6차시 정리

```
// 패키지의 개념과 필요성

// 응용프로그램을 분담하여 개발할 경우 동일한 이름의 클래스가 존재할 가능성이 있는데 이러면 합칠때 오류발생

// 패키지
// 서로 관련된 클래스와 인터페이스의 컴파일 된 클래스 파일들을 하나의 디렉터리에 묶어놓은 것

// 모듈
// 여러 패키지와 이미지 등의 자원을 모아 놓은 컨테이너
// JDK 9부터 자바 API 의 모든 클래스들을 패키지 기반에서 모듈들로 완전히 재구성
// 응용프로그램 역시 여러 개의 모듈로 분할하여 작성가능
// Java 9부터 전면적으로 도입, 복잡한 개념, 큰 자바 응용프로그램에는 개발, 유지보수 등에 적합
// 모듈의 목적
// 자바 API를 여러 모듈(99개)로 분할하여 응용프로그램의 실행에 적합한 모듈들로만 실행 환경을 구출할 수 있도록 함
// 메모리 등의 자원이 열악한 작은 소형기기에 꼭 필요한 모듈로 구성된 작은 크기의 실행 이미지를 만들기 위함.

// 자바 JDK에서 제공되는 모듈 파일들
// 자바가 설치된 Jmods 디렉터리에 모듈 파일 존재(모듈파일은 압축되어있다. jdk10번의 경우 99개의 모듈이 있다.)
// 모듈 파일에는 자바 API의 패키지와 클래스들이 들어있다.
// jmod명령을 이용하여 모듈 파일에 있는 패키지들을 풀어낼수 있다.

// 예) Scanner 클래스는 java파일안에 util파일안에 존재하므로 경로는 java.util.Scanner 이다.

// 다른 패키지에 작성된 클래스를 사용하기 위해서는 import를 사용하거나 안하거나의 방법으로 나뉜다.
// import를 이용하지 않는 경우 : 클래스를 사용할때 경로를 다 써줘야함.
// 예) java.util.Scanner scanner = new java.util.Scanner(System.in);
// import를 사용하는 경우 큭정 클래스의 이름만 선언하거나 *를 사용하여 전부 선언한다음 클래스를 사용한다.
// 예) import java.util.Scanner or import java.util.*
// 예) Scanner scanner = new Scanner(System.in);

// 패키지 선언
// 소스 파일 첫줄에 'package 패키지이름'을 선언하여 작성되는 클래스를 작성한 패키지에 저장하라는 뜻
// 예) package UI;
// 예) public class Tools { ... } // 이제 이 클래스의 경로명은 UI.Tools 가된다.
// 다른 클래스에서 Tools 클래스를 쓰려면 import UI.Tools; 라고 선언해야 한다.
```

Package 프로젝트 - lib 패키지 -

```
package lib;

// 다른 패키지에서 Calculator 클래스에 접근하도록 하기 위해 public으로 선언
public abstract class Calculator {
	public abstract int add(int a, int b);
	public abstract int subtract(int a, int b);
	public abstract double average(int[] a);
}
```

Package 프로젝트 - app 패키지 -

```
package app;
// 다른 패키지에 있는 추상 클래스를 불러와 오버라이딩을 하기위해 import 패키지이름.클래스이름 을 추가한다.
import lib.Calculator;

public class GoodCalc extends Calculator{ // 추상 클래스를 상속받아 오버라이딩하여 사용
	public int add(int a, int b) { // 오버라이딩
		return a + b;
	}
	public int subtract(int a, int b) { // 오버라이딩
		return a - b;
	}
	public double average(int[] a) { // 오버라이딩
		double sum = 0; // 초기화 무조건 해줘야함
		for(int i=0; i < a.length ; i++) {
			sum += a[i];
		}
		return sum / a.length;
	}
	public static void main(String[] args) {
		Calculator c = new GoodCalc() ; // 업캐스팅
		System.out.println(c.add(2,3));
		System.out.println(c.subtract(2,3));
		System.out.println(c.average(new int [] {2,3,4}));
	}
}
// Run Confiquraions 에서 main class가 GoodCalc인걸 확인한다음 실행한다.

