

6장 상속 패키지와 모듈

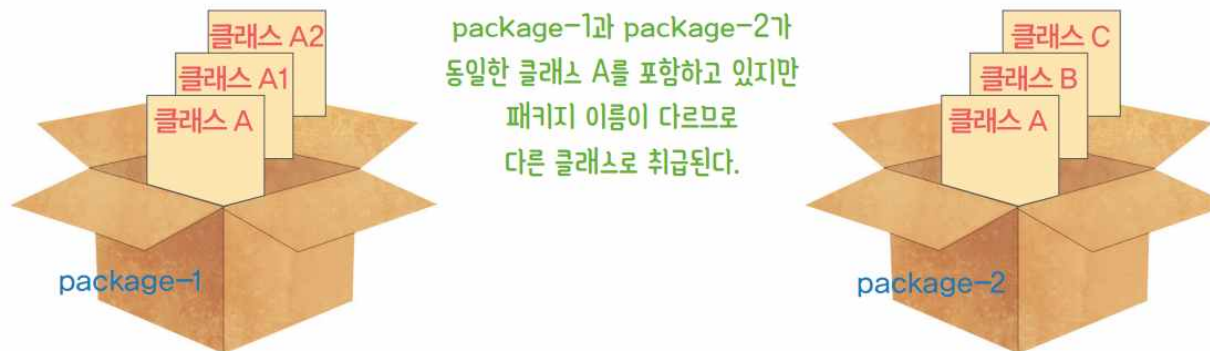
목차

- 상속의 개념과 필요성
- 클래스 상속
- 메서드 오버라이딩
- 패키지
- 자식 클래스와 부모 생성자
- 상속과 접근제어
- **final** 클래스와 메서드
- 타입 변환과 다형성
- 모듈화

패키지

■ 의미

- 클래스 파일을 묶어서 관리하기 위한 수단으로 파일 시스템의 폴더를 이용
- 패키지에 의한 장점
 - 패키지마다 별도의 이름 공간(Namespace)이 생기기 때문에 클래스 이름의 유일성을 보장.
 - 클래스를 패키지 단위로도 제어할 수 있기 때문에 좀 더 세밀하게 접근 제어



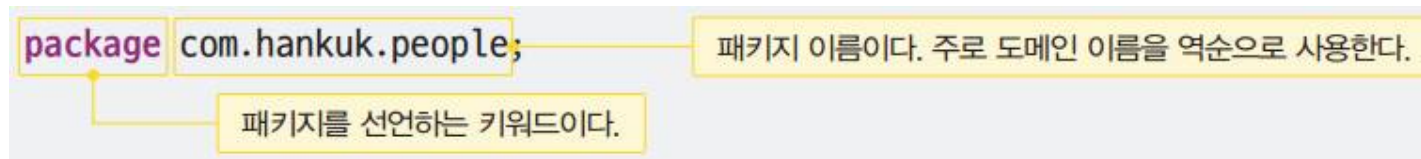
■ 대표적인 패키지

- java.lang 패키지는 import 문을 선언하지 않아도 자동으로 임포트되는 자바의 기본 클래스를 모아 둔 것
- java.awt 패키지는 그래픽 프로그래밍에 관련된 클래스를 모아 둔 것
- java.io 패키지는 입출력과 관련된 클래스를 모아 둔 것

패키지

■ 패키지 선언

- 주석문을 제외하고 반드시 첫 라인에 위치
- 패키지 이름은 모두 소문자로 명명하는 것이 관례. 일반적으로 패키지 이름이 중복되지 않도록 회사의 도메인 이름을 역순으로 사용



- url 예
http://www.bc.ac.kr

패키지

■ 패키지 선언

- 예 : 소스 코드와 컴파일(명령창)

[예제 6-6] 패키지 클래스 C:\Temp\Yona.java

```
01 package com.hankuk.people;
02
03 public class Yona {
04     public static void main(String[] args) {
05         System.out.println("안녕, 연아!");
06     }
07 }
```

C:\Temp> javac -d . Yona.java

패키지 이름은 반드시 첫 행에 위치해야 한다.

현재 폴더를 의미한다.

컴파일 명령어이다.

패키지 폴더를 생성하는 옵션이다.

패키지

■ 패키지 선언

- 예 : 실행 결과(명령창)



```
C:\Temp> java com.hankuk.people.Yona
안녕, 연아!
```

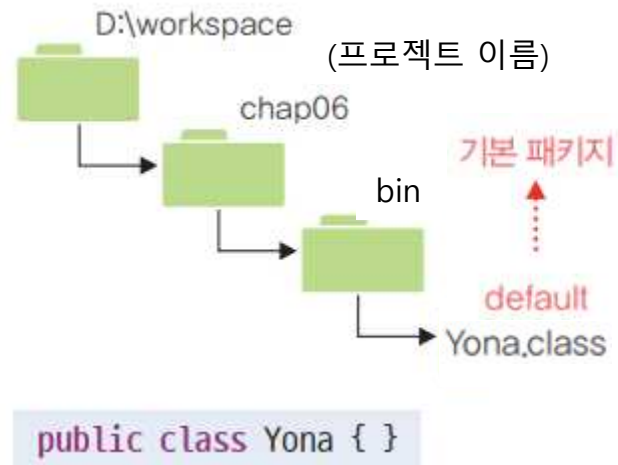
패키지 이름

클래스 이름(확장자가 없어야 한다.)

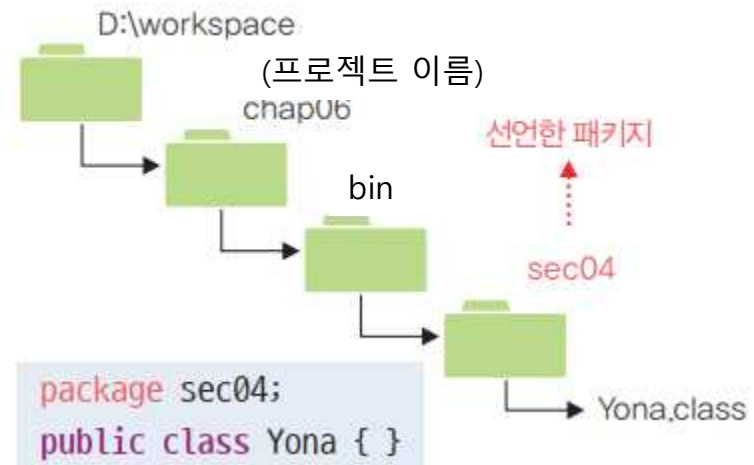
패키지

■ 패키지 선언

- 예 : 실행 결과(이클립스)



(a) 패키지를 선언하지 않을 때



(b) 패키지를 선언할 때

패키지

■ 패키지의 사용

- 다른 패키지에 있는 공개된 클래스를 사용하려면 패키지 경로를 컴파일러에게 알려줘야 한다.



패키지

■ import 문

- 패키지의 경로를 미리 컴파일러에게 알려주는 문장

```
import 패키지이름.클래스;
```

또는

```
import 패키지이름.*;
```

- import 문은 소스 파일에서 package 문과 첫 번째 클래스 선언부 사이에 위치

- 주의 사항

```
import com.hankuk.*;           // com.hankuk 패키지에 포함된 모든 클래스이다.  
import com.hankuk.people.*;    // com.hankuk.people 패키지에 포함된 모든 클래스이다.
```

=> *는 패키지 경로에 있는 모든 클래스를 의미한다. 그러나 *는 패키지 아래의 다른 패키지 경로까지는 포함하지 않기에 2개의 import문을 모두 선언해야 한다.

패키지

■ import 문

● 예제

```
package com.hankuk.people;
```

```
import com.usa.people.Lincoln;
```

컴파일러에 Lincoln 클래스의 경로를 알려 준다.

```
public class ShowWorldPeople {
```

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

```
        Lincoln greatman = new Lincoln();
```

```
    }
```

```
}
```

import 문으로 경로를 알려 주었으므로 com.usa.people
이라는 경로 정보는 필요 없다.

