● **패키지(package): 클래스의 묶음. 클래스 또는 인터페이스를 포함**시킬 수 있으며, 서로 관련된 클래스들끼리 그룹 단위로 묶어 놓음으로써 클래스를 효율적으로 관리할 수 있다. **클래스의 실제 이름(full name)은 패키지명을 포함한 것**이기 때문에, **같은 이름의 클래스일지라도 서로 다른 패키지에 속하면 패키지명으로 구별이 가능**하다. 클래스가 물리적으로 하나의 클래스파일(.class)인 것과 같이 패키지는 물리적으로 **하나의 디렉토리(폴더)**이다(클래스는 클래스파일이고, 패키지는 폴더이고, 파위 패키지는 하위 폴더이다). 그래서 어떤 패키지에 속한 패키지는 해당 디렉토리에 존재하는 클래스 파일(.class)이어야 한다. 또한 패키지도 다든 패키지를 포함할 수 있으며 점'.'으로 구분된다. 예를 들면 java.lang.String클래스는 물리적으로 디렉토리 java 패키지의 서브디렉토리(하위 패키지)인 lang패키지에 속한 String.class파일이다.

- 하나의 소스파일에는 첫 번째 문장으로 단 한 번의 패키지 선언만은 허용한다. 선언하는 방법은 클래스나 인터페이스의 소스파일(.java)의 맨 위에**(주석과 공백은 제외한 첫번째 문장)** 다음과 같이 적으면 된다. **package 패키지명; 패키지명은 소문자로 한다** (대문자인 클래스명과 구분짓는다). 같은 소스 파일의 클래스들은 모두 같은 패키지에 속하게 된다.
- 모든 클래스는 반드시 하나의 패키지에 속해 있어야 하며, 소스파일에 패키지를 선언해주지 않으면 자바에서 기본적으로 제공하는 이름없는 패키지'unnamed package'(default package)에 클래스가 속하게 된다. 결국 패키지를 지정하지 않는 모든 클래스들은 같은 패키지에 속하게 된다.
- 패키지는 점(.)을 구분자로 하여 계층구조로 구성할 수 있다.
- 패키지는 물리적으로 클래스 파일(.class)을 포 함하는 하나의 디렉토리다. 참고로 클래스 파일들을 압축한 것이 jar파일(.jar)인데, 이 파일은 jar.exe이나 알집이나 winzip으로 압축을 풀 수 있다. 예를 들면 String클래스(java.lang.String)는 rt.jar파일에 압축되어 있으며, 이 파일의 압축을 풀면 rt > java > lnag > String.class가 나타난다.

> 컴파일
> 예를 들면 ,
> package com.codechobo.book;
> class PacakgeTest {
> public static void main(String[] args) {
> System.out.println("Hello World!");
> }
> }
> 이처럼 작성한 뒤 컴파일하려면 '-d'옵션을 추가하여 컴파일한다.
> C:₩jdk1.8₩work>javac -d . PackageTest.java
> '-d'옵션은 소스파일에 지정된 경로를 통해 패키지의 위치를 찾아서 클래스파일을 생성하는데, 만일 지정된 패키지와 일치하는 디렉토리가 존재하지 않으면 자동적으로 생성한다. '-d' 옵션 뒤에는 해당 패키지의 루트(root)디렉토리의 경로를 적어준다. (.)은 현재 디렉토리, 즉, C:₩jdk1.8₩work으로 지정했기 때문에, 컴파일을 수행하고 나면 다음과 같은 구조로 디렉토리가 완성된다.
> jdk1.8 > work > com > codechobo > book > PackageTest.class
> 이제는 패키지 루트 디렉토리를 클래스패스(claspath)에 포함시켜야 한다. 클래스패스에 등록하면 더이상 루트 디렉토리를 매번 입력하는 수고를 하지 않아도 되기 때문이다. com.code.chobo.book패키지의 루트 디렉토리는 C:₩jdk1.8₩work이고, 이 디렉토리를 클래스패스에에 포함시켜야만 실행 시 JVM이 PackageTest클래스를 찾을 수 있다. 클래스패스(Classpath)는 컴파일러(javac.exe)나 JVM이 클래스의 위치를 찾는데 사되는 경로이다. 윈도우즈에서의 클래스패스 설정 방법은 제어판-시스템-고급시스템설정-환경변수-새로만들기에서 변수 이름이 CLASSPATH를 입력하고 변수 값에는 '.;c:₩jdk1.8₩work'를 입력하면 된다. ';'를 구분자로 하여 여러 개의 경로를 클래스패스에 지정할 수 있으며, 맨 앞에 .;를 추가한 이유는 현재 디렉토리(.)를 클래스패스에 포함시키기 위해서이다. 클래스패스를 지정해주지 않으면 기본적으로 현재 디렉토리(.)가 클래스패스로 지정되지만, 이처럼 클래스패스를 따로 지정해주는 경우에는 더이상 현재 디렉토리가 자동적으로 클래스패스로 지정되지 않기 때문에 이처럼 별도로 추가를 해주어야 한다.
> jar파일을 클래스패스에 추가하기 위해서는 경로와 파일 명을 적어주어야 한다. 예를 들면 'C:₩jdk1.8₩work₩util.jar'파일을 클래스 패스에 포함시키려면, C:₩WINDOWS>SET CLASSPATH = .;C:₩jdk1.8₩work;C:₩jdk1.8₩work₩util.jar; 로 적어주어야 한다.
> 클래스패스가 바르게 설정되어있는지 확인하는 방법은 아래와 같은 명령어를 입력해 보면 된다.
> C:₩WINDOWS>echo %classpath%
> .;C:₩jdk1.8₩work; 현재 디렉토리를 의미하는 '.'와 'C:₩jdk1.8₩work'가 클래스패스로 잘 지정되어있음을 알 수 있다.
> 이제 PacakgeTest예제를 실행시켜 보면,
> C:₩WINDOWS>java com.codechobo.book.PackageTest
> Hello Word!라고 잘 뜨는 것을 알 수 있다.