// 출력
// 5
// -1
// 3.0
```

```
// 디폴트 패키지
// package 선언문이 없이 만들어진 클래스의 패키지로 현재 디렉터리를 뜻한다.

// 패키지의 특징
// 패키지는 계층구조이다 : 관련된 클래스 파일들 하나의 패키지로 계층화하여 관리용이
// 패키지별 접근 제한 : 패키지 별로 접근 권한 가능
// 동일한 이름의 클래스와 인터페이스의 사용가능 : 서로다른 패키지에 같은 이름의 파일이 존재가능
// 높은 소프트웨어 재사용성

// 모듈
// Java 9에서 도입된 개념
// 패키지와 이미지 등의 리소스를 담은 컨테이너
// 모듈 파일(.jmod)로 저장
```

```
// 자바 플랫폼
// 자바의 개발 환경(JDK)과 자바의 실행환경(JRE)을 지칭, Java SE(자바 API) 포함
// 자바 API의 모든 클래스는 여러개의 모듈로 재구성되어 JDK의 jmods 디렉터리에 저장되어 배포
// 모듈 파일로부터 모듈을 부는 명령
// 예) jmod extract "C:\Program Files\java\jdk-10\jmods\java.base.jmod"
// 예) - 현재 디렉터리에 java.base 모듈이 풀림

// 자바 실행 환경
// JRE : 디폴트 자바 실행 환경
// 자바 모듈, 자바 가상 기계 등으로 구성, 원래는 실행되는 컴퓨터에 저장하고 사용하다가(용양이 매우 큼)
// 자바9 이우 모듈의 비공개 파일에 저장하여 응용프로그램 실행시 모듈 파일에서 로딩을 하도록 바뀌었음

// Custom JRE : 맞줌형 실행 환경 -> 오라클이 추구하는 방향이다.
// 자바 8까지 소형기기에는 용량이 부족해 사용하지 못하다가 자바9이후 모듈의 탄생으로 필요한 모듈만 가지고
// 메모리가 열악한 소형 기기에서도 Custom JRE구축이 가능해졌다.

// 자바 모듈화의 목적
// 자바 컴포넌트들을 필요에 따라 조립하여 사용하기 위하여 사용하고
// 불필요한 모듈의 로딩 및 저장을 안하도록 하여 시스템의 부담이 감소시켜 소형기기에도 사용 가능하게 한다.

// 주요패키지
// java.lang(자바 언어 패키지)-스트링, 함수, 입출력 등 기본적인 클래스와 인터페이스(import문 없이 사용가능)
// java.util(자바 유틸리티 패키지)-날짜, 시간 등 다양한 유틸 클래스와 인터페이스
// java.id(자바 입출력 패키지)-키보드, 모니터 등으로 입출력할 수 잇게 하는 클래스와 인터페이스
// java.awt(자바 GUI 패키지)-GUI 프로그래밍을 위한 클래스와 인터페이스
// javax.swing(자바 GUI 패키지)-GUI 프로그래밍을 위한 스윙 패키지(java.awt 경량화)

// Object 클래스
// java.lang 패키지에 포함 - import 선언 필요 없음
// 모든 클래스의 수퍼 클래스
// 모든 클래스에 강제 상속이고 모든 객체가 공통으로 가지는 객체의 속성을 나타내는 메소드 보유
// 못보던 메소드들이 포함

class Point {
	private int x, y;
	public Point(int x, int y) {
		this.x = x;
		this.y = y;
	}
}
public class ObjectPropertyEx {
	public static void print(Object obj) {
		// 현 객체의 런타임 클래스의 이름을 리턴(.getName이 없으면 Class Point반환)
		System.out.println(obj.getClass().getName()); // 클래스 이름
		// 현 객체에 대한 해시 코드 값 리턴
		System.out.println(obj.hashCode()); // 해시 코드 값, 16진로 반환
		// 현 객체에 대한 문자열 표현을 리턴
		System.out.println(obj.toString()); // 객체를 문자열로 만들어 출력
		System.out.println(obj); // 객체 출력
	}
	public static void main(String [] args) {
		Point p = new Point(2,3);
		print(p);
	}
}
// 츨력
// Point
// 1665404403
// Point@63440df3
// Point@63440df3
```

```
// toString()
// java.lang 모듈에 Object 클래스 내 주요 메소드
// 다음과 같이 구현되어 있음
// public String toString() {
//          return getClass().getName() + "@" + Integer.toHexString(hashCode());
// }
// 객체를 문자열로 반환해주며
// 유리가 계속 쓰던 문자열 출력은 다 이 메소드를 통해 출력하는 것이다.