패키지

■ 정적 import 문

- 패키지 단위로 임포트하지 않고 패키지 경로와 정적 메서드를 함께 임포트
- 예제 : [sec04/StaticImportDemo](#)

```
package sec04;

import static java.util.Arrays.sort; //정적 import문 : 패키지 경로와 정적 메서드를 함께 임포트
import java.util.Calendar;          // 비정적 import문

public class StaticImportDemo {
    public static void main(String[] args) {
        int[] data = { 3, 5, 1, 7 };

        sort(data); // 정적 import문이 있어서 클래스 이름 없이 멤버를 사용 가능

        System.out.println(Calendar.JANUARY); //import문일때는 클래스 이름과 함께 필드 사용한다.

        Calendar.getInstance(); //import문일때는 클래스 이름과 함께 메서드를 사용한다.

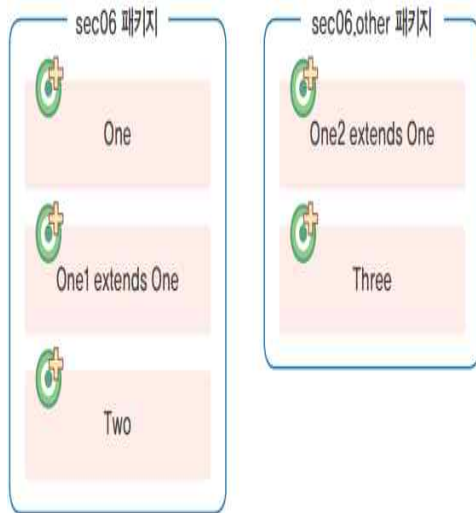
    }
}
```

[환경 변수] 대화상자 열기

설정=>시스템=>정보 => 고급시스템 설정=> 고급탭=> 환경변수 클릭

[quiz_1_식별자] 패키지 이용과 패키지 안의 패키지 생성1 (상속과 접근 제어)

■ 패키지과 접근 지정자 예



■ 예제 :

[sec06/One](#),

[sec06/One1](#),

[sec06/Two](#),

[sec06/other/One2](#),

[sec06/other/Three](#)

```
package sec06;
public class One {
    private int secret = 1;
    int roommate = 2;
    protected int child = 3;
    public int anybody = 4;
    public void show() {
    }
}
```

```
package sec06.other; //다른 패키지의 자식 객체
import sec06.One;
public class One2 extends One {
    void print() {
        // System.out.println(secret); private로
        // System.out.println(roommate); 디폴트로
        System.out.println(child);
        System.out.println(anybody);    }
}
```

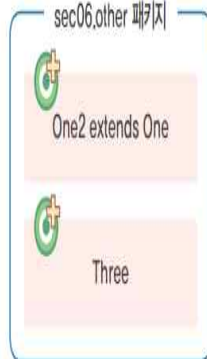
```
package sec06; //같은 패키지의 자식 객체
public class One1 extends One {
    void print() {
        // System.out.println(secret); 오류
        System.out.println(roommate);
        System.out.println(child);
        System.out.println(anybody); }
    // void show() {    }
    //오버라이딩할 때 접근 범위가 좁아지면
    //오류 발생
}
```

```
package sec06.other; //다른 패키지의 다른 클래스
import sec06.One;
public class Three {
    void print() {
        One o = new One();
        // System.out.println(o.secret); 접근 불가
        // System.out.println(o.roommate); 접근 불가
        // System.out.println(o.child); 접근 불가
        System.out.println(o.anybody);
    }
}
```

```
package sec06; //같은 패키지의 다른 클래스
public class Two {
    void print() {
        One o = new One(); //다른 멤버를 접근하기에 객체 생성 해야함
        // System.out.println(o.secret); 오류
        System.out.println(o.roommate); //같은 패키지의 다른 객체에 접근 가능
        System.out.println(o.child);
        System.out.println(o.anybody);
    }
}
```

[quiz_1_식별자] 패키지 이용과 패키지 안의 패키지 생성2 (상속과 접근 제어)

패키지와 접근 지정자 예



```
package sec06;
public class One { //생략 }
```

```
package sec06; //같은 패키지의 자식 객체
public class One1 extends One { //생략 }
```

```
package sec06; //같은 패키지의 다른 클래스
public class Two { //생략 }
```

```
package sec06.other;
import sec06.*;
public class quiz_1_1_111 {
    public static void main(String[] args) {
        Two t2 = new Two();
        t2.print();
    } //t2의 print()를 public으로 수정 해야 함
```

```
package sec06.other; //다른 패키지의 자식 객체
import sec06.One;
public class One2 extends One { //생략 }
```

```
package sec06.other; //다른 패키지의 다른 클래스
import sec06.One;
public class Three { //생략 }
```

```
package sec06;
import sec06.other.*;
public class quiz_1_2_111 {
    public static void main(String[] args) {
        Three t3 = new Three();
        t3.print();
    } //t3의 print()를 public으로 수정 해야 함
```

예제 :

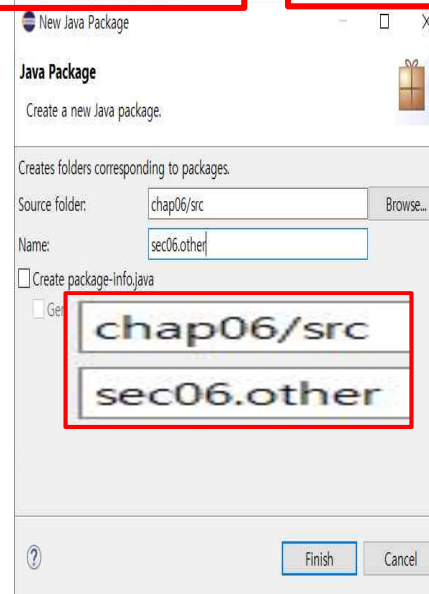
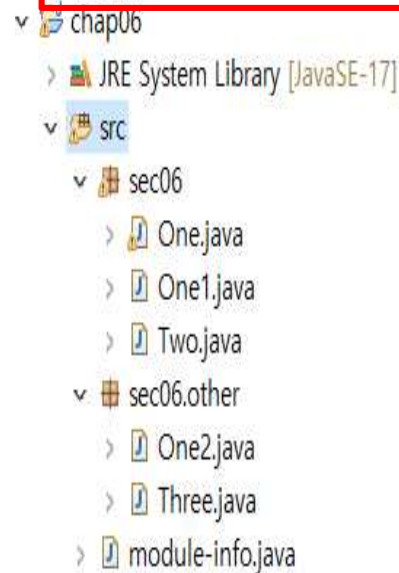
[sec06/One](#),

[sec06/One1](#),

[sec06/Two](#),

[sec06/other/One2](#),

[sec06/other/Three](#)



[quiz_2_1식별자]인터페이스 활용한 타입변환과 다형성

다음 프로그램에서 빈 곳을 채워 프로그램을 완성한 후 주석을 작성하고
Main 메서드를 인터페이스를 활용한 타입 변환과 다형성을 고려하여 중복된 메서드를 간단하게 표현하시오.

```
package sec03;
public interface Controllable {
    default void repair() { //디폴트 메서드
        show("장비를 수리한다."); }
    static void reset() { // 정적 메서드
        System.out.println("장비를 초기화한다.");
    }
    private void show(String s) {
        System.out.println(s); }
    void turnOn(); //추상 메서드, 표준화
    void turnOff(); //추상 메서드, 표준화
```

전자제품에 포함되어야 하는 제어부의 요구 조건

- 모든 전자제품에는 전원을 온·오프하는 기능이 있으며, 수리 및 공장 초기화를 할 수 있다.
- 전자제품 객체는 turnOn() 메서드, turnOff() 메서드로만 전원을 조절할 수 있어야 한다.
- 수리 및 공장 초기화 기능을 미리 구현해 놓아서 필요할 때 사용할 수 있어야 한다.
- 수리 기능은 자식 클래스에서 오버라이딩 할 수도 있다.

```
TV를 켜다.
TV를 끈다.
장비를 수리한다.
컴퓨터를 켜다.
컴퓨터를 끈다.
장비를 수리한다.
장비를 초기화한다.
```

```
package sec03;
public class TV implements Controllable {
    @Override //자식은 부모보다 접근지정자가 넓어야함!!
    public void turnOn() { //반드시 오버라이딩 구현
        System.out.println("TV를 켜다."); }
    @Override
    public void turnOff() { //반드시 오버라이딩 구현
        System.out.println("TV를 끈다."); }
}
```

```
package sec03;
public interface RemoteControllable extends Controllable {
    void remoteOn();
    void remoteOff(); }
}
```

```
package sec03;
public class Computer implements Controllable {
    @Override //자식은 부모보다 접근지정자가 넓어야함!!
    public void turnOn() { //반드시 오버라이딩 구현
        System.out.println("컴퓨터를 켜다."); }
    @Override
    public void turnOff() { //반드시 오버라이딩 구현
        System.out.println("컴퓨터를 끈다."); }
}
```

```
package sec04;
import sec03.Computer;
import sec03.Controllable;
import sec03.TV;
public class quiz_2_1_111 {
    public static void main(String[] args) {
        TV tv = new TV();
        Computer com = new Computer();
        tv.turnOn(); tv.turnOff(); tv.repair();
        com.turnOn(); com.turnOff(); com.repair();
        Controllable.reset();
        // tv.reset();
        // com.reset();
    }
}
```

[quiz_2_2식별자] 인터페이스 활용한 타입변환과 다형성

다음 프로그램에서 빈 곳을 채워 프로그램을 완성한 후 주석을 작성하고
Main 메서드를 인터페이스를 활용한 타입 변환과 다형성을 고려하여 중복된 메서드를 간단하게 표현하시오.

```
package sec03;
public interface Controllable {
    default void repair() { //디폴트 메서드
        show("장비를 수리한다."); }
    static void reset() { // 정적 메서드
        System.out.println("장비를 초기화한다.");
    }
    private void show(String s) {
        System.out.println(s);
    }
    void turnOn(); //추상 메서드, 표준화
    void turnOff(); //추상 메서드, 표준화
}
```

```
package sec03;
public class TV implements Controllable {
    @Override //자식은 부모보다 접근지정자가 넓어야함!!
    public void turnOn() { //반드시 오버라이딩 구현
        System.out.println("TV를 켜다."); }
    @Override
    public void turnOff() { //반드시 오버라이딩 구현
        System.out.println("TV를 끈다."); }
}
```

```
package sec03;
public class Computer implements Controllable {
    @Override //자식은 부모보다 접근지정자가 넓어야함!!
    public void turnOn() { //반드시 오버라이딩 구현
        System.out.println("컴퓨터를 켜다."); }
    @Override
    public void turnOff() { //반드시 오버라이딩 구현
        System.out.println("컴퓨터를 끈다."); }
}
```

```
package sec03;
public interface RemoteControllable extends Controllable {
    void remoteOn();
    void remoteOff(); }
}
```

```
package sec04;
public class quiz_2_1_111 {
    public static void main(String[] args) {
        TV tv = new TV();
        Computer com = new Computer();
        tv.turnOn(); tv.turnOff(); tv.repair();
        com.turnOn(); com.turnOff(); com.repair();
        Controllable.reset();
        // tv.reset();
        // com.reset();
    } }
}
```

```
TV를 켜다.
TV를 끈다.
장비를 수리한다.
컴퓨터를 켜다.
컴퓨터를 끈다.
장비를 수리한다.
장비를 초기화한다.
```

```
package sec04;
import sec03.Computer;
import sec03.Controllable;
import sec03.TV;
public class quiz_2_2_111 {
    public static void main(String[] args) {
        Controllable[] controllable = { new TV(), new Computer() };
        for (Controllable c : controllable) {
            c.turnOn();
            c.turnOff();
            c.repair();
        }
        Controllable.reset();
    }
}
```

