**패키지**

2013-12-17 18:21:22

**패키지**



이전 시간에 클래스 패스를 배웠다. 클래스 패스란 컴퓨터의 저장장치 어딘가에 존재하는 클래스 파일을 사용하기 위한 방법이다. 이번 시간에 살펴볼 패키지(Package)는 하나의 클래스 안에서 같은 이름의 클래스들을 사용하기 위한 방법이라고 할 수 있다.

비유를 해보자. 서로 다른 내용의 파일 java.txt가 하나의 컴퓨터에 동시에 공존할 수는 없다. 그래서 고안된 것이 디렉토리다. java.txt 파일을 각각 a와 b라는 디렉터리에 저장한다면 하나의 컴퓨터 안에 같은 이름의 파일이 공존할 수 있게 된다. 누군가에게 'a 디렉터리에 있는 java.txt'를 이메일로 보내달라고 요청할 수 있게 되는 것이다.

패키지도 이와 유사하다. 클래스가 많아짐에d 따라서 같은 이름을 가진 클래스가 생겨날 가능성이 높아지게 되는데 이름의 충돌을 방지하기 위한 고안된 것이 패키지라고 할 수 있다.

정보 공학에서는 '이름의 충돌'이라는 문제를 해결하기 위해서 다양한 노력을 하고 있다. 전역변수와 지역변수, 객체도 그런 연장선에 있다고 볼 수 있다.

**패키지 만들기**

우리는 이미 패키지를 사용해왔다. 일부 IDE들은 패키지의 사용을 강제하고 있기 때문에 독자가 패키지가 무엇인지도 숙지하지 않은 상태에서 패키지를 사용한 것이다. 주제에서 어긋나는 이야기이지만 지식이라는 것이 사실은 순차적이지 않고 네트워크적이라는 점을 생각해볼 필요가 있다. 순차적인 것은 순서가 있다는 의미다. 즉 먼저 배워야 할 것이 있고 나중에 배워야 할 것이 있다는 의미다. 우리 수업도 그런 식으로 되어 있다. 수업이 그렇다보니 배우는 입장에서는 지식을 순차적으로만 받아들이게 된다.  
  
하지만 패키지의 예를 통해서도 엿볼 수 있듯이 처음부터 사용되지만, 나중에 설명할 수밖에 없는 것이 있다. 지금 패키지를 설명하고 있지만, 패키지의 어떤 측면은 아직도 설명할 수 없는 것이 있다. 그리고 이러한 부분이 독자 입장에서는 고통스러울 것이다. 필자는 배우는 입장에서 중요한 능력이 두 가지 있다고 생각한다. 하나는 모르는 것을 알고자 하는 호기심이다. 다른 하나는 모르는 것을 견디는 인내심이다. 이 두 가지 상반되는 것 같은 미덕이 조화롭게 공존 할 때 지식은 받아들여지는 것 같다. 처음에는 온갖 개념들이 머릿속에서 춤을 출 것이다. 하지만 차츰 끊어져 있던 개념들이 거미줄처럼 연결되기 시작하면서 개념들의 존재감은 조용히 사라질 것이다. 그러다 어떤 맥락을 만났을 때 거짓말처럼 뛰쳐나와서 춤을 추기 시작한다.  
  
출처는 기억나지 않지만, 인상적으로 간직하고 있는 문구가 있다.  
  
"배우고 익히고 잊어버려라"



이미 사용해본 패키지들을 살펴보자.

[토픽 "클래스와 인스턴스 그리고 객체"의 예제](http://opentutorials.org/module/516/5400#package)에는 아래와 같은 구문이 있다.

[?](https://www.opentutorials.org/course/1194/5531)

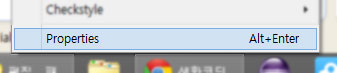
|  |  |
| --- | --- |
| 1 | package org.opentutorials.javatutorials.object; |

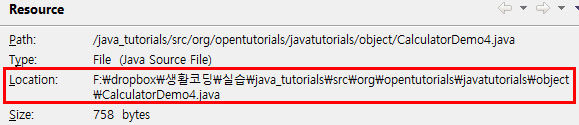
[또 토픽 "클래스 맴버와 인스턴스 맴버"의 예제](http://opentutorials.org/module/516/5440#example1)에는 아래와 같은 구문이 있다.

[?](https://www.opentutorials.org/course/1194/5531)

|  |  |
| --- | --- |
| 1 | package org.opentutorials.javatutorials.classninstance; |

그럼 각 클래스들의 위치를 찾아보자. 이클립스에서는 파일을 선택하고 오른쪽 클릭을 하면 메뉴 하단에 properties 항목을 선택하면 아래와 같은 대화상자가 나타날 것이다.





Location이 소스코드가 위치하는 경로다. 필자의 경로는 아래와 같다.

[?](https://www.opentutorials.org/course/1194/5531)

|  |  |
| --- | --- |
| 1 | F:\dropbox\생활코딩\실습\java\_tutorials\src\org\opentutorials\javatutorials\object\CalculatorDemo4.java |

경로를 분석해보자. 다음은 프로젝트가 위치하는 경로다.