// 예) 평소 print(p) -> 사실 print(p.toString())
// 개발자는 자신만의 toString()작성 필요

class Point {
        private int x, y;
        public Point(int x, int y) {
                this.x = x;
                this.y = y;
        }
        public String toString() {
                return "Point(" + x + "," + y + ")";
        }
}
public class toStringEx {
        public static void main(String [] args) {
                Point p = new Point(2,3);
                System.out.println(p.toString());
                System.out.println(p); // p는 p.toString()으로 자동 변환
                System.out.println(p + "입니다."); // p.toString() + "입니다"로 자동 변환
        }
}

// 출력
// Point(2,3)
// Point(2,3)
// Point(2,3)입니다.
```

```
// 객체 비교와 equals()
// = 연산자
// 기본타입이 아니면 전부 레퍼런스(주소)를 넘겨준다.

// == 연산자
// 두개의 래퍼런스 비교, 주소가 같은지 비교한다.

// boolean equals(Object obj)
// 객체 내용이 같은지 비교, 안데 있는 값이 같은지 확인한다.

class Point{
	private int x, y;;
	public Point(int x, int y) {
		this.x = x;
		this.y = y;
	}
	public boolean equals(Object obj){
		Point p = (Point)obj;
		if(x == p.x && y == p.y)
			return true;
		else
			return false;
	}
}
public class EqualsEx {
	public static void main(String[] args) {
		Point a = new Point(2,3);
		Point b = new Point(2,3);
		Point c = new Point(3,4);
		Point d = a;

		if(a == b) // false
			System.out.println("a==b");
		if(a == d) // true
			System.out.println("a==d");
		if(b == d) // false
			System.out.println("b==d");
		if(a.equals(b)) // true
			System.out.println("a is equal to b");
		if(a.equals(c)) // false
			System.out.println("a is equal to c");
		if(b.equals(d)) // true
			System.out.println("b is equal to d");
	}
}

// 출력
// a==d
// a is equal to b
// b is equal to d
```

```
// 객체 비교와 equals()
// = 연산자
// 기본타입이 아니면 전부 레퍼런스(주소)를 넘겨준다.

// == 연산자
// 두개의 래퍼런스 비교, 주소가 같은지 비교한다.

// boolean equals(Object obj)
// 객체 내용이 같은지 비교, 안데 있는 값이 같은지 확인한다.