타입 변환과 표준화

final 클래스와 메서드

■ final 클래스

- 더 이상 상속될 수 없는 클래스
- 대표적인 final 클래스로는 String 클래스
- 예제 : [sec07/FinalClassDemo](#)

```
package sec07;
class Good { }
class Better extends Good { }
final class Best extends Better { }
// class NumberOne extends Best { }
//=> Best클래스는 final클래스이기에 자식클래스로 확장 안됨
public class FinalClassDemo {
    public static void main(String[] args) {
        // new NumberOne();
        new Best();
    }
}
```

■ final 메서드

- final클래스는 클래스 내부의 모든 메서드를 오버라이딩할 수 없다. 특정 메서드만 오버라이딩하지 않도록 하려면 final 메서드로 선언
- 예제 : [sec07/FinalMethodDemo](#)

```
package sec07;
class Chess {
    enum ChessPlayer { WHITE, BLACK }
    final ChessPlayer getFirstPlayer() { //final 메서드
        return ChessPlayer.WHITE; }
}
class WorldChess extends Chess {
    // ChessPlayer getFirstPlayer() { } //오버라이팅 하면 오류발생
}
public class FinalMethodDemo {
    public static void main(String[] args) {
        WorldChess w = new WorldChess();
        w.getFirstPlayer();
    }
}
```


모듈화

■ 필요성

- 클래스와 패키지로 접근을 제어하는 현재의 자바로는 다음과 같은 이유로 소프트웨어를 효과적으로 개발하기 어렵다.
- **패키지의 캡슐화 기능 부족**
 - 클래스 멤버의 캡슐화는 다양하지만 패키지에 포함된 클래스는 public지정자만 사용 할 수 있다.
 - 위의 경우로 일부 패키지에게만 공개하고 싶은 경우에도 모든 패키지에 공개가 된다.
- **누락된 클래스의 탐지 어려움**=>자바는 클래스 로더를 통한 동적 클래스 로딩을 지원 하기 때문에 프로그램을 실행 하지 건에 필요한 클래스의 누락 여부를 일일이 확인하기 어렵다.
- **단일 구성 런타임 플랫폼의 비효율성**=> 런타임 파일의 크기가 점점 거대해 진다.
- 예 : 패키지의 캡슐화 기능 부족
 - 라이브러리 A는 utils와 developer이란 2개의 패키지로 구성되어 있다.
 - developer패키지는 공개하되 내부 개발자만 공개할 의도로 개발된 패키지라 가정하면
=>developer패키지의 secret클래스가 public이기에 utils에 있는 Open클래스에서 import하여 사용 가능 그러나 원하지 않게 App클래스에서도 secret클래스를 접근하는 문제점 발생

```
package programmer;
import utils.*;
import developer.*;

public class App {
    public static void main( ... ) {
        new Open().yahoo();
        X new Secret().hush();
    }
}
```

library A

```
package utils;
import developer.*;

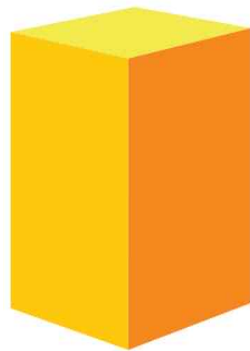
public class Open {
    public void yahoo( ) {
        O new Secret().hush();
    }
}

package developer;

public class Secret {
    public void hush( ) {
    }
}
```

모듈화

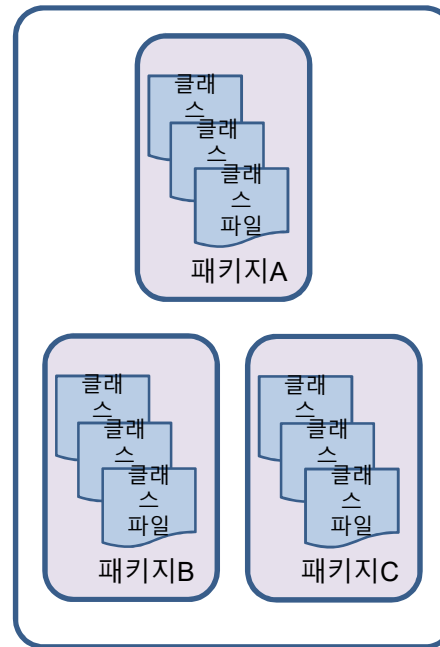
■ 모노리딕 구조와 모듈 구조



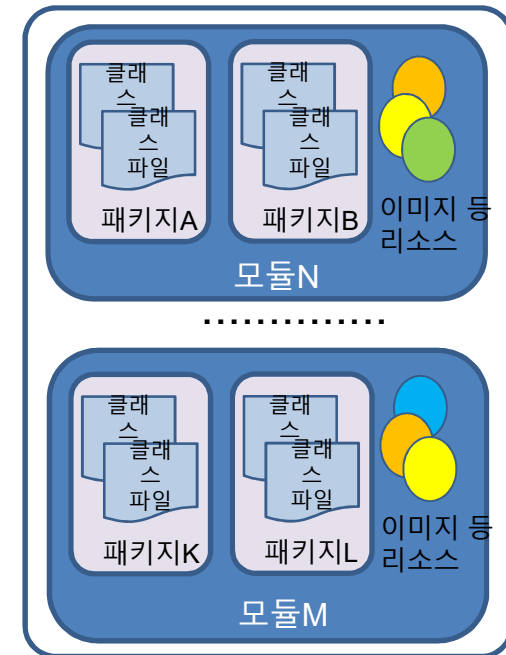
(a) 모노리딕



(b) 모듈



Java 8에서 클래스와 패키지



Java 9 이후 클래스와 패키지, 그리고 모듈

■ 자바 9에서 도입한 모듈

- Java 9에서 도입된 개념
- 모듈은 밀접한 관계가 있는 패키지, 리소스, 모듈 기술자 파일을 함께 묶어 놓은 것
=> 패키지와 이미지 등의 리소스를 담은 컨테이너
- 모듈은 JAR 파일과 달리 다른 모듈과의 의존성 정보와 패키지의 공개 여부에 대한 정보를 포함
- 특별한 경우를 제외하곤 고유한 모듈 이름을 가진다
- 모듈을 통해 패키지의 캡슐화 기능을 개선하고 누락 클래스를 실행 전에 탐지할 수 있고, 배포할 런타임의 크기를 줄일 수 있다.
- 모듈 파일(jmod)로 저장 => 모듈 파일은 JDK의 jmods 디렉터리에 저장하여 배포

자바의 모듈(module)과 패키지(package)

□ 패키지

- 서로 관련된 클래스와 인터페이스의 컴파일 된 클래스 파일들을 하나의 디렉터리에 묶어 놓은 것

□ 모듈

- 여러 패키지와 이미지 등의 자원을 모아 놓은 컨테이너
- JDK 9부터 자바 API의 모든 클래스들(자바 실행 환경)을 패키지 기반에서 모듈들로 완전히 재구성
- 응용프로그램 역시 여러 개의 모듈로 분할하여 작성 가능
 - 클래스들은 패키지로 만들고, 다시 패키지를 모듈로 만들
- 목적
 - 자바 API를 여러 모듈(99개)로 분할하여 응용프로그램의 실행에 적합한 모듈들로만 실행 환경을 구축할 수 있도록 함
 - 메모리 등의 자원이 열악한 작은 소형 기기에 꼭 필요한 모듈로 구성된 작은 크기의 실행 이미지를 만들기 위함
- 모듈의 현실
 - Java 9부터 전면적으로 도입, 복잡한 개념, 큰 자바 응용프로그램에는 개발, 유지보수 등에 적합

