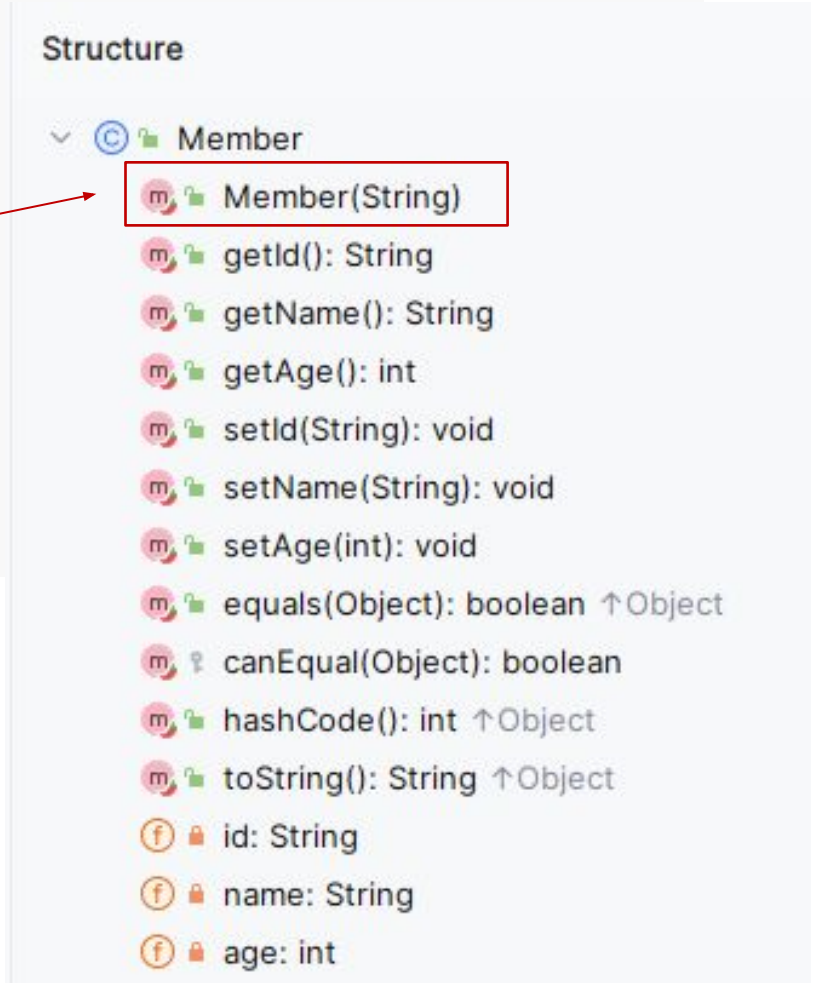
- 생성된 코드 확인
  - Structure 아이콘 클릭 또는 Alt + 7



- **Member.java**

```
package ch12.sec03.exam05;

import lombok.Data;
import lombok.NonNull;

@Data
public class Member {
        private String id;
        @NonNull // 필수 항목 → @RequiredArgsConstructor에 의해 생성자 추가됨
  private String name;
        private int age;
}
```



- **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 |

- **ErrExample.java** **콘솔 출력**

```
package ch12.sec04;

public class ErrExample {

	public static void main(String[] args) {
		try {
			int value = Integer.parseInt("1oo");
		} catch(NumberFormatException e) {
			System.err.println("[에러 내용]");
			System.err.println(e.getMessage());
		}
	}
}
```

```
[에러 내용]
For input string: "1oo"
```

- 키보드 입력
  - System.in 객체 이용

```
int keyCode = System.in.read();
```

  - 키 코드

| | | | | | |
|---|---|---|---|---|---|
| 0 = 48 | A = 65 | N = 78 | a = 97 | n = 110 | [Windows] |
| 1 = 49 | B = 66 | O = 79 | b = 98 | o = 111 | Enter = 13, 10 |
| 2 = 50 | C = 67 | P = 80 | c = 99 | p = 112 | |
| 3 = 51 | D = 68 | Q = 81 | d = 100 | q = 113 | [macOS] |
| 4 = 52 | E = 69 | R = 82 | e = 101 | r = 114 | Enter = 10 |
| 5 = 53 | F = 70 | S = 83 | f = 102 | s = 115 | |
| 6 = 54 | G = 71 | T = 84 | g = 103 | t = 116 | |
| 7 = 55 | H = 72 | U = 85 | h = 104 | u = 117 | |
| 8 = 56 | I = 73 | V = 86 | i = 105 | v = 118 | |
| 9 = 57 | J = 74 | W = 87 | j = 106 | w = 119 | |
| | K = 75 | X = 88 | k = 107 | x = 120 | |
| | L = 76 | Y = 89 | l = 108 | y = 121 | |
| | M = 77 | Z = 90 | m = 109 | z = 122 | |