class Rect {
	private int width;
	private int height;
	public Rect(int width, int height) {
		this.width = width;
		this.height = height;
	}
	public boolean equals(Object obj) {
		Rect p = (Rect)obj;
		if (width*height == p.width*p.height)
			return true;
		else
			return false;
	}
}
public class EqualsExRect {
	public static void main(String[] args) {
		Rect a = new Rect(2,3);
		Rect b = new Rect(3,2);
		Rect c = new Rect(6,1);
		Rect d = new Rect(3,4);
		Rect e = new Rect(6,2);

		if(a.equals(b)) // True
			System.out.println("a is equal to b");
		if(b.equals(c)) // True
			System.out.println("b is equal to c");
		if(a.equals(c)) // True
			System.out.println("a is equal to c");
		if(a.equals(d)) // False
			System.out.println("a is equal to d");
		if(d.equals(e)) // True
			System.out.println("d is equal to e");
	}
}

// 출력
// a is equal to b
// b is equal to c
// a is equal to c
// d is equal to e
```

```
// Wrapper 클래스
// 자바의 기본 타입을 클래스화한 8개 클래스
// 기본타입-Wrapper클래스 짝을 지었을때
// byte-Byte , short-Short , int-Integer , long - Long
// float-Float , double-Double , char-Character , boolean-Boolean

// 이름이 Wrapper인 클래스는 존재하지 않음
// Wrapper클래스로 표현하는 이유 : 기본 타입의 값을 객테로 다룰 수 있게 함

// Wrapper 객체들은 거의 유사, 많은 메소드가 static 타입

// 기본타입의 값으로 Wrapper 객체 생성(자바 9부터 new객체를 이용한 Wrapper 생성 폐기)
// Integer ii = Integer.valueOf(10);
// Character cc = Character.valueOf('c');
// Float ff = Float.valueOf((double)3.14); // Float 객체는 double 타입의 값으로 생성 가능
// Double dd = Double.valueOf(3.14);
// Boolean bb = Boolean.valueOf(false);

// Wrapper 객체로부터 기본 타입 값 알아내기
// int i = ii.intValueOf();
// char c = cc.charValue();
// double d = dd.doubleValue();
// boolean b = bb.booleanValue();

// 기본타입을 문자열로 변환
// String s1 = Integer.toString(123);
// String s2 = Integer.toHexString(123); // 정수 123을 16진수의 문자열 "7b"로 변환
// String s3 = Double.toString(3.14);
// String s4 = Character.toString('a');
// String s5 = Booiean.toString(true);

// 문자열을 기본타입으로 변환
// int i = Integer.parseInt(s1);
// double d = Double.parseDouble(s3);
// boolean b = Boolean.parseBoolean(s5)

// 문자열로 Wrapper 객체 생성
// Integer ii = Integer.valueOf("10");
// Double dd = Double.valueOf("3.14");
// Boolean bb = Boolean.valueOf("false");
```

```
public class WrapperEx {
	public static void main(String[] args) {

		System.out.println(Character.toLowerCase('A')); // 'A'를 소문자로 변환
		char c1='4', c2='F';

		if(Character.isDigit(c1)) // 문자 c1이 숫자이면 true
			System.out.println(c1 + "는 숫자");

		if(Character.isAlphabetic(c2)) // 문자 c2가 영문자이면 true
			System.out.println(c2 + "는 영문자\n");

		System.out.println(Integer.parseInt("-123")); // "-123"을 10진수로 변환
		System.out.println(Integer.toHexString(28)); // 정수 28을 16진수 문자열로 변환
		System.out.println(Integer.toBinaryString(28)); // 28을 2진수 문자열로 변환
		System.out.println(Integer.bitCount(28)+"\n"); // 28에 대한 2진수의 1의 개수

		Double d = Double.valueOf(3.14);
		System.out.println(d.toString()); // Double을 문자열 "3.14"로 변환
		System.out.println(Double.parseDouble("3.14")+"\n"); // 문자열을 실수 3.14로 변환

		boolean b = (4>3); // b는 true
		System.out.println(Boolean.toString(b)); // true를 문자열 "true"로 변환
		System.out.println(Boolean.parseBoolean("false")); // 문자열을 false로 변환
	}
}
// 출력
// a
// 4는 숫자
// F는 영문자
//
// -123
// 1c
// 11100
// 3
//
// 3.14
// 3.14
//
// true
// false
```

```
// 박싱과 언박싱
// 박싱 : 기본타입의 값을 Wrapper 객체로 변환
// 언박싱 : Wrapper 객체에 들어 있는 기본 타입의 값을 빼내는 것

// JDK1.5부터 자동 박싱과 자동 언박싱을 제공

public class AutoBoxingUnBoxingEx {
	public static void main(String[] args) {

		int n = 10;
		Integer intObject = n; // 자동 박싱(auto boxing)
		System.out.println("intObject = " + intObject);

		int m = intObject + 10; // 자동 언박싱(auto unboxing)
		System.out.println("m = " + m);
	}
}
// 출력
// intObject = 10
// m = 20
```

```
// String 클래스
// 하나의 문자열로 표현, 리스트 형식으로 들어와도 문자열로 표현

// java.lang 패키지에 존재
// 주소 : java.lang.String

// 문자열(String) 생성 방법
// 리터럴로 생성하는 방법, 예) String s = "Hello";
// JVM이 리터럴 관리, 같은 문자열을 저장한다고 하면 두 변수를 주소를 공유한다.