> 실행시에는 이처럼 PackageTest클래스의 패키지명을 모두 적어주어야 한다.
> JDK에 기본적으로 설정되어 있는 클래스패스를 이용하면 위의 예제에서와 같이 클래스패스를 따로 설정하지 않아도 된다. 새로 추가하고자 하는 클래스를 'JDK설치디렉토리₩jre₩classes'디렉토리에, jar파일인 경우에는 'JDK설치디렉토리₩jre₩lib₩ext'디렉토리에 넣기만 하면 된다. (참고로 jre디렉토리 아래의 classes디렉토리는 JDK 설치 시에 자동적으로 생성되지 않으므로 사용자가 직접 생성해야 한다). 또는 실행 시에 '-cp'옵션을 이용해서 일시적으로 클래스패스를 지정해 줄 수도 있다.
> C:₩WINDOWS>java -cp c:₩jdk1.8₩work com.codechobo.book.PackageTest

● import 문: import문으로 사용하고자 하는 클래스의 패키지를 미리 명시해주면 소스코드에 사용되는 클래스이름에서 패키지명을 생략할 수 있다. 원래는 다든 패키지의 클래스를 사용하려면 패키지명이 포함된 클래스 이름을 사용해야 하지만, import문으로 패키지명을 생략할 수 있는 것이다 (단, 같은 패키지 내의 클래스들은 import문을 지정하지 않고도 패키지명을 생략할 수 있다). import문의 역할은 컴파일러에게 소스파일에 사용된 클래스의 패키지에 대한 정보를 제공하는 것이다. 컴파일 시에 컴파일러는 import문을 통해 소스파일에 사용된 클래스들의 패키지를 알아낸 다음, 모든 클래스 이름 앞에 패키지명을 붙여 준다.

▶ 선언문
- import문은 package문 다음에, 그리고 클래스 선언문 이전에 위치해야 한다. import문은 한 소스파일에 여러 번 선언할 수 있다. 일반적인 소스파일(.java)의 구성은 다음의 순서로 되어 있다: 1. package문 2.import문 3. 클래스 선언
- import문의 선언 방법:

```
import 패키지명.클래스명;
또는
import 패키지명.*;   // 지정된 패키지에 속한 모든 클래스를 패키지명 없이 사용할 수 있다.
// 단, import하는 패키지의 수가 많을 때는 어느 클래스가 어느 패키지에 속하는지 구별하기가
// 어렵다는 단점이 있다.
// 또한 import문에서 클래스 이름 대신 '*'을 사용하는 것이 하위 패키지의 클래스까지 포함하는
// 것은 아니다. 예를 들면 java.util.*와 java.text.*을 java.*과 같이 할 수는 없다.
```

▶ import java.lang.* : 모든 소스파일에 묵시적으로 선언되어 있는 import문. System과 String 같은 java.lang의 패키지들의 패키지명 없이 사용할 수 있었던 이유다. 매우 빈번하게 사용되는 클래스들이 속한 패키지이기 때문에 따로 import문으로 지정하지 않아도 되도록 한 것이다.

▶ static import문: static멤버를 호출할 때 클래스 이름을 생략할 수 있다. 특정 클래스의 static멤버를 자주 사용할 때 편리하다.

```
import static java.lang.Integer.*;    // Integer클래스의 모든 static멤버(static변수와메서드
import static java.lang.Math.random;   // Math.random()만.
import static java.lag.System.out;     // system.out을 out으로만 참조 가능

// System.out.println(Math.random()); 이 코드를 아래와 같이 간략하게 할 수 있다.
out.println(random());
```

```java
import static java.lang.System.out;
import static java.lang.Math.*;  // Math클래스의 모든 static멤버(상수, 변수, 메서드 포함)

class StaticImportEx1 {
    public static void main(String[] args) {
        // System.out.println(Math.random());
        out.println(random());

        // System.out.println("Math.PI :"+Math.PI);
        out.println("Math.PI :" + PI);
    }
}
```

Resources: 자바의 정석