- **ErrExample.java　　키보드 입력**

```
package ch12.sec04;

public class InExample {
	public static void main(String[] args) throws Exception {
		int speed = 0;
		int keyCode = 0;

		while(true) {
			//Enter 키를 읽지 않았을 경우에만 실행
			if(keyCode != 13 && keyCode != 10) {
				if (keyCode == 49) {                    //숫자 1 키를 읽었을 경우
					speed++;
				} else if (keyCode == 50) {       //숫자 2 키를 읽었을 경우
					speed--;
				} else if (keyCode == 51) {       //숫자 3 키를 읽었을 경우
					break;
				}
				System.out.println("-----------------------------");
				System.out.println("1. 증속 | 2. 감속 | 3. 중지");
				System.out.println("-----------------------------");
				System.out.println("현재 속도= " + speed);
				System.out.print("선택: ");
			}
```

- **ErrExample.java** 키보드 입력

```
            //키를 하나씩 읽음
            keyCode = System.in.read();
        }   // end of while

        System.out.println("프로그램 종료");
    }
}
```

```
------------------------------
1. 증속 | 2. 감속 | 3. 중지
------------------------------
현재 속도= 0
선택: 1
------------------------------
1. 증속 | 2. 감속 | 3. 중지
------------------------------
현재 속도= 1
선택: 2
------------------------------
1. 증속 | 2. 감속 | 3. 중지
------------------------------
현재 속도= 0
선택: 3
프로그램 종료
```

- **프로세스 종료**
  - 프로세스 process
    - 운영체제는 실행 중인 프로그램을 프로세스로 관리
    - 자바 프로그램을 시작하면 JVM 프로세스가 생성
      - 이 프로세스가 main() 메서드 호출
  - System.exit() 메서드로 현재 프로세스를 강제 종료 가능
    - System.exit(int status)
    - 매개변수로 종료 상태 코드 지정
      - 0: 정상 종료(관례)
      - 그 외 에러 상태를 나타내는 정수 지정

- **ExitExample.java**

```
package ch12.sec04;

public class ExitExample {
	public static void main(String[] args) {
		for(int i=0; i<10; i++) {
			//i값 출력
			System.out.println(i);
			if(i == 5) {
				//JVM 프로세스 종료
				System.out.println("프로세스 강제 종료");
				System.exit(0);
			}
		}
	}
}
```

```
0
1
2
3
4
5
프로세스 강제 종료
```

- 진행 시간 읽기

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

- **MeasureRunTimeExample.java**

```
package ch12.sec04;

public class MeasureRunTimeExample {
	public static void main(String[] args) {
		long time1 = System.nanoTime();
		int sum = 0;
		for(int i=1; i<=1000000; i++) {
			sum += i;
		}
		long time2 = System.nanoTime();

		System.out.println("1~1000000까지의 합: " + sum);
		System.out.println("계산에 " + (time2-time1) + " 나노초가 소요되었습니다.");
	}
}
```

```
1~1000000까지의 합: 1784293664
계산에 4161100 나노초가 소요되었습니다.
```

- 시스템 프로퍼티 읽기
  - 시스템 프로퍼티 System Property
    - 자바 프로그램이 시작될 때 자동 설정되는 시스템의 속성
      - 키(key), 값(value)의 쌍으로 구성
    - 주요 속성

| 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 |

- **GetPropertyExample.java**

```
package ch12.sec04;

import java.util.Properties;
import java.util.Set;

public class GetPropertyExample {
	public static void main(String[] args) {
		//운영체제와 사용자 정보 출력
		String osName = System.getProperty("os.name");
		String userName = System.getProperty("user.name");
		String userHome = System.getProperty("user.home");

		System.out.println(osName);
		System.out.println(userName);
		System.out.println(userHome);
```

```
Windows 10
a
C:₩Users₩a
```

- **GetPropertyExample.java**

```
        //전체 키와 값을 출력
        System.out.println("------------------------------");
        System.out.println(" key: value");
        System.out.println("------------------------------");

        Properties props = System.getProperties();
        Set keys = props.keySet();

        for(Object objKey : keys) {
            String key = (String) objKey;
            String value = System.getProperty(key);
            System.out.printf("%-40s: %s\n", key, value
        }
    }
}
```

```
------------------------------
 key: value
------------------------------
java.specification.version               : 17
sun.cpu.isalist                          : amd64
sun.jnu.encoding                         : MS949
java.class.path                          :
C:\fullstack\java\study\out\production\study;C:\fullstack\java\study
\lib\lombok.jar
…
```

- 문자열 관련 주요 클래스

| 클래스 | 설명 |
| --- | --- |
| String | 문자열을 저장하고 조작할 때 사용 |
| StringBuilder | 효율적인 문자열 조작 기능이 필요할 때 사용 |
| StringTokenizer | 구분자로 연결된 문자열을 분리할 때 사용 |

- **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

- **BytesToStringExample.java**

```
package ch12.sec05;

import java.util.Arrays;

public class BytesToStringExample {
	public static void main(String[] args) throws Exception  {
		String data = "자바";

		//String -> byte 배열(기본: UTF-8 인코딩)
		byte[] arr1 = data.getBytes();
		//byte[] arr1 = data.getBytes("UTF-8");
		System.out.println("arr1: " + Arrays.toString(arr1));

		//byte 배열 -> String(기본: UTF-8 디코딩)
		String str1 = new String(arr1);
		//String str1 = new String(arr1, "UTF-8");
		System.out.println("str1: " + str1);
```