// String 객체로 생성하는 방법, 예) String t = new String("Hello")
// 힙 메모리에 String 객체 생성, 같은 문자열을 저장해도 주소가 다 다르다. 하지만 문자열의 수정은 불가능하다.

// String 객체의 주요기능
// 스트링 객체는 수정이 불가능하다.
// 기존에 문자열에 문자열을 붙여서 새로운 문자열을 만들수 있다. concat 메소드 사용
// 스트릴 비교시 반드시 주소가 아닌 값을 비교하는 equals() 메소드를 사용한다.
// 문자열 비교시 compareTo() 메소드를 사용한다. 사전의 문자 순서를 기준으로 음수, 0, 양수를 리턴한다.

// 문자열 비교시 compareTo() 메소드를 사용
// 문자열을 비교할 때 == 를 사용하면 안된다.

// 문자열에 문자 포함 확인
// contains() 메소드를 이용하여 문자열 안에 해당 문자가 들어있는지 확인한다.

// 문자열 대체하기
// replace(변하기전문자,바꿀문자) 메소드를 이용하여 해당문자를 원하는 문자로 바꾼다.

// 문자열 연결
// 기본타입의 문자열은 + 로 연결이 가능하다.
// String 객체 타입의 문자열은 concat() 메소드를 이용하여 새로운 문자열을 만들어 낸다.

// 문자열 분리
// split(기준문자) 메소드를 기준으로 문자열을 나눈다. 저장하는 변수는 배열로 선언하는게 좋다.

// 문자열 부분저장
// substring(인덱스) 메소드를 이용하여 인덱스부터의 문자열을 따로 분리한다.

// 문자열 공백 제거
// trim() 메소드를 이용하여 문자열 앞, 뒤에 공백 문자(탭, 엔터, 스페이스)들을 제거한다.

// 문자열 길이
// length() 메소드를 이용하여 문자열의 길이를 구한다. 공백 또한 포함한다.

// 문자열의 각 문자 접근
// charAt(index) 메소드를 이용하여 인덱스 번호의 문자에 접근한다. 인덱스 시작은 0부터 한다.
```

```
public class StringEx {
	public static void main(String[] args) {

		String a = new String(" C#");
		String b = new String(",C++ ");
		System.out.println(a + "의 길이는 " + a.length()); // 문자열의 길이(문자 개수)
		System.out.println(a.contains("#")); // 문자열의 포함 관계

		System.out.println(a.compareTo(" C#")); // 두 문자열이 같은지 확인, 같으면 0
		System.out.println(a.equals("C#")); // 두 문자열이 같은지 확인, 같으면 True

		a = a.concat(b); // 문자열 연결
		System.out.println(a);

		a = a.trim(); // 문자열 앞 뒤의 공백 제거
		System.out.println(a);

		a = a.replace("C#","Java"); // 문자열 대치
		System.out.println(a);

		String s[] = a.split(","); // 문자열 분리

		for (int i=0; i<s.length; i++)
			System.out.println("분리된 문자열" + i + ": " + s[i]);

		a = a.substring(5); // 인덱스 5부터 끝까지 서브 스트링 리턴
		System.out.println(a);

		char c = a.charAt(2); // 인덱스 2의 문자 리턴
		System.out.println(c);
	}
}

// 출력
//  C#의 길이는 3
// true
// 0
// false
//  C#,C++
// C#,C++
// Java,C++
// 분리된 문자열0: Java
// 분리된 문자열1: C++
// C++
// +
```

```
// StringBuffer 클래스
// 가변 크기의 문자열 저장 클래스
// java.lang 패키지에 StringBuffer 클래스

// String 클래스와 달리 문자열 변경 가능
// StringBuffer 객체의 크기는 스트링의 길이에 따라 가변적
// 생성, 예) StringBuffer sb = new StringBuffer("java");

// 스트링 버퍼에 연결
// append() 메소드를 이용하여 스트링 버터를 연결하는데 문자열을 연결하면

// 스트링 버퍼의 현재 크기 반환
// capacity() 메소드를 이용하여 스트링 버퍼의 현재 크기를 반환한다.