□ 패키지명과 클래스의 경로명

- 점(.)으로 연결
 - Project.FileIO.Tools.class
 - Project.UI.Tools.class

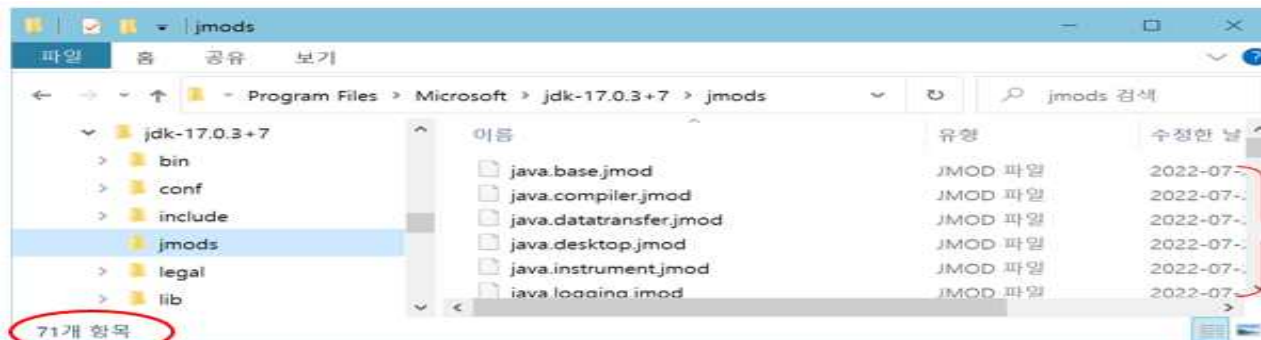
모듈화

■ 자바 런타임의 기본 모듈

- 모듈을 도입한 자바 9부터 자바 플랫폼 자체도 Jigsaw 프로젝트라는 이름으로 기능에 따라 재구성하여 모듈화되어 있으며, 다음과 같은 모듈을 포함
 - java.base 모듈
 - java.desktop 모듈
 - java.compiler 모듈
 - java.se 모듈

■ 자바 JDK에 제공되는 모듈 파일들

- 자바가 설치된 jmods 디렉터리에 모듈 파일 존재
 - 자바 버전에 따라 100개에 가까운 모듈 파일
 - 모듈 파일(jmod 파일)은 ZIP 포맷으로 압축된 파일
- 모듈 파일에는 자바 API의 패키지과 클래스들이 들어 있음
- Jmod 명령을 이용하여 모듈 파일에 들어 있는 패키지를 풀어 낼 수 있음



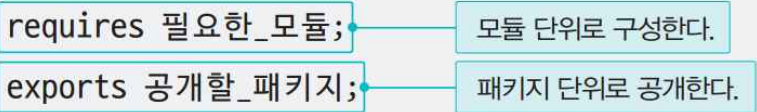
자바 API의
모듈
(모듈 파일)

모듈화

■ 모듈 작성과 응용

- 모듈 기술자는 module-info.java라는 파일 이름을 사용하며 일반적으로 패키지 최상단 수준의 폴더에 위치
- 모듈 기술자의 구성

```
module 모듈_이름 {  
    requires 필요한_모듈;  
    exports 공개할_패키지;  
}
```



The diagram shows two annotations in the code block, each with a light blue box and a line pointing to it:

- A box containing `requires` points to the `requires` keyword in the code, with the text "모듈 단위로 구성한다." (Configure at the module level).
- A box containing `exports` points to the `exports` keyword in the code, with the text "패키지 단위로 공개한다." (Expose at the package level).

모듈화

■ 예 : 모듈의 캡슐화 기능

module1

```
package programmer;
import utils.*;
// import developer.*;

public class App {
    public static void main( ... ) {
        new Open().yahoo();
        // new Secret().hush();
    }
}
```

module-info.java

```
requires module2;
```

module2

```
package utils;
import developer.*;

public class Open {
    public void yahoo( ) {
        new Secret().hush();
    }
}
```

```
package developer;

public class Secret {
    public void hush( ) {
    }
}
```

module-info.java

```
export utils;
```

8장 기본 패키지

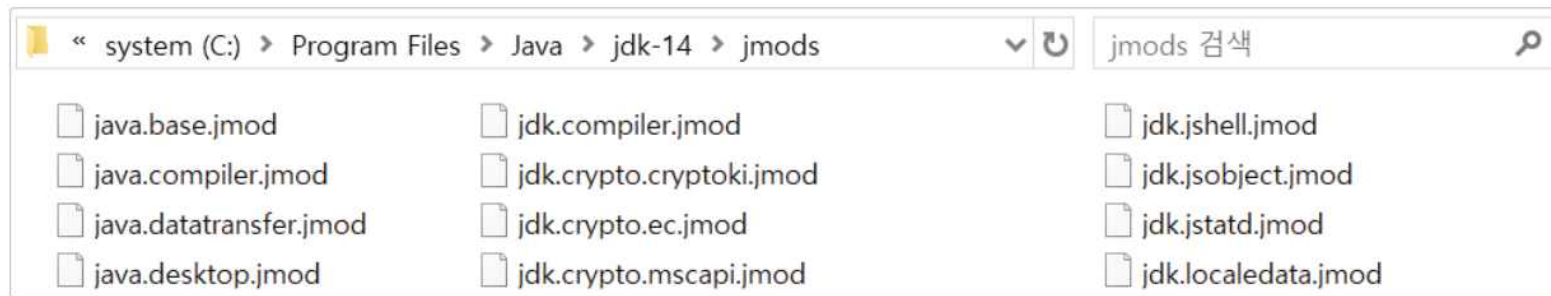
패키지와 API 문서

■ 자바 라이브러리

- 개발자가 편리하게 사용할 수 있도록 패키지 혹은 모듈을 압축한 파일

■ 패키지과 모듈

- 패키지 : 상호 관련 있는 클래스와 인터페이스를 한곳에 묶어 놓은 것
- 모듈 : 밀접한 관계가 있는 패키지와 리소스를 묶어 놓은 것. JDK를 설치하면 jmods 폴더에 jmod 파일을 제공하는데 jmod 파일이 모듈 파일



- JDK 8까지는 개발자가 편리하게 프로그래밍할 수 있도록 기본 패키지를 rt.jar 파일로 제공
- JDK 9부터는 jmod 파일을 통하여 필요한 패키지를 제공

■ 자바 모듈화의 가장 큰 목적

- 자바 컴포넌트들을 필요에 따라 조립하여 사용하기 위함
- 컴퓨터 시스템의 불필요한 부담 감소
 - 세밀한 모듈화를 통해 필요 없는 모듈이 로드되지 않게 함
 - 소형 IoT 장치에도 자바 응용프로그램이 실행되고 성능을 유지하게 함

패키지와 API 문서

■ 자바의 주요 패키지 및 모듈

패키지	설명
java.awt	그래픽을 처리하는 API
java.io	입출력을 스트림하는 API
java.lang	자바 프로그램의 필수 API
java.math	수학과 관련된 API
java.net	네트워크를 처리하는 API
java.text	날짜, 시간, 문자열 등 지역화를 처리하는 API
java.time	JDK 8이 지원하는 날짜 및 시간을 처리하는 API
java.util	날짜, 리스트, 벡터 등 유틸리티 API
javax.swing	스윙 컴포넌트 API

- java.lang
 - 자바 language 패키지
 - 스트링, 수학 함수, 입출력 등 자바 프로그래밍에 필요한 기본적인 클래스와 인터페이스
 - 자동으로 import. import 문 필요 없음
- java.util
 - 자바 유틸리티 패키지
 - 날짜, 시간, 벡터, 해시맵 등과 같은 다양한 유틸리티 클래스와 인터페이스 제공
- java.io
 - 키보드, 모니터, 프린터, 디스크 등에 입출력을 할 수 있는 클래스와 인터페이스 제공
- java.awt
 - 자바 GUI 프로그래밍을 위한 클래스와 인터페이스 제공
- javax.swing
 - 자바 GUI 프로그래밍을 위한 스윙 패키지