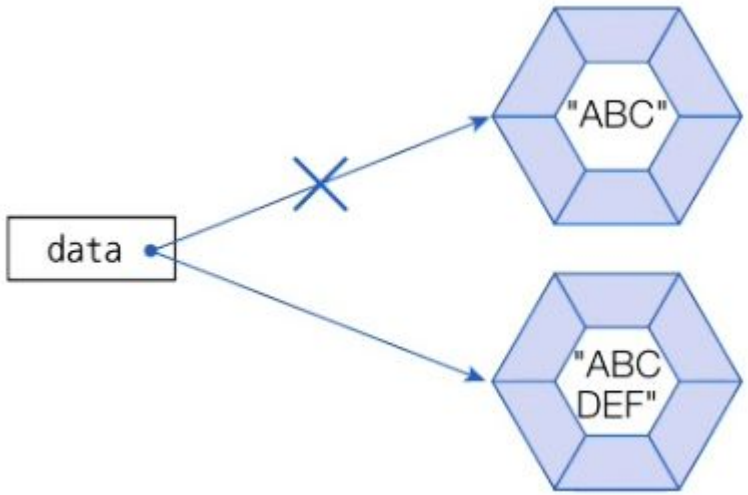
## BytesToStringExample.java

```
        //String -> byte 배열(EUC-KR 인코딩)
        byte[] arr2 = data.getBytes("EUC-KR");
        System.out.println("arr2: " + Arrays.toString(arr2));

        //byte 배열 -> String(기본: UTF-8 디코딩)
        String str2 = new String(arr2, "EUC-KR");
        System.out.println("str2: " + str2);
    }
}
```

```
arr1: [-20, -98, -112, -21, -80, -108]
str1: 자바
arr2: [-64, -38, -71, -39]
str2: 자바
```

- **StringBuilder 클래스**
  - 잦은 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 |

- **StringBuilderExample.java**

```
package ch12.sec05;

public class StringBuilderExample {
	public static void main(String[] args) {
		String data = new StringBuilder()
				.append("DEF")
				.insert(0, "ABC")
				.delete(3, 4)
				.toString();
		System.out.println(data);
	}
```

메소드 체이닝 패턴

```
ABCEF
```

- **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, "/");
```

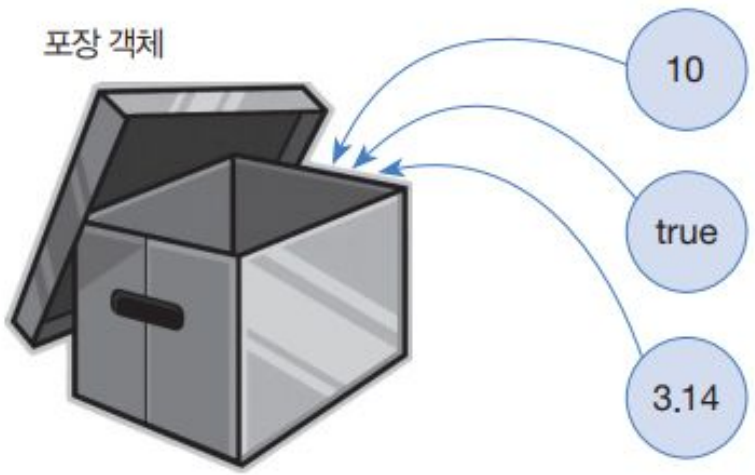
| 리턴 타입 | 메소드(매개변수) | 설명 |
|---|---|---|
| int | countTokens( ) | 분리할 수 있는 문자열의 총 수 |
| boolean | hasMoreTokens( ) | 남아 있는 문자열이 있는지 여부 |
| String | nextToken( ) | 문자열을 하나씩 가져옴 |

- **StringTokenizerExample.java**

```
package ch12.sec05;

import java.util.StringTokenizer;

public class StringTokenizerExample {
	public static void main(String[] args) {
		String data1 = "홍길동&이수홍,박연수";
		String[] arr = data1.split("&|,");

		for(String token : arr) {
			System.out.println(token);
		}
		System.out.println();

		String data2 = "홍길동/이수홍/박연수";
		StringTokenizer st = new StringTokenizer(data2, "/");
		while (st.hasMoreTokens()) {
			String token = st.nextToken();
			System.out.println(token);
		}
	}
}
```

```
홍길동
이수홍
박연수

홍길동
이수홍
박연수
```

- **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
```

- **BoxingUnBoxingExample.java**

```
package ch12.sec06;

public class BoxingUnBoxingExample {
	public static void main(String[] args) {
		//Boxing
		Integer obj = 100;
		System.out.println("value: " + obj.intValue());

		//Unboxing
		int value = obj;
		System.out.println("value: " + value);

		//연산 시 Unboxing
		int result = obj + 100;
		System.out.println("result: " + result);
	}
}
```

```
value: 100
value: 100
result: 200
```