// 스트칭 버퍼의 입부분 삭제
// delete(시작,끝) 메소드를 이용하여 시작과 끝사이의 문자열을 삭제한다.

// 스트링 버퍼의 삽입
// insert(인덱스, 문자열) 메소드를 이요하여 인덱스 부분에 문자열을 삽입한다.

// 스트링 버퍼에 문자열 대체하기
// replace(시작, 끝, 문자열)메소드를 이용하여 시작과 끝 사이를 문자열로 대체한다.

// 스트링 버퍼에 역순
// reverse() 메소드를 이용하여 스트링 버퍼 문자열의 순서를 바꿔서 반환한다.

// 스트링 버퍼의 길이 재정의
// setLength(길이) 메소드를 이용하여 기존의 스트링 버퍼 문자열을 길이에 맞게 다시 설정한다.

public class StringBufferEx {
	public static void main(String[] args) {

		StringBuffer sb = new StringBuffer("This");

		sb.append(" is pencil"); // 문자열 덧붙이기
		System.out.println(sb);

		sb.insert(7, " my"); // "my" 문자열 삽입
		System.out.println(sb);

		sb.replace(8, 10, "your"); // "my"를 "your"로 변경
		System.out.println(sb);

		sb.delete(8, 13); // "your " 삭제
		System.out.println(sb);

		sb.setLength(4); // 스트링 버퍼 내 문자열 길이 수정
		System.out.println(sb);
	}
}

// 출력
// This is pencil
// This is my pencil
// This is your pencil
// This is pencil
// This
```

```
// StringTokenizer
// java.util 패키지에 String.Tokenizer 클래스
// 하나의 문자열을 여러 문자열로 분리 해준다.
// 생성 예) StringTokenizer stz = new StringTokenizer(문자열, 구분기호)
// 구분기호에 문자열로 들어온다면 그 문자열의 요소 하나하나가 구분기호가 된다.
// String에 split() 메소드를 이용하여 동일한 구현 가능

// 토크나이저가 분리한 토큰의 개수 리턴
// countTokens() 메소드를 사용

// 토크나이저에 다음 토큰이 있으면 True 리턴 아니면 False 리턴
// hasMoreTokens() 메소드 사용

// 토크나이저에 들어 잇는 다음 토큰 리턴
// nextToken() 메소드 사용

import java.util.StringTokenizer;

public class StringTokenizerEx {
	public static void main(String[] args) {

		StringTokenizer st = new StringTokenizer("홍길동/장화/홍련/콩쥐/팥쥐", "/");

		System.out.println(st.countTokens()); // 토크나이저가 분리한 토큰의 개수 리턴

		while (st.hasMoreTokens())
			System.out.println(st.nextToken()); // 토크나이저에 들어 잇는 다음 토큰 리턴
	}
}

// 출력
// 5
// 홍길동
// 장화
// 홍련
// 콩쥐
// 팥쥐
```

```
// Math 클래스
// 산술연산 메소드 제공
// java.lang 패키지에 Math 클래스
// 모든 매소드는 static 타입이므로 클래스 이름으로 바로 호출해야한다.

// 난수 발생
// Math.random() 메소드를 사용하여 0이상 1미만의 난수를 발생시킨다.