▼ 📁 chap08

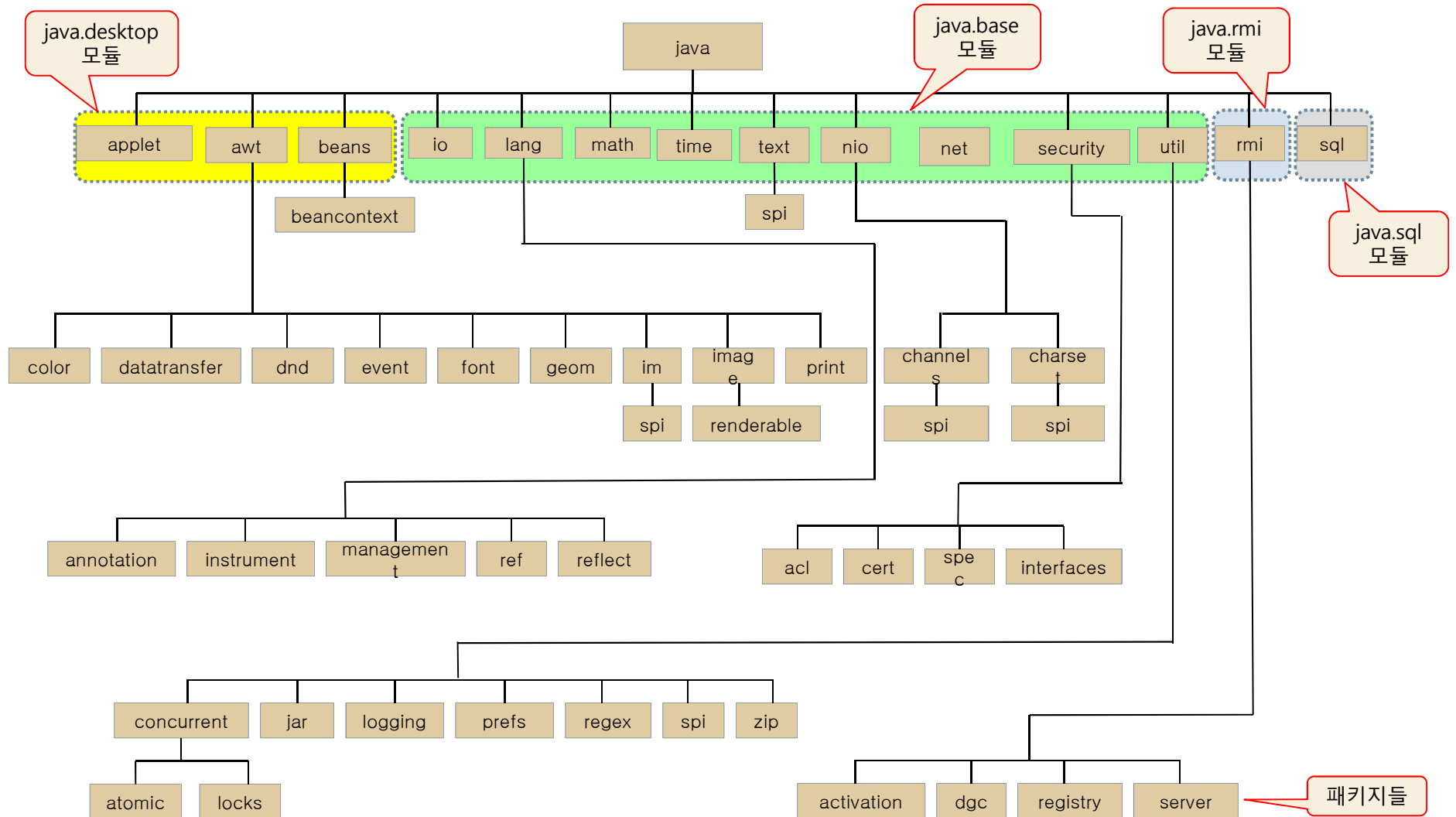
▼ 📁 JRE System Library [jdk-14]

- > 📁 java.base - C:\Program Files\Java\jdk-14\lib\jrt-fs.jar
- > 📁 java.compiler - C:\Program Files\Java\jdk-14\lib\jrt-fs.jar
- > 📁 java.datatransfer - C:\Program Files\Java\jdk-14\lib\jrt-fs.jar
- > 📁 java.desktop - C:\Program Files\Java\jdk-14\lib\jrt-fs.jar

이클립스 파일 탐색기에
나타난 다수의 모듈 파일



자바 모듈과 패키지 구조



패키지와 API 문서

■ API 문서

- <https://docs.oracle.com/en/java/javase/14/docs/api>

Java® Platform, Standard Edition & Java Development Kit
Version 14 API Specification

This document is divided into two sections:

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

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

All Modules | Java SE | JDK | Other Modules

Module	Description
java.base	Defines the foundational APIs of the Java SE Platform.
java.compiler	Defines the Language Model, Annotation Processing, and Java Compiler APIs.

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

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

This document is divided into two sections:

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

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

All Modules | Java SE | JDK | Other Modules

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

java.lang 패키지

■ 필수 패키지

- 자바 프로그램에서 가장 기본이 되는 클래스와 인터페이스를 보관
- import문 없이 사용
- **Java.lang** 패키지에 포함된 주요 클래스

클래스	설명
Class	실행 중에 클래스 정보를 제공한다.
Math	각종 수학 함수를 제공한다.
Object	최상위 클래스로 기본적인 메서드를 제공한다.
String, StringBuffer, StringBuilder	문자열을 처리하는 메서드를 제공한다.
System	시스템 정보나 입출력을 처리하는 메서드를 제공한다.
Thread	스레드를 처리하는 메서드를 제공한다.
포장 클래스	기초 타입 데이터를 객체로 처리하는 메서드를 제공한다.

java.lang 패키지

■ Object 클래스

- 모든 클래스의 조상으로 모든 클래스에 강제 상속
 - 모든 객체가 공통으로 가지는 객체의 속성을 나타내는 메소드 보유
- java.lang 패키지에 포함
- Object 클래스가 제공하는 주요 메서드

메서드	설명
public String toString()	객체의 문자 정보를 반환한다.
public boolean equals(Object o)	현재 객체와 동일한지 여부를 반환한다.
public int hashCode()	객체의 해시코드를 반환한다.
protected Object clone()	객체의 사본을 생성한다.
protected void finalize()	가비지 컬렉터가 객체를 수거할 때 호출한다.
public final Class<?> getClass()	객체 정보를 반환한다. 현 객체의 런타임 클래스를 리턴

- Object 클래스가 제공하는 toString()과 equals()메서드는 대다수 클래스에서는 거의 도움이 되지 않아 메서드 오버라이딩하여 사용한다.
- toString()메서드는 객체에서 '클래스이름@16진수 해시코드'로 구성
- equals()메서드는 두 객체의 내용이 아니라 두 객체의 동일 여부(==)를 조사한다.
- hashCode()메서드는 생성자를 통하여 새로운 인스턴스가 메모리로부터 생성될 때 그 인스턴스의 주소값을 기준으로 만들어지는 일련번호를 반환한다.
- 이외에도 final로 지정된 wait(), notify(), notifyAll() 메서드가 있음

[quiz_3_식별자] java.lang 패키지의 Object 클래스

■ Object 클래스

- 대다수 클래스는 Object 클래스가 제공하는 toString()과 equals() 메서드를 오버라이딩해서 사용
- String은 이 두 메서드를 이미 오버라이딩한 클래스임
 - equals()는 두 String객체가 가진 문자열이 같은지 아닌지에 따라 true 혹은 false를 반환으로 오버라이딩
 - toString()은 String객체의 문자열을 반환으로 오버라이딩
- 예제 : [sec02/Keyboard](#), [sec02/ObjectMethodDemo](#)



```
package sec02;
class Mouse { //???
    String name;
    public Mouse(String name) {
        ???name = name;
    }
}
class Keyboard {
    String name;
    public Keyboard(String name) {
        this.name = name;
    }
    public boolean equals(??? obj) { //???
        if (obj ??? Keyboard) {
            Keyboard k = (Keyboard) obj;
            if (name.???k.name) //???
                return true;
        }
        return false;
    }
    public String toString() { //???
        return "키보드입니다.";
    }
}
```

```
package sec02;
public class quiz_3_111 {
    public static void main(String[] args) {
        Mouse m1 = new Mouse("Logitech");
        Mouse m2 = new Mouse("Logitech");
        Mouse m3 = m1;
        Keyboard k1 = new Keyboard("Microsoft");
        Keyboard k2 = new Keyboard("Microsoft");

        System.out.println(m1.toString());
        System.out.println(m1);

        System.out.println(k1.toString());
        System.out.println(k1);

        System.out.println(m1.equals(m2));
        System.out.println(m1.equals(m3));
        System.out.println(k1.equals(k2));
    }
}
```

객체 비교와 equals()

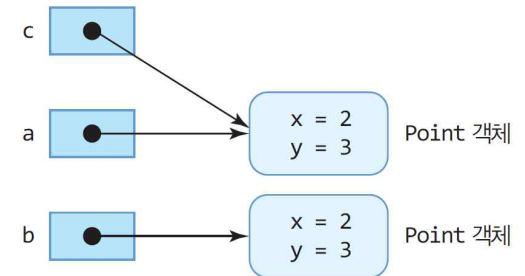
□ == 연산자

▣ 두 개의 레퍼런스 비교

```
class Point {  
    private int x, y;  
    public Point(int x, int y) {  
        this.x = x; this.y = y;  
    }  
}
```

```
Point a = new Point(2,3);  
Point b = new Point(2,3);  
Point c = a;  
if(a == b) // false  
    System.out.println("a==b");  
if(a == c) // true  
    System.out.println("a==c");
```

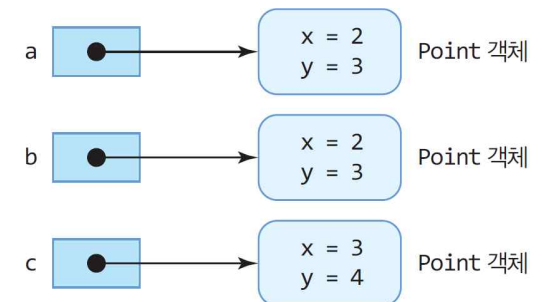
a==c



```
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;  
    }  
}
```

```
Point a = new Point(2,3);  
Point b = new Point(2,3);  
Point c = new Point(3,4);  
  
if(a == b) // false  
    System.out.println("a==b");  
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");
```

a is equal to b

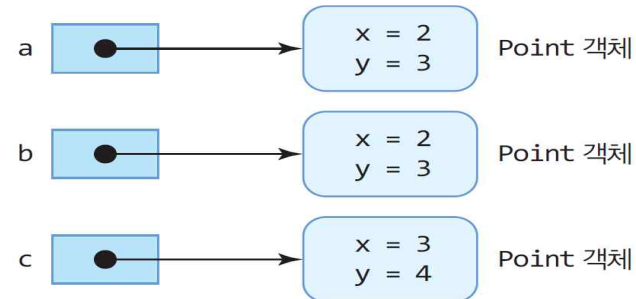


[quiz_4_식별자] Point 클래스에 equals() 작성

Point 클래스에 두 점의 좌표가 같으면 true를 리턴하는 equals()를 작성하시오.

```
package w9_111;
class Point {
    private int x, y;
    public Point(int x, int y) {
        this.x = x; this.y = y;
    }
    public boolean equals(Object obj) {
        // 여러 문장
    }
}
```

a is equal to b



```
public class quiz_4_111 {
    public static void main(String[] args) {
        Point a = new Point(2,3);
        Point b = new Point(2,3);
        Point c = new Point(3,4);

        if(a == b) // ???
            System.out.println("a==b");
        if(a.equals(b)) // ???
            System.out.println("a is equal to b");
        if(a.equals(c)) // ???
            System.out.println("a is equal to c");
    }
}
```