- **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

- **BoxingUnBoxingExample.java**

```
package ch12.sec06;

public class ValueCompareExample {
	public static void main(String[] args) {
		//-128~127 초과값일 경우
		Integer obj1 = 300;
		Integer obj2 = 300;
		System.out.println("==: " + (obj1 == obj2));
		System.out.println("equals(): " + obj1.equals(obj2));
		System.out.println();

		//-128~127 범위값일 경우
		Integer obj3 = 10;
		Integer obj4 = 10;
		System.out.println("==: " + (obj3 == obj4));
		System.out.println("equals: " + obj3.equals(obj4));
	}
}
```

```
==: false
equals(): true

==: true
equals: true
```

- **Math 클래스**
  - 수학 계산에 사용할 수 있는 정적 메소드 제공

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

- **MathExample.java**

```
package ch12.sec07;

public class MathExample {
	public static void main(String[] args) {
		//큰 정수 또는 작은 정수 얻기
		double v1 = Math.ceil(5.3);
		double v2 = Math.floor(5.3);
		System.out.println("v1=" + v1);
		System.out.println("v2=" + v2);

		//큰값 또는 작은값 얻기
		long v3 = Math.max(3, 7);
		long v4 = Math.min(3, 7);
		System.out.println("v3=" + v3);
		System.out.println("v4=" + v4);

		//소수 이하 두 자리 얻기
		double value = 12.3456;
		double temp1 = value * 100;
		long temp2 = Math.round(temp1);
		double v5 = temp2 / 100.0;
		System.out.println("v5=" + v5);
	}
}
```

```
v1=6.0
v2=5.0
v3=7
v4=3
v5=12.35
```

- 랜덤 숫자 얻기
  - Math.random()
    - 0.0과 1.0 사이의 double 타입 난수를 리턴
  - 생성자

| 객체 생성 | 설명 |
|---|---|
| Random( ) | 현재 시간을 이용해서 종자값을 자동 설정한다. |
| Random(long seed) | 주어진 종자값을 사용한다. |

  - 메소드

| 리턴값 | 메소드(매개변수) | 설명 |
|---|---|---|
| boolean | nextBoolean( ) | boolean 타입의 난수를 리턴 |
| double | nextDouble( ) | double 타입의 난수를 리턴(0.0<= ~ < 1.0) |
| int | nextInt( ) | int 타입의 난수를 리턴($-2^{32}$<= ~ <=$2^{32}-1$); |
| int | nextInt(int n) | int 타입의 난수를 리턴(0<= ~ <n) |

- **RandomExample.java**

```
package ch12.sec07;

import java.util.Arrays;
import java.util.Random;

public class RandomExample {
	public static void main(String[] args) {
		//선택번호
		int[] selectNumber = new int[6];
		Random random = new Random(3);
		System.out.print("선택번호: ");
		for(int i=0; i<6; i++) {
			selectNumber[i] = random.nextInt(45) + 1;
			System.out.print(selectNumber[i] + " ");
		}
		System.out.println();
```

```
선택번호: 15 21 16 17 34 28
```

- **RandomExample.java**

```
        //당첨번호
        int[] winningNumber = new int[6];
        random = new Random(5);
        System.out.print("당첨번호: ");
        for(int i=0; i<6; i++) {
            winningNumber[i] = random.nextInt(45) + 1;
            System.out.print(winningNumber[i] + " ");
        }
        System.out.println();

        //당첨여부, 비교하기 전에 배열 항목을 정렬
        Arrays.sort(selectNumber);
        Arrays.sort(winningNumber);
        boolean result = Arrays.equals(selectNumber, winningNumber);        // 배열 항목 비교
        System.out.print("당첨여부: ");
        if(result) {
            System.out.println("1등에 당첨되셨습니다.");
        } else {
            System.out.println("당첨되지 않았습니다.");
        }
    }
}
```

```
당첨번호: 18 38 45 15 22 36
당첨여부: 당첨되지 않았습니다.
```

- **java.util 패키지**

| 클래스 | 설명 |
|---|---|
| Date | 날짜 정보를 전달하기 위해 사용 |
| Calendar | 다양한 시간대별로 날짜와 시간을 얻을 때 사용 |
| LocalDateTime | 날짜와 시간을 조작할 때 사용 |

- **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();
```

- **DateExample.java**

```
package ch12.sec08;

import java.text.*;
import java.util.*;

public class DateExample {
	public static void main(String[] args) {
		Date now = new Date();
		String strNow1 = now.toString();
		System.out.println(strNow1);

		SimpleDateFormat sdf = new SimpleDateFormat("yyyy.MM.dd HH:mm:ss");
		String strNow2 = sdf.format(now);
		System.out.println(strNow2);
	}
}
```

```
Thu Jan 18 12:22:20 KST 2024
2024.01.18 12:22:20
```

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

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

```
int year   = now.get(Calendar.YEAR);          //년도를 리턴
int month  = now.get(Calendar.MONTH) + 1;     //월을 리턴
int day    = now.get(Calendar.DAY_OF_MONTH);  //일을 리턴
int week   = now.get(Calendar.DAY_OF_WEEK);   //요일을 리턴
int amPm   = now.get(Calendar.AM_PM);         //오전/오후를 리턴
int hour   = now.get(Calendar.HOUR);          //시를 리턴
int minute = now.get(Calendar.MINUTE);        //분을 리턴
int second = now.get(Calendar.SECOND);        //초를 리턴
```

- **CalendarExample.java**