// 0 부터 100까지 난수 생성-> (int)(Math.random()*101)
// 1 부터 100까지 난수 생성-> (int)(Math.random()*100 + 1)

// java.util.Random 클래스를 이요하여 다양한 형태로 난수 발생 가능

public class MathEx {
	public static void main(String[] args) {

		System.out.println(Math.PI); // 원주율 상수 출력
		System.out.println(Math.ceil(Math.PI)); // ceil(올림)
		System.out.println(Math.floor(Math.PI)); // floor(내림)
		System.out.println(Math.sqrt(9)); // 제곱근
		System.out.println(Math.exp(2)); // e의 2승
		System.out.println(Math.round(3.14)); // 반올림

		// [1, 45] 사이의 정수형 난수 5개 발생
		System.out.print("이번주 행운의 번호는 ");
		for(int i=0; i<5; i++)
			System.out.print((int)(Math.random()*45 + 1) + " ");
	}
}

// 쿨력
// 3.141592653589793
// 4.0
// 3.0
// 3.0
// 7.38905609893065
// 3
// 이번주 행운의 번호는 28 35 29 30 7
```

```
// Calendar 클래스
// java.util 패키지에 Calendar 클래스
// 시간과 날짜 정보를 저장 및 관리

// 생성 예) Calendar now = Calendar.getInstance();
// now는 현재의 날짜와 시간 정보를 가지고 정보를 생성
// Calendar 클래스는 추상 클래스 이므로 new를 이용한 선언 불가

// 날짜와 시간 알아내기
// int year = now.get(Calendar.YEAR); -> now에 저장된 년도 저장
// int month = noe.get(Calendar.MONTH) + 1 -> now에 저장된 달

// 날짜와 시간 설정하기
// now.clear(); -> 현재의 시간 및 날짜를 지운다.
// now.get(2000, 03, 29); -> 2000년 4월 29일로 설정, 달은 0~11로 표현되기 때문에 1을 빼준다.
// now.get(Calendar.HOUR_OF_DAY, 20); -> 저녁 8시로 설정
// now.get(Calendar.MIMUTE, 30); -> 30분으로 설정
```

```
import java.util.Calendar;

public class CalendarEx {
	public static void printCalendar(String msg, Calendar cal) {

		int year = cal.get(Calendar.YEAR); // get()은 0~30까지의 정수 리턴.
		int month = cal.get(Calendar.MONTH) + 1;
		int day = cal.get(Calendar.DAY_OF_MONTH);

		int dayOfWeek = cal.get(Calendar.DAY_OF_WEEK);

		int hourOfDay = cal.get(Calendar.HOUR_OF_DAY);  // 24시간 형식으로 시간 반환

		int ampm = cal.get(Calendar.AM_PM);
		int hour = cal.get(Calendar.HOUR);  // 12시간 형식으로 시간 반환M);
		int minute = cal.get(Calendar.MINUTE);
		int second = cal.get(Calendar.SECOND);
		int millisecond = cal.get(Calendar.MILLISECOND);

		System.out.print(msg + year + "/" + month + "/" + day + "/");
		switch(dayOfWeek) {
			case Calendar.SUNDAY : System.out.print("일요일"); break;
			case Calendar.MONDAY : System.out.print("월요일"); break;
			case Calendar.TUESDAY : System.out.print("화요일"); break;
			case Calendar.WEDNESDAY : System.out.print("수요일"); break;
			case Calendar.THURSDAY : System.out.print("목요일"); break;
			case Calendar.FRIDAY: System.out.print("금요일"); break;
			case Calendar.SATURDAY : System.out.print("토요일"); break;
		}
		System.out.print("(" + hourOfDay + "시)");
		if(ampm == Calendar.AM)
			System.out.print("오전");
		else
			System.out.print("오후");

		System.out.println(hour + "시 " + minute + "분 " + second + "초 "+ millisecond +"밀리초");
	}

	public static void main(String[] args) {

		Calendar now = Calendar.getInstance();
		printCalendar("현재 ", now);

		Calendar firstDate = Calendar.getInstance();
		firstDate.clear(); // 날짜 및 시간 정보 초기화
		firstDate.set(2016, 11, 25); // 2016년 12월 25일. 12월을 표현하기 위해 month에 11로 설정
		firstDate.set(Calendar.HOUR_OF_DAY, 20); // 저녁 8시
		firstDate.set(Calendar.MINUTE, 30); // 30분
		printCalendar("처음 데이트한 날은 ", firstDate);
	}
}

// 출력
// 현재 2023/10/29/일요일(5시)오전5시 17분 20초 283밀리초
// 처음 데이트한 날은 2016/12/25/일요일(20시)오후8시 30분 0초 0밀리초
```