패키지와 라이브러리

#01. 패키지

클래스를 폴더 형태로 분류한 상태.

`src` 폴더 안에 하위 폴더를 만들고 그 안에 소스파일을 배치한다.

`src 폴더의 하위 폴더`를 패키지라고 한다.

1) 패키지 구성하기

어떤 패키지에 포함된 클래스는 소스코드 첫 라인에서 자신이 어떤 패키지에 속해 있는지 명시해야 한다.

```
package kr.hossam;

public class HelloWorld {

}
```

패키지 이름이 `kr.hossam`인 경우 아래와 같이 폴더구조가 생성되어 있다는 의미

```
프로젝트
└src
    └kr
        └hossam
```

2) 다른 패키지에 포함된 클래스에 대한 객체 생성

전체 경로 명시

다른 패키지의 클래스로부터 객체를 생성해야 할 경우 해당 패키지의 전체 경로를 명시해야 한다.

다소 코드가 길어진다.

```
kr.hossam.HelloWorld hello = new kr.hossam.HelloWorld();
```

`import`구문의 사용

클래스 정의 전 다른 패키지안의 클래스를 참조하겠다는 선언문을 명시하면 클래스 이름만으로 객체 생성이 가능하다.

```

```

```
// kr/hossam 폴더 안에 있는 HelloWorld라는 클래스를 참조한다는 것을 의미
import kr.hossam.HelloWorld;

public class Foo {
  ...
  HelloWorld h = new HelloWorld();
  ...
}
```

> Scanner를 사용할 때 1라인에서 명시하던 import문은 `java/util` 폴더 안에 있는 Scanner 클래스를 참조한다는 것을 의미. Java언어 안에는 개발자가 참조할 수 있는 수 많은 클래스들이 package 형태로 내장되어 있다.

3) 20단원의 예제 재구성

파일 재배치

```
프로젝트
    └src
        ├ App.java
        └kr
            └hossam
                ├ Protoss.java
                │
                └unit
                    ├ Dragun.java
                    └ Zilot.java
```

패키지 구문 및 import문 확인

소스파일에 package 선언문과 import문이 정확하지 않게 추가되는 경우도 있기 때문에 반드시 한번은 직접 확인해야 한다.

kr.hossam.Protoss.java

```
package kr.hossam;

public abstract class Protoss {
    ... 생략 ...
}
```

kr.hossam.unit.Dragun.java

```java
package kr.hossam.unit;

import kr.hossam.Protoss;

public class Dragun extends Protoss {
    ... 생략 ...
}
```

kr.hossam.unit.Zilot.java

```java
package kr.hossam.unit;

import kr.hossam.Protoss;

public class Zilot extends Protoss {
    ... 생략 ...
}
```

App.java

```java
import kr.hossam.Protoss;
import kr.hossam.unit.Dragun;
import kr.hossam.unit.Zilot;

public class App {
    ...
}
```

#02. 라이브러리

컴파일이 완료된 클래스들을 배포를 목적으로 패키지 단위로 그룹지어 압축한 형태.

완성한 기능을 소스파일의 노출 없이 기능 단위로만 배포할 수 있다.

자바의 라이브러리는 `*.jar` 확장자를 갖는다.

1) 라이브러리 생성 방법

`Ctrl + Shift + P` -> `Java: Export Jar` 명령 선택. --> `<without main class>` 선택

일반적으로 라이브러리는 다른 프로그램이 활용할 수 있는 재료의 목적으로 생성하기 때문에 main 클래스를 포함하지 않는 형태로 압축한다.

2) jar 파일 활용하기

새로운 프로젝트 생성 : `25_1_라이브러리활용`

앞서 생성한 라이브러리를 새로운 프로젝트의 `lib` 폴더에 넣는다.

main 클래스에서 라이브러리 안의 기능을 활용한다.

```
import kr.hossam.Protoss;
import kr.hossam.unit.Dragun;
import kr.hossam.unit.Zilot;

public class App {
    public static void main(String[] args) throws Exception {
        // 추상 클래스는 직접적으로 객체 할당 불가 --> `new` 예약어 사용 불가
        // Protoss p = new Protoss("프로브1", 50, 30, 10);

        Zilot z = new Zilot("질럿1호", 150, 130, 100);
        z.move("저그 멀티");
        z.attack("저그 멀티");
        System.out.println("------------");

        Dragun d = new Dragun("드라군1호", 200, 150, 100);
        d.move("저그 멀티");
        d.attack("저그 멀티");
        System.out.println("------------");

        /** 추상 클래스는 선언은 가능하지만 할당은 불가능 */
        // 드라군과 질럿을 프로토스 타입으로 할당
        /**
        Protoss p1 = new Dragun("드라군2호", 200, 150, 100);
        Protoss p2 = new Zilot("질럿1호", 150, 130, 100);
        /*/
        Protoss p1 = new Zilot("질럿1호", 150, 130, 100);
        Protoss p2 = new Dragun("드라군2호", 200, 150, 100);
        /**/

        // 추상 클래스를 상속받은 자식 클래스들은 부모의 추상 메서드를 반드시 재정의
        // --> 부모가 정의하고 있는 메서드를 포함하고 있다는 것이 보장된다.
        // p1과 p2는 할당되는 구현체 클래스의 종류에 따라 같은 메서드를 호출하더라도 다
른 결과를 만들어 낸다.
        // --> 다형성
        p1.move("저그 멀티");
        p1.attack("저그 멀티");
        p2.move("저그 멀티");
        p2.attack("저그 멀티");
    }
}
```