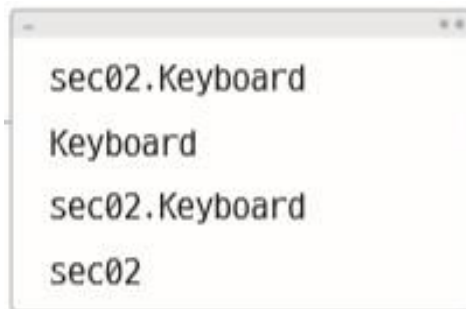

java.lang 패키지

■ Class 클래스

- 실행 중인 자바 프로그램 내부에 포함된 클래스와 인터페이스 정보를 제공하려고 getName(), getSimpleName() 등 다양한 메서드를 제공
- Class 클래스는 생성자가 없고 어떤 객체라도 생성하면 JVM이 대응하는 Class 객체를 자동으로 생성
- 실행 도중 객체 정보를 얻으려면 getClass()의 결과인 Class객체를 사용
- 예를 들어 주어진 obj객체에서 Class 객체를 얻으려면



- 예제 : [sec02/ClassDemo](#)



```
package sec02;

public class ClassDemo {
    public static void main(String[] args) {
        Keyboard k = new Keyboard("Logitech");

        Class c = k.getClass();

        System.out.println(c.getName());

        System.out.println(c.getSimpleName());

        System.out.println(c.getTypeName());

        System.out.println(c.getPackage().getName());
    }
}
```

java.lang 패키지

■ Math 클래스

- 모든 메서드가 static이기 때문에 객체를 생성하지 않고 메서드 호출
- Math 클래스가 제공하는 주요 double 타입의 메서드

메서드	반환 값
static double abs(double a)	실수 a의 절댓값
static double cos(double a)	실수 a의 cosine 값
static double exp(double a)	e^a 값
static double log(double a)	실수 a에 대한 자연 로그 값
static double log10(double a)	실수 a에 대한 10의 로그 값
static double max(double a, double b)	실수 a와 b 중 큰 값
static double min(double a, double b)	실수 a와 b 중 작은 값
static double pow(double a, double b)	a^b 값
static double random()	0.0 이상 1.0 미만의 난수
static double sin(double a)	실수 a의 sine 값
static double sqrt(double a)	실수 a의 제곱근 값
static double tan(double a)	실수 a의 tangent 값

- 예제 : [sec02/MathDemo](#)

```
Math.pow(2, 8) : 256.0
Math.random() : 0.27418513672415135
Math.sin(Math.PI) : 1.2246467991473532E-16
Math.min(10, 20) : 10
```

```
package sec02;
public class MathDemo {
    public static void main(String[] args) {
        System.out.println("Math.pow(2, 8) : " + Math.pow(2, 8));
        System.out.println("Math.random() : " + Math.random());

        System.out.println("Math.sin(Math.PI) : " + Math.sin(Math.PI));
        System.out.println("Math.min(10, 20) : " + Math.min(10, 20));
    }
}
```

java.lang 패키지

■ String 클래스

- 상수 문자열이기 때문에 **String 객체에 포함된 문자열을 수정 불가.**
- String 객체의 문자열을 수정하는 것은 내부적으로는 수정된 문자열을 포함하는 새로운 String 객체를 생성
- 따라서 문자열 내용을 자주 변경한다면 String 클래스를 사용하는 것은 좋지 않다.

■ StringBuffer 및 StringBuilder 클래스

- 자바는 변경될 수 있는 문자열을 다룰 수 있도록 **StringBuilder와 StringBuffer 클래스를 제공**
- 다중 스레드 환경에서 StringBuffer 클래스가 안전하다는 점을 제외하면 StringBuffer 클래스는 **StringBuilder 클래스와 거의 유사**
- **다중 스레드 환경이 아니라면 StringBuilder 클래스를 사용하는 것이 효율적**

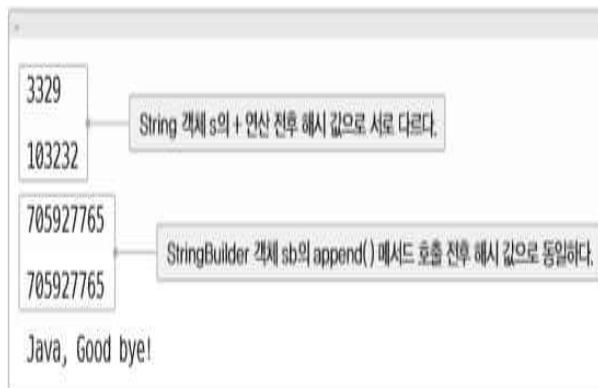
[quiz_5_식별자] java.lang 패키지의 String 클래스

■ StringBuilder 클래스가 제공하는 주요 메서드

메서드	설명
StringBuilder append(String s)	문자열 s를 버퍼에 덧붙인다.
int capacity()	현재 버퍼의 크기를 반환한다.
StringBuilder delete(int start, int end)	문자열의 일부분을 버퍼에서 제거한다.
StringBuilder insert(int offset, String s)	문자열 s를 버퍼의 offset 위치에 삽입한다.
StringBuilder replace(int start, int end, String s)	문자열의 일부분을 문자열 s로 대체한다.
StringBuilder reverse()	버퍼에 있는 문자열을 반대 순서로 변경한다.

● 예제 : [sec02/StringBuilderDemo](#)

- String과 StringBuilder 클래스의 차이
=> hashCode(해시 값) 비교



```
package sec02;
public class quiz_5_111 {
    public static void main(String[] args) {
        ??? s = new String("hi");
        System.out.println(s.???);
        s = s + "!";
        System.out.println(s. ???);

        ??? sb = new StringBuilder("hi");
        System.out.println(sb.???);
        sb = sb.append("!");
        System.out.println(sb.???);

        System.out.println(sb.replace(0, 2, "Good bye").insert(0, "Java, "));
    }
}
```

java.lang 패키지

■ System 클래스

- 표준 입출력을 비롯한 실행 시스템과 관련된 필드와 메서드를 static으로 제공
- System.out.println()도 System 클래스가 제공하는 메서드
- System클래스의 세 가지 필드

필드	설명
static InputStream in	표준 입력 스트림이다.
static PrintStream out	표준 출력 스트림이다.
static PrintStream err	표준 오류 출력 스트림이다.

java.lang 패키지

■ System 클래스

- System클래스가 제공하는 주요 메서드

메서드	설명
<code>static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)</code>	주어진 위치에서 주어진 길이만큼 배열 src를 배열 dest로 복사한다.
<code>static long currentTimeMillis()</code>	현재 시각을 밀리초 단위로 반환한다.
<code>static void exit()</code>	현재 실행 중인 JVM을 종료한다.
<code>static void gc()</code>	가비지 컬렉터를 실행한다.
<code>static String getenv(String name)</code>	지정된 환경 변수 값을 반환한다.
<code>static String getProperty(String key)</code>	주어진 key 값에 해당하는 시스템 특성을 반환한다.
<code>static long nanoTime()</code>	현재 시각을 나노초 단위로 반환한다.

- 예제 : [sec02/SystemDemo](#)

```
100 200 300 3 4 5 6
1457507710788
C:\Program Files\Java\jdk1.8.0_131
Windows 10
113787788160112
```

```
package sec02;
public class SystemDemo {
    public static void main(String[] args) {
        int[] src = new int[] { 1, 2, 3, 4, 5, 6 };
        int[] dst = { 100, 200, 300, 400, 500, 600, 700 };

        System.arraycopy(src, 2, dst, 3, 4);
        for (int i = 0; i < dst.length; i++) {
            System.out.print(dst[i] + " ");
        }
        System.out.println();

        System.out.println(System.currentTimeMillis());
        System.out.println(System.getenv("JAVA_HOME"));

        System.out.println(System.getProperty("os.name"));
        System.out.println(System.nanoTime());
    }
}
```

java.lang 패키지

■ System 클래스

- 자바에서는 운영체제로부터 할당 받은 메모리를 JVM이 관리
- JVM은 메모리가 부족하거나 주기적으로 가비지 컬렉터를 사용해 가비지를 수거
- 가비지를 수거하는 순서는 객체의 생성 순서와는 무관
- 프로그램에서 가비지 컬렉터를 직접 호출 불가. 대신에 `System.gc()`로 JVM에 가능하면 빨리 가비지 컬렉터를 실행하도록 요청 가능