```
package ch12.sec08;

import java.util.*;

public class CalendarExample {
	public static void main(String[] args) {
		Calendar now = Calendar.getInstance();

		int year  = now.get(Calendar.YEAR);
		int month  = now.get(Calendar.MONTH) + 1;
		int day  = now.get(Calendar.DAY_OF_MONTH);

		int week  = now.get(Calendar.DAY_OF_WEEK);
		String strWeek = null;
		switch(week) {
			case Calendar.MONDAY:  strWeek = "월";      break;
			case Calendar.TUESDAY:  strWeek = "화"; break;
			case Calendar.WEDNESDAY: strWeek = "수"; break;
			case Calendar.THURSDAY:      strWeek = "목"; break;
			case Calendar.FRIDAY:    strWeek = "금"; break;
			case Calendar.SATURDAY:      strWeek = "토"; break;
			default:                     strWeek = "일";
		}
```

- **CalendarExample.java**

```
        int amPm  = now.get(Calendar.AM_PM);
        String strAmPm = null;
        if(amPm == Calendar.AM) {
            strAmPm = "오전";
        } else {
            strAmPm = "오후";
        }

        int hour    = now.get(Calendar.HOUR);
        int minute  = now.get(Calendar.MINUTE);
        int second  = now.get(Calendar.SECOND);

        System.out.print(year + "년 ");
        System.out.print(month + "월 ");
        System.out.println(day + "일 ");
        System.out.print(strWeek + "요일 ");
        System.out.println(strAmPm + " ");
        System.out.print(hour + "시 ");
        System.out.print(minute + "분 ");
        System.out.println(second + "초 ");
    }
}
```

```
2024년 1월 18일
목요일 오후
0시 25분 0초
```

- **TimeZone**
  - 타임존 지정하기

```
TimeZone timeZone = TimeZone.getTimeZone("America/Los_Angeles");
Calendar now = Calendar.getInstance( timeZone );
```

- **LosAngelesExample.java**

```
package ch12.sec08;

import java.util.Calendar;
import java.util.TimeZone;

public class LosAngelesExample {
	public static void main(String[] args) {
		TimeZone timeZone = TimeZone.getTimeZone("America/Los_Angeles");
		Calendar now = Calendar.getInstance( timeZone );

		int amPm = now.get(Calendar.AM_PM);
		String strAmPm = null;
		if(amPm == Calendar.AM) {
			strAmPm = "오전";
		} else {
			strAmPm = "오후";
		}
		int hour = now.get(Calendar.HOUR);
		int minute = now.get(Calendar.MINUTE);
		int second = now.get(Calendar.SECOND);
```

- **LosAngelesExample.java**

```
            System.out.print(strAmPm + " ");
            System.out.print(hour + "시 ");
            System.out.print(minute + "분 ");
            System.out.println(second + "초 ");
        }
}
```

```
오후 7시 26분 11초
```

- **PrintTimeZoneID.java**

```
package ch12.sec08;

import java.util.TimeZone;

public class PrintTimeZoneID {
	public static void main(String[] args) {
		String[] availableIDs = TimeZone.getAvailableIDs();
		for(String id : availableIDs) {
			System.out.println(id);
		}
	}
}
```

```
Africa/Abidjan
Africa/Accra
Africa/Addis_Ababa
Africa/Algiers
:
```

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

```
LocalDateTime now = LocalDateTime.now();
```

| 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 |

- **DateTimeOperationExample.java**

```
package ch12.sec08;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class DateTimeOperationExample {
	public static void main(String[] args) {
		LocalDateTime now = LocalDateTime.now();
		DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy.MM.dd a HH:mm:ss");
		System.out.println("현재 시간: " + now.format(dtf));

		LocalDateTime result1 = now.plusYears(1);
		System.out.println("1년 덧셈: " + result1.format(dtf));

		LocalDateTime result2 = now.minusMonths(2);
		System.out.println("2월 뺄셈: " + result2.format(dtf));

		LocalDateTime result3 = now.plusDays(7);
		System.out.println("7일 덧셈: " + result3.format(dtf));
	}
}
```

```
현재 시간: 2024.01.18 오후 12:28:40
1년 덧셈: 2025.01.18 오후 12:28:40
2월 뺄셈: 2023.11.18 오후 12:28:40
7일 덧셈: 2024.01.25 오후 12:28:40
```

- **날짜와 시간 비교**
  - LocalDataTime 클래스는 날짜와 시간을 비교할 수 있는 메소드 제공

| 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 |

  - 특정 날짜, 시간 만들기

```
LocalDateTime target = LocalDateTime.of(year, month, dayOfMonth, hour, minute,
second);
```

- **DateTimeCompareExample.java**

```
package ch12.sec08;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.time.temporal.ChronoUnit;

public class DateTimeCompareExample {
	public static void main(String[] args) {
		DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy.MM.dd a HH:mm:ss");

		LocalDateTime startDateTime = LocalDateTime.of(2021, 1, 1, 0, 0, 0);
		System.out.println("시작일: " + startDateTime.format(dtf));

