패키지

- 관련된 파일을 폴더를 통해 관리하듯이 서로 밀접한 관련을 가진 클래스, 인터페이스등을 모아놓기 위해서 패키지를 사용할 수 있음
- 패키지는 실제로 디렉토리이며 **비슷한 성격의 자바 클래스들을 모아 놓은 자바의 디렉토리**임
- 패키지를 사용하는 이유
  - 클래스를 체계적으로 관리하고 구분 짓기 위해서
- 같은 패키지에 동일한 이름의 클래스, 인터페이스는 존재할 수 없음
  - 그런데 코드를 작성하다 보면 다른 사람이 작성한 라이브러리를 사용해야 하는 경우도 많은데, 이 때 자바 클래스나 인터페이스의 이름이 겹쳐서 동일해도 **패키지명이 다르면 충돌없이 사용할 수 있음**

```
// 서로 클래스 이름이 동일해도 패키지가 다르면 상관 없음
kr.hs.emirim.MyClass
kr.hs.emirim.another.MyClass
```

패키지 선언 방법 (패키지를 선언할 클래스, 인터페이스들은 당연히 같은 패키지 이름 내 폴더에 생성되어 있어야 함)

```
package 패키지이름;
```

실제 사례

```
// kr.hs.emirim.MyClass.java
package kr.hs.emirim;

class MyClass {}
```

- 이름 없는 패키지(unnamed package, aka 디폴트 패키지)
  - 특정 패키지에 속하지 않는 클래스, 인터페이스들은 모두 이름 없는 패키지에 속하게 됨 (=src 폴더에 생성된 클래스, 인터페이스)
  - 예제로 사용할 간단한 프로젝트가 아닌 이상 **가급적 하나 이상의 패키지를 가지도록 프로젝트 구조를 만드는 것이 권장**됨
- 패키지 이름과 관련된 컨벤션이 존재함
  - 보통 도메인 이름을 뒤집은 것으로 정함 (e-mirim.hs.kr => kr.hs.emirim)
  - 이름은 전부 소문자로 구성
  - 숫자로 시작하거나 언더스코어, 달러표시(_, $)를 제외한 특수 문자 사용 금지
  - 예약어(int, static, for 등) 사용 금지

import 구문 사용

- 같은 패키지에 속하는 클래스들은 바로 접근 가능히지만, 다른 패키지에 속하는 클래스를 사용하려면 두 가지 방법 중 하나를 선택해야 함
- **import 구문**을 이용해서 다른 패키지에 포함된 클래스, 인터페이스를 불러올 수 있음

import 구문

```
// 해당 패키지에 속한 특정 클래스나 인터페이스만 불러오기 위해서 사용
import 패키지이름.클래스이름;
// 해당 패키지에 속한 모든 클래스와 인터페이스를 불러오기 위해서 사용
import 패키지이름.*;
```

- import 문을 선언할 때 별표(*)를 사용하는 것이 해당 패키지에 포함된 다른 모든 하위 패키지의 클래스까지 포함해 주는 것은 아님

```
import java.awt.*;
import java.util.*;

// 위와 같은 두 개의 import 문을 아래와 같이 하나의 import 문으로 표현할 수는 없음
// (밑의 명령어는 java 패키지 내부의 모든 클래스와 인터페이스만 불러옴)
import java.*;
```

- 자바에서는 가장 많이 사용되는 **java.lang 패키지**에 대해서는 import 문을 사용하지 않아도 클래스 이름만으로 사용이 가능 (ex: String, Integer, Object 등등)
- 다른 이름의 패키지에서 각각 가져온 같은 이름의 클래스를 동시에 쓰는 경우에는 **FQCN(Fully Qualified Class Name)**을 기술
    - FQCN(Fully Qualified Class Name) => 패키지 이름을 클래스 이름과 같이 쓰는 것
    - 단, 이러한 일이 실제로 발생할 일은 거의 없음

```
// 다른 패키지의 String 클래스를 import
import kr.hs.emirim.String;

public class Main {
    // 해당 String은 kr.hs.emirim 패키지 내부의 String 클래스
    String s1 = new String();
    // 기존 문자열 클래스와 이름이 겹치므로 FQCN을 이용해서 모두 패키지와 클래스명 모두 적어주기
    java.lang.String s2 = new java.lang.String("Hello");
}
```

서브패키지

- 패키지 내부에 다른 패키지를 포함할 수 있음
    - 실제로는 디렉토리 내부의 하위 디렉토리를 생성하여 패키지를 만듬
    - 디렉터리의 계층 구조는 **점(.)으로 구분**됨
- 서브 패키지를 통해서 패키지를 계층화하여 관리할 수 있음

패키지의 계층화

```
hellotodo
    .database
        .repository
        .entity
    .util
        .commons
        .format
    .gui
    .network
```

```
// hello 패키지 내부의 world 하위 패키지 존재
// 폴더구조는 hello/world
hello.world

// kr 패키지 내부의 hs 하위 패키지 내부의 emirim 하위 패키지 존재
// 폴더구조는 kr/hs/emirim
kr.hs.emirim
```

```
// hello 패키지 내부의 MyClass 클래스 불러오기
import hello.MyClass;
// hello 패키지의 하위 패키지인 world의 MyClass 클래스 불러오기
import hello.world.MyClass;
```

- 다음의 패키지 구조 설명해보기
    - Q1) 똑같은 이름(MyClass)의 클래스가 3개 있는데 괜찮은가?
    - Q2) 각각의 클래스의 FQCN은?