- 예제 : [sec02/GarbageDemo](#)



```
package sec02;
class Garbage {
    public int no;
    public Garbage(int no) {
        this.no = no;
        System.out.printf("Garbage(%d) 생성\n", no);
    }
    protected void finalize() {
        System.out.printf("Garbage(%d) 수거\n", no);
    }
}

public class GarbageDemo {
    public static void main(String[] args) {
        for (int i = 0; i < 3; i++)
            new Garbage(i);

        System.gc();
    }
}
```

java.lang 패키지

■ 포장 클래스

- 대부분의 기본 패키지가 제공하는 클래스의 메서드는 참조 타입을 매개변수로 사용하기 때문에 기초 타입을 사용하면 객체 지향 언어의 특징(캡슐화, 상속, 다형성)을 이용 불가
- 자바는 기초 타입을 포장해 클래스화 한 포장 클래스(wrapper class)를 제공해서 기초 타입 데이터도 기본 패키지를 활용토록 함
- 기본 자료형을 참조형 자료형처럼 사용하기 위한 클래스이다.
- 기본 자료형에 대해서 객체로서 인식되도록 '포장'을 했다는 뜻입니다.
- 자바의 기본 타입을 클래스화한 8개 클래스

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

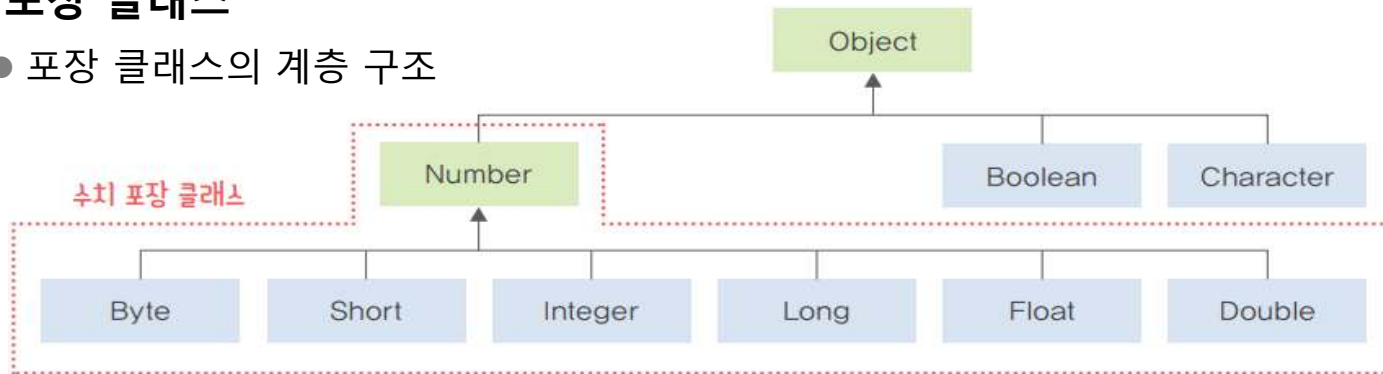
- 이름이 Wrapper인 클래스는 존재하지 않음
- 용도
 - 기본 타입의 값을 객체로 다룰 수 있게 함



java.lang 패키지

■ 포장 클래스

- 포장 클래스의 계층 구조



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

포장 클래스	생성자
Byte	Byte(byte value), Byte(String s)
Short	Short(short value), Short(String s)
Integer	Integer(int value), Integer(String s)
Long	Long(long value), Long(String s)
Float	Float(double value), Float(float value), Float(String s)
Double	Double(double value), Double(String s)
Character	Character(char value)
Boolean	Boolean(boolean value), Boolean(String s)

java.lang 패키지

■ 포장 클래스

- Integer 클래스가 제공하는 주요 메서드

메서드	설명
int intValue()	int 타입으로 반환한다.
double doubleValue()	double 타입으로 반환한다.
float floatValue()	float 타입으로 반환한다.
static int parseInt(String s)	문자열을 int 타입으로 반환한다.
static String toBinaryString(int i)	int 타입을 2진수 문자열로 반환한다.
static String toHexString(int i)	int 타입을 16진수 문자열로 반환한다.
String toString(int i)	int 타입을 10진수 문자열로 반환한다.
static Integer valueOf(String s)	문자열을 Integer 객체로 반환한다.
static Integer valueOf(String s, int radix)	radix 진수의 문자열을 Integer 객체로 반환한다.

java.lang 패키지

■ 포장 클래스

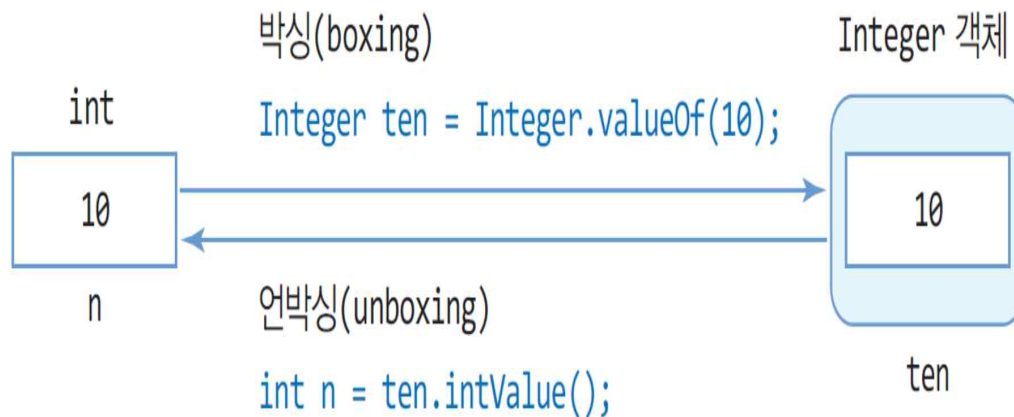
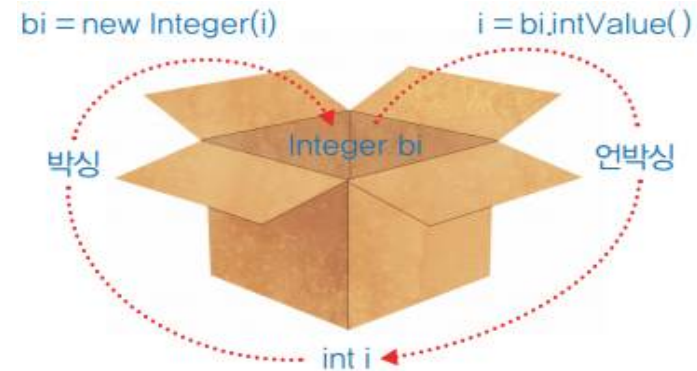
● 박싱과 언박싱

□ 박싱(boxing)

- 기본 타입의 값을 Wrapper 객체로 변환
- 기초 타입 데이터를 포장해 객체화 하는 것
- 박싱을 수행하려면 포장 클래스의 생성자나 `valueOf()` 메서드를 사용

□ 언박싱(unboxing)

- Wrapper 객체에 들어 있는 기본 타입의 값을 빼내는 것



java.lang 패키지

■ 포장 클래스

● 박싱과 언박싱

□ 박싱(boxing)

- 기본 타입의 값을 Wrapper 객체로 변환
- 기초 타입 데이터를 포장해 객체화 하는 것
- 박싱을 수행하려면 포장 클래스의 생성자나 `valueOf()` 메서드를 사용

□ 언박싱(unboxing)

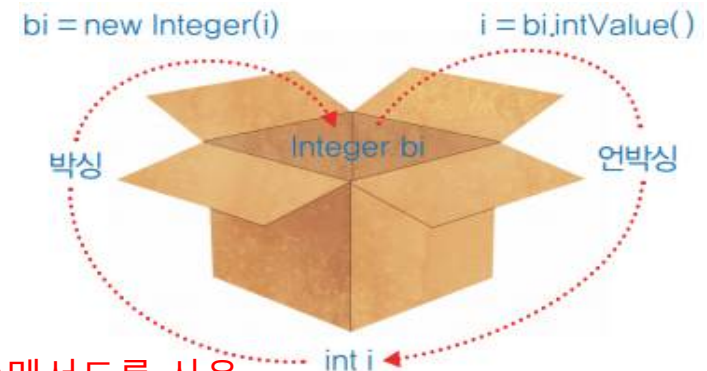
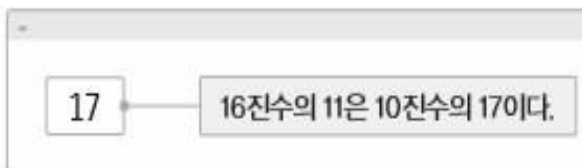
- Wrapper 객체에 들어 있는 기본 타입의 값을 빼내는 것

```
Integer bi = new Integer(10);  
Integer bi = Integer.valueOf(10);
```

```
Integer bi = new Integer(10);  
int i = bi.intValue();  
double d = bi.doubleValue();
```

```
Integer bi = 10;      // 자동 박싱  
int i1 = bi;          // 자동 언박싱  
int i2 = bi + 20;     // 자동 언박싱
```

● 예제 : [sec02/WrapperDemo](#)



```
package sec02;  
public class WrapperDemo {  
    public static void main(String[] args) {  
        Integer bi1 = new Integer(10);  
        int i1 = bi1.intValue();  
        double d = bi1.doubleValue();  
        Integer bi2 = 20;  
        int i2 = bi2 + 20;  
        String s1 = Double.toString(3.14);  
        Float pi = Float.parseFloat("3.14");  
        Integer bi3 = Integer.valueOf("11", 16);  
        System.out.println(bi3);  
    }  
}
```

java.util 패키지

- 날짜, 시간, 리스트, 벡터, 해시 테이블, 컬렉션 등 다양한 유틸리티 클래스와 인터페이스를 제공
- java.util 패키지가 제공하는 주요 클래스

클래스	설명
Arrays	배열을 비교, 복사, 정렬 등 조작할 때 사용한다.
Calendar	날짜와 시간 정보가 필요할 때 사용한다.
Date	밀리초 단위의 현재 시각이 필요할 때 사용한다.
StringTokenizer	특정 문자로 구분된 문자열을 뽑아낼 때 사용한다.
Random	난수가 필요할 때 사용한다.