		LocalDateTime endDateTime = LocalDateTime.of(2021, 12, 31, 0, 0, 0);
		System.out.println("종료일: " + endDateTime.format(dtf));

		if(startDateTime.isBefore(endDateTime)) {
			System.out.println("진행 중입니다.");
		} else if(startDateTime.isEqual(endDateTime)) {
			System.out.println("종료합니다.");
		} else if(startDateTime.isAfter(endDateTime)) {
			System.out.println("종료했습니다.");
		}
```

```
시작일: 2021.01.01 오전 00:00:00
종료일: 2021.12.31 오전 00:00:00
진행 중입니다.
```

- **DateTimeCompareExample.java**

```
            long remainYear = startDateTime.until(endDateTime, ChronoUnit.YEARS);
            long remainMonth = startDateTime.until(endDateTime, ChronoUnit.MONTHS);
            long remainDay = startDateTime.until(endDateTime, ChronoUnit.DAYS);
            long remainHour = startDateTime.until(endDateTime, ChronoUnit.HOURS);
            long remainMinute = startDateTime.until(endDateTime, ChronoUnit.MINUTES);
            long remainSecond = startDateTime.until(endDateTime, ChronoUnit.SECONDS);

            System.out.println("남은 해: " + remainYear);
            System.out.println("남은 월: " + remainMonth);
            System.out.println("남은 일: " + remainDay);
            System.out.println("남은 시간: " + remainHour);
            System.out.println("남은 분: " + remainMinute);
            System.out.println("남은 초: " + remainSecond);
        }
}
```

```
남은 해: 0
남은 월: 11
남은 일: 364
남은 시간: 8736
남은 분: 524160
남은 초: 31449600
```

- **DecimalFormat**
  - 숫자를 형식화된 문자열로 변환

| 기호 | 의미 | 패턴 예 | 1234567.89 → 변환 결과 |
|---|---|---|---|
| 0 | 10진수(빈자리는 0으로 채움) | 0<br>0.0<br>0000000000.00000 | 1234568<br>1234567.9<br>0001234567.89000 |
| # | 10진수(빈자리는 채우지 않음) | #<br>#.#<br>##########.##### | 1234568<br>1234567.9<br>1234567.89 |
| . | 소수점 | #.0 | 1234567.9 |
| – | 음수 기호 | +#.0<br>–#.0 | +1234567.9<br>–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)<br>–1,234,568(음수일 때) |
| % | % 문자 | #.# % | 123456789 % |
| \u00A4 | 통화 기호 | \u00A4 #,### | ₩1,234,568 |

- **Format 형식 클래스**
  - 숫자 또는 날짜를 원하는 형태의 문자열로 변환해주는 기능 제공
  - java.text 패키지에 정의

| Format 클래스 | 설명 |
|---|---|
| DecimalFormat | 숫자를 형식화된 문자열로 변환 |
| SimpleDateFormat | 날짜를 형식화된 문자열로 변환 |

- **DecimalFormat**
  - 숫자를 형식화된 문자열로 변환하는 기능 제공

```
DecimalFormat df = new DecimalFormat("#,###.0");
String result =  df.format(1234567.89);  //1,234,567.9
```

- **DecimalFormat**
  - 변환 패턴

| 기호 | 의미 | 패턴 예 | 1234567.89 → 변환 결과 |
|---|---|---|---|
| 0 | 10진수(빈자리는 0으로 채움) | 0<br>0.0<br>0000000000.00000 | 1234568<br>1234567.9<br>0001234567.89000 |
| # | 10진수(빈자리는 채우지 않음) | #<br>#.#<br>##########.##### | 1234568<br>1234567.9<br>1234567.89 |
| . | 소수점 | #.0 | 1234567.9 |
| – | 음수 기호 | +#.0<br>–#.0 | +1234567.9<br>–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)<br>–1,234,568(음수일 때) |
| % | % 문자 | #.# % | 123456789 % |
| \u00A4 | 통화 기호 | \u00A4 #,### | ₩1,234,568 |

- **DecimalFormatExample.java**

```
package ch12.sec09;

import java.text.DecimalFormat;

public class DecimalFormatExample {
	public static void main(String[] args) {
		double num = 1234567.89;

		DecimalFormat df;

		//정수 자리까지 표기
		df = new DecimalFormat("#,###");
		System.out.println( df.format(num) );

		//무조건 소수 첫째 자리까지 표기
		df = new DecimalFormat("#,###.0");
		System.out.println( df.format(num) );
	}
}
```

```
1,234,568
1,234,567.9
```

- **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 | | |

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy년 MM월 dd일");
String strDate = sdf.format(new Date());  //2021년 11월 28일
```

- **SimpleDateFormatExample.java**

```
package ch12.sec09;

import java.text.SimpleDateFormat;
import java.util.Date;

public class SimpleDateFormatExample {
	public static void main(String[] args) {
		Date now = new Date();

		SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
		System.out.println( sdf.format(now) );

		sdf = new SimpleDateFormat("yyyy년 MM월 dd일");
		System.out.println( sdf.format(now) );

		sdf = new SimpleDateFormat("yyyy.MM.dd a HH:mm:ss");
		System.out.println( sdf.format(now) );

		sdf = new SimpleDateFormat("오늘은 E요일");
		System.out.println( sdf.format(now) );
```

```
2024-01-18
2024년 01월 18일
2024.01.18 오후 12:38:11
오늘은 목요일
```

- **SimpleDateFormatExample.java**

```
            sdf = new SimpleDateFormat("올해의 D번째 날");
            System.out.println( sdf.format(now) );

            sdf = new SimpleDateFormat("이달의 d번째 날");
            System.out.println( sdf.format(now) );
        }
}
```

```
오늘은 목요일
올해의 18번째 날
이달의 18번째 날
```

- **정규 표현식**
  - 문자 또는 숫자와 관련된 표현과 반복 기호가 결합된 문자열

| 표현 및 기호 | 설명 | | |
|---|---|---|---|
| [ ] | 한 개의 문자 | [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 | | |

- **정규 표현식**
  - 전화번호를 위한 정규 표현식
    - 02-123-1234
    - 02-124-5648

```
(02|010)-\d{3,4}-\d{4}
```

  - 이메일을 위한 정규 표현식
    - white@naver.com

```
\w+@\w+\.\w+(\.\w+)?
```

- **Pattern 클래스로 검증**
  - java.util.regex 패키지의 Pattern 클래스
    - 정규 표현식으로 문자열을 검증하는 matches() 메소드 제공

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

- **PatternExample.java**

```
package ch12.sec10;
import java.util.regex.Pattern;

public class PatternExample {
	public static void main(String[] args) {
		String regExp = "(02|010)-\\d{3,4}-\\d{4}";
		String data = "010-123-4567";
		boolean result = Pattern.matches(regExp, data);
		if(result) {
			System.out.println("정규식과 일치합니다.");
		} else {
			System.out.println("정규식과 일치하지 않습니다.");
		}

		regExp = "\\w+@\\w+\\.\\w+(\\.\\w+)?";
		data = "angel@mycompanycom";
		result = Pattern.matches(regExp, data);
		if(result) {
			System.out.println("정규식과 일치합니다.");
		} else {
			System.out.println("정규식과 일치하지 않습니다.");
		}
	}
}
```

```
정규식과 일치합니다.
정규식과 일치하지 않습니다.
```

- **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();
```

② ① 클래스로부터 얻는 방법

③ 객체로부터 얻는 방법

```
Class clazz = String.class;

Class clazz = Class.forName("java.lang.String");

String str = "감자바";
Class clazz = str.getClass();
```

- 패키지와 타입 정보 얻기
  - 패키지와 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 |

- **Car.java**

```
package ch12.sec11.exam01;
public class Car { }
```

- **CarExample.java**

```
package ch12.sec11.exam01;

public class GetClassExample {
	public static void main(String[] args) throws Exception {
		//how1, class static 필드이용
		Class clazz = Car.class;
		//how2, 클래스 문자열 이용
		//Class clazz = Class.forName("ch12.sec11.exam01.Car");
		//how3, 인스턴스를 이용
		//Car car = new Car();
		//Class clazz = car.getClass();

		System.out.println("패키지: " + clazz.getPackage().getName());
		System.out.println("클래스 간단 이름: " + clazz.getSimpleName());
		System.out.println("클래스 전체 이름: " + clazz.getName());
	}
}
```

```
패키지: ch12.sec11.exam01
클래스 간단 이름: Car
클래스 전체 이름: ch12.sec11.exam01.Car
```

- 멤버 정보 얻기
  - 타입(클래스, 인터페이스)가 가지고 있는 멤버 정보 얻기

| 메소드 | 용도 |
| --- | --- |
| Constructor[ ] getDeclaredConstructors( ) | 생성자 정보 읽기 |
| Field[ ] getDeclaredFields( ) | 필드 정보 읽기 |
| Method[ ] getDeclaredMethods( ) | 메소드 정보 읽기 |

- **Car.java**

```
package ch12.sec11.exam02;

public class Car {
	//필드
	private String model;
	private String owner;

	//생성자
	public Car() {
	}
	public Car(String model) {
		this.model = model;
	}

	//메소드
	public String getModel() { return model; }
	public void setModel(String model) { this.model = model; }
	public String getOwner() { return owner; }
	public void setOwner(String owner) { this.owner = owner; }
}
```

- **ReflectionExample.java**

```
package ch12.sec11.exam02;

import java.lang.reflect.*;

public class ReflectionExample {
	public static void main(String[] args) throws Exception {
		Class clazz = Car.class;

		System.out.println("[생성자 정보]");
		Constructor[] constructors = clazz.getDeclaredConstructors();
		for(Constructor constructor : constructors) {
			System.out.print(constructor.getName() + "(");
			Class[] parameters = constructor.getParameterTypes();
			printParameters(parameters);
			System.out.println(")");
			}
		System.out.println();
```

```
[생성자 정보]
ch12.sec11.exam02.Car()
ch12.sec11.exam02.Car(java.lang.String)
```

- **ReflectionExample.java**