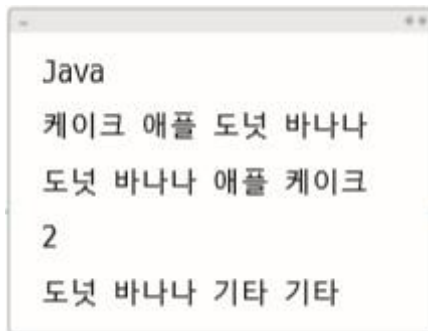
java.util 패키지

■ Arrays 클래스

- Arrays 클래스가 제공하는 주요 정적 메서드

메서드	설명
List asList(배열)	배열을 리스트로 변환한다.
int binarySearch(배열, 키)	배열에서 키 값이 있는 인덱스를 반환한다.
배열 copyOf(배열, 길이)	원본 배열을 길이만큼 복사한다.
배열 copyOfRange(배열, 시작, 끝)	원본 배열을 지정한 영역만큼 복사한다.
boolean equals(배열, 배열)	두 배열의 동일 여부를 비교한다.
void fill(배열, 값)	배열을 지정된 값으로 저장한다.
void fill(배열, 시작, 끝, 값)	배열의 지정된 영역에 지정된 값을 저장한다.
void sort(배열)	배열을 오름차순으로 정렬한다.

- 예제 : [sec03/ArraysDemo](#)



```
package sec03;

import java.util.Arrays;

public class ArraysDemo {
    public static void main(String[] args) {
        char[] a1 = { 'J', 'a', 'v', 'a' };
        char[] a2 = Arrays.copyOf(a1, a1.length);
        System.out.println(a2);

        String[] sa = { "케이크", "애플", "도넛", "바나나" };
        print(sa);
        Arrays.sort(sa);
        print(sa);

        System.out.println(Arrays.binarySearch(sa, "애플"));

        Arrays.fill(sa, 2, 4, "기타");
        print(sa);
    }
    static void print(Object[] oa) {
        for (Object o : oa)
            System.out.print(o + " ");
        System.out.println();
    }
}
```

java.util 패키지

■ Date 클래스

- Date클래스는 현재 날짜와 시각 정보를 제공.
- 국제화에 맞지 않아 대부분의 메서드는 현재 폐기 중
- 주로 하위 호환성이나 간단한 날짜 정보를 원할 때만 사용

■ Calendar 클래스

- 지역이나 문화에 따라 달력을 표시하는 방식이 다르기 때문에 추상 클래스로 되어 있음
- 표준 달력을 사용한다면 다음과 같이 객체 생성

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

java.util 패키지

■ Calendar 클래스

- Calendar 클래스가 제공하는 정수 타입의 상수

필드 이름	의미
AM, AM_PM, PM	오전 및 오후
DATE	날짜
JANUARY, FEBRUARY, ...	1월, 2월 등
SUNDAY, MONDAY, ...	일요일, 월요일 등
MINUTE	분
HOUR	시간(0~11)
HOUR_OF_DAY	시간(0~23)
MONTH	월(0~11)
DAY_OF_MONTH	한 달 내에서의 날짜
WEEK_OF_YEAR	일 년 내에서의 몇 주차
YEAR	연도

- MONTH는 0~11사이의 정수

java.util 패키지

■ Calendar 클래스

- Calendar 클래스가 제공하는 주요 메서드

메서드	설명
boolean after(Object when)	주어진 시간보다 뒤쪽이면 true를 반환한다.
boolean before(Object when)	주어진 시간보다 앞쪽이면 true를 반환한다.
void clear(int field)	지정된 필드를 미정의 상태로 변경한다.
int compareTo(Calendar anotherCalendar)	2개의 Calendar 객체를 비교한다.
int get(int fields)	주어진 필드 값을 반환한다.
int getFirstDayOfWeek()	첫 날이 무슨 요일인지 반환한다.
Date getTime()	Calendar 객체를 Date 객체로 변환한다.
void set(int field, int value)	주어진 필드를 주어진 값으로 변경한다.
void set(int year, int month, int date)	연, 월, 일 값을 변경한다.
void setTime(Date date)	Date 객체로 Calendar 객체를 설정한다.

java.util 패키지

■ Calendar 클래스

- 예제 : [sec03/CalendarDemo](#)



```
Wed Jun 07 17:54:32 KST 2017
java.util.GregorianCalendar[time=1...
2017
6
7
4
23
2
5
17
54
```

```
package sec03;
import java.util.Calendar;
import java.util.Date;
public class CalendarDemo {
    public static void main(String[] args) {
        Date now = new Date();
        System.out.println(now);
        Calendar c = Calendar.getInstance();
        System.out.println(c);

        System.out.println(c.get(Calendar.YEAR));

        System.out.println(c.get(Calendar.MONTH) + 1);

        System.out.println(c.get(Calendar.DAY_OF_MONTH));

        System.out.println(c.get(Calendar.DAY_OF_WEEK));

        System.out.println(c.get(Calendar.WEEK_OF_YEAR));

        System.out.println(c.get(Calendar.WEEK_OF_MONTH));

        System.out.println(c.get(Calendar.HOUR));

        System.out.println(c.get(Calendar.HOUR_OF_DAY));
        System.out.println(c.get(Calendar.MINUTE));
    }
}
```

java.util 패키지

■ StringTokenizer 클래스

- 문자열을 토큰으로 분리하는 데 사용
- 토큰은 공백이나 줄 바꿈 등 구분자를 사용해 문자열을 분리
- StringTokenizer 클래스의 주요 생성자

생성자	설명
StringTokenizer(String s)	주어진 문자열을 기본 구분자로 파싱한 StringTokenizer 객체를 생성한다.
StringTokenizer(String s, String delim)	주어진 문자열을 delim 구분자로 파싱한 StringTokenizer 객체를 생성한다.

- 기본 구분자는 공백, 탭, 줄 바꿈, 복귀, 용지 먹임 문자

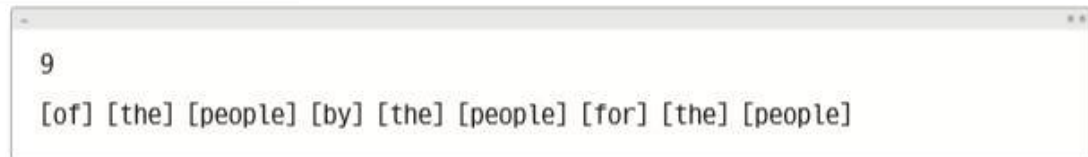
java.util 패키지

■ StringTokenizer 클래스

- StringTokenizer 클래스가 제공하는 주요 메서드

메서드	설명
int countTokens()	남아 있는 토큰의 개수를 반환한다.
boolean hasMoreTokens()	남아 있는 토큰이 있는지 여부를 반환한다.
String nextToken()	다음 토큰을 꺼내 온다.

- 예제 : [sec03/StringTokenizerDemo](#)



```
9
[of] [the] [people] [by] [the] [people] [for] [the] [people]
```

```
package sec03;
import java.util.StringTokenizer;

public class StringTokenizerDemo {
    public static void main(String[] args) {
        String s = "of the people, by the people, for the people";
        StringTokenizer st = new StringTokenizer(s, " ,");

        System.out.println(st.countTokens());

        while (st.hasMoreTokens()) {
            System.out.print "[" + st.nextToken() + " ] ";
        }
    }
}
```

java.util 패키지

■ Random 클래스

생성자	설명
Random()	Random 객체를 생성한다.
Random(long seed)	주어진 시드를 사용하는 Random 객체를 생성한다.

● 예제 : [sec03/RandomDemo](#)



메서드	설명
boolean nextBoolean()	논리 타입 난수를 발생시킨다.
double nextDouble()	0.0 이상 1.0 미만의 double 타입 난수를 발생시킨다.
float nextFloat()	0.0 이상 1.0 미만의 float 타입 난수를 발생시킨다.
double nextGaussian()	평균, 표준편차가 0.0 및 1.0인 정규분포 난수를 발생시킨다.
int nextInt()	int 타입의 난수를 발생시킨다.
int nextInt(int n)	0~(n-1) 사이의 int 타입 난수를 발생시킨다.
long nextLong()	long 타입의 난수를 발생시킨다.
void setSeed(long seed)	시드 값을 설정한다.

```
package sec03;
import java.util.Random;

public class RandomDemo {
    public static void main(String[] args) {
        Random r = new Random();

        for (int i = 0; i < 5; i++)
            System.out.print(r.nextInt(100) + " ");
    }
}
```