```
System.out.println("[필드 정보]");
Field[] fields = clazz.getDeclaredFields();
for(Field field : fields) {
        System.out.println(field.getType().getName() + " " + field.getName());
}
System.out.println();
```

```
[필드 정보]
java.lang.String model
java.lang.String owner
```

- **ReflectionExample.java**

```
            System.out.println("[메소드 정보]");
            Method[] methods = clazz.getDeclaredMethods();
            for(Method method : methods) {
                System.out.print(method.getName() + "(");
                Class[] parameters = method.getParameterTypes();
                printParameters(parameters);
                System.out.println(")");
            }
        }

        private static void printParameters(Class[] parameters) {
            for(int i=0; i<parameters.length; i++) {
                System.out.print(parameters[i].getName());
                if(i<(parameters.length-1)) {
                    System.out.print(",");
                }
            }
        }
}
```

```
[메소드 정보]
getOwner()
setOwner(java.lang.String)
setModel(java.lang.String)
getModel()
```

- 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 |

- 리소스 경로 얻기
  - 가정

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

```
String photo1Path = clazz.getResource("photo1.jpg").getPath();
                                                    //C:\...\bin\photo1.jpg

String photo2Path = clazz.getResource("images/photo2.jpg").getPath();
                                                    //C:\...\bin\images\photo2.jpg
```

- **Car.java**

```
package ch12.sec11.exam03;

public class Car {
}
```


- **CarExample.java**

```
package ch12.sec11.exam03;

public class GetResourceExample {
	public static void main(String[] args) {
		Class clazz = Car.class;

		String photo1Path = clazz.getResource("photo1.jpg").getPath();
		String photo2Path = clazz.getResource("images/photo2.jpg").getPath();

		System.out.println(photo1Path);
		System.out.println(photo2Path);
	}
}
```

```
/C:/fullstack/java/study/out/production/study/ch12/sec11/exam03/photo1.jpg
/C:/fullstack/java/study/out/production/study/ch12/sec11/exam03/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 타입 정의와 적용

```
public @interface AnnotationName {
  String prop1();    // 기본값이 없으므로, 필수 속성이 됨
  int prop2() default 1;  // 기본값이 지정, 생략 가능
}
```

```
@AnnotationName(prop1 = "값")
@AnnotationName(prop1 = "값", prop2 = 3)
```

○ value 기본 속성

```
public @interface AnnotationName {
  String value();
  int prop2() default 1;
}
```

```
@AnnotationName("값")    // 기본 속성만 지정하는 경우
```

```
@AnnotationName(value = "값", prop2 = 3) // 다른 속성 지정시, value 속성 지정
```

- 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 |

- 어노테이션 적용 대상

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

```java
@AnnotationName                          // TYPE(클래스)에 적용
public class ClassName {
  @AnnotationName                        // 필드에 적용
  private String fieldName;

  //@AnnotationName                      // @Target에 CONSTRUCT가 없으므로 생성자에는 적용 못함
  public ClassName() { }

  @AnnotationName                        // 메소드에 적용
  public void methodName() { }
}
```

- 어노테이션 유지 정책
  - 어노테이션 정의 시 @AnnotationName을 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 |

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

- **어노테이션 설정 정보 이용**
  - 애플리케이션은 리플렉션을 이용해 적용 대상에서 어노테이션의 정보를 다음 메소드로 얻어냄

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

- **PrintAnnotation.java**

```
package ch12.sec12;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target({ElementType.METHOD})                          // 적용 대상: METHOD
@Retention(RetentionPolicy.RUNTIME)          // 유지 정책: RUNTIME
public @interface PrintAnnotation {
      String value() default "-";              // value 속성: 선의 종류
      int number() default 15;                       // number 속성: 출력 횟수
}
```

- **Service.java**

```
package ch12.sec12;

public class Service {
	@PrintAnnotation
	public void method1() {
		System.out.println("실행 내용1");
	}

	@PrintAnnotation("*")
	public void method2() {
		System.out.println("실행 내용2");
	}

	@PrintAnnotation(value="#", number=20)
	public void method3() {
		System.out.println("실행 내용3");
	}
}
```

- **printLine.java**

```
package ch12.sec12;

import java.lang.reflect.Method;

public class PrintAnnotationExample {
	public static void main(String[] args) throws Exception {
		Method[] declaredMethods = Service.class.getDeclaredMethods();
		for(Method method : declaredMethods) {
			//PrintAnnotation 얻기
			PrintAnnotation printAnnotation = method.getAnnotation(PrintAnnotation.class);

			//설정 정보를 이용해서 선 출력
			printLine(printAnnotation);

			//메소드 호출
			method.invoke(new Service());

			//설정 정보를 이용해서 선 출력
			printLine(printAnnotation);
		}
	}
```

- **printLine.java**

```
    public static void printLine(PrintAnnotation printAnnotation) {
        if(printAnnotation != null) {
            //number 속성값 얻기
            int number = printAnnotation.number();
            for(int i=0; i<number; i++) {
                //value 속성값 얻기
                String value = printAnnotation.value();
                System.out.print(value);
            }
            System.out.println();
        }
    }
}
```

```
---------------
실행 내용1
---------------
***************
실행 내용2
***************
###################
실행 내용3
###################
```