[?](https://www.opentutorials.org/course/1194/5531)

|  |  |
| --- | --- |
| 1 | F:\dropbox\생활코딩\실습\java\_tutorials |

다음은 이 프로젝트의 소스코드가 위치하는 경로다. 이 경로는 이클립스가 지정한 것이다.

[?](https://www.opentutorials.org/course/1194/5531)

|  |  |
| --- | --- |
| 1 | \src |

다음 경로가 우리 수업의 주인공이다.

[?](https://www.opentutorials.org/course/1194/5531)

|  |  |
| --- | --- |
| 1 | \org\opentutorials\javatutorials\object\ |

위의 경로는 패키지의 이름과 일치한다.

[?](https://www.opentutorials.org/course/1194/5531)

|  |  |
| --- | --- |
| 1 | package org.opentutorials.javatutorials.object; |

패키지는 기본적으로 디렉터리와 일치한다. 그렇기 때문에 아래의 패키지들은 물리적으로 같은 디렉터리에 존재할 수 없다.

* org.opentutorials.javatutorials.object
* org.opentutorials.javatutorials.classninstance

그럼 패키지는 실제로 어떻게 쓰이는가를 알아보자.

아래 코드를 보자. 아래 코드의 파일명은 A.java이다. 패키지명은 일반적으로 클래스를 제작한 개인이나 단체가 소속된 웹사이트의 도메인을 이용한다. 패키지의 이름도 중복될 수 있는데 웹사이트의 도메인 전세계에서 유일무일한 식별자이기 때문에 이러한 중복의 문제를 피할 수 있다.

[?](https://www.opentutorials.org/course/1194/5531)

|  |  |
| --- | --- |
| 1  2 | package org.opentutorials.javatutorials.packages.example1;  public class A {} |

아래 코드는 위에서 정의한 클래스 A를 클래스 B에서 사용하는 예제다. 정상적으로 동작한다.

[?](https://www.opentutorials.org/course/1194/5531)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | package org.opentutorials.javatutorials.packages.example1;  public class B {  public static void main(String[] args) {  A a = new A();  }  } |

이번에는 패키지를 바꿔보자.

[?](https://www.opentutorials.org/course/1194/5531)

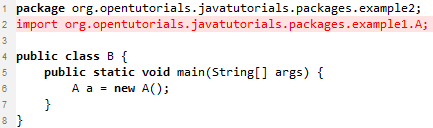
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | package org.opentutorials.javatutorials.packages.example2;  public class B {  public static void main(String[] args) {  //클래스 A가 다른 패키지에 있기 때문에 로드 할 수 없다.  A a = new A();  }  } |

위의 코드는 동작하지 않는다. 주석으로 처리한 A a = new A(); 부분에서 에러가 발생하기 때문이다. 그 이유는 여기서 사용하려는 클래스 A와 B가 서로 다른 패키지에 소속되어 있기 때문이다. 아래와 같이 코드를 고쳐서 이 문제를 해결할 수 있다.

[?](https://www.opentutorials.org/course/1194/5531)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | package org.opentutorials.javatutorials.packages.example2;  import org.opentutorials.javatutorials.packages.example1.A;  public class B {  public static void main(String[] args) {  A a = new A();  }  } |

차이점은 아래와 같다.

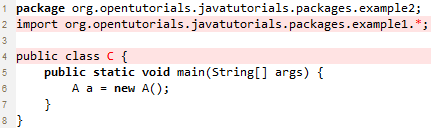


서로 다른 패키지에 있는 클래스를 가져오려면 import를 통해서 다른 패키지의 클래스를 현재의 소스코드로 불러와야 한다. 만약 특정 패키지에 있는 모든 클래스를 로드하고 싶다면 아래와 같이 하면 된다.

[?](https://www.opentutorials.org/course/1194/5531)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | package org.opentutorials.javatutorials.packages.example2;  import org.opentutorials.javatutorials.packages.example1.\*;  public class C {  public static void main(String[] args) {  A a = new A();  }  } |

차이점은 아래와 같다. \*는 '모든'이라는 뜻이다.



이렇게 해서 패키지가 무엇인가에 대해서 알아봤다. 그럼 이클립스 없이 패키지를 사용하는 방법을 알아보자.

**손 컴파일**



자 손으로 컴파일을 해보자. 개발도구 없이 코딩하는 경우는 거의 없다. 그러므로 혹시 이해가 안 되면 나중에 다시 봐도 되는 부분이다. 하지만 언젠가는 알고 있어야 하는 부분이다. 이클립스 내부에서 어떤 일이 일어나는지 궁금하지 않은가?

프로젝트 디렉터리의 구성을 살펴보자.

* src : 소스 코드가 들어있다.
* bin : 컴파일된 클래스 파일이 들어있다.

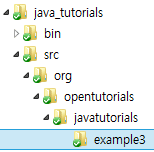
위와 같이 구분한 이유는 관리의 편의성을 위해서다. 그럼 src에 소스코드를 만들고 그것을 컴파일 한 결과를 bin 하위에 위치하도록 작업해보자.

우선 우리가 컴파일 하려는 클래스는 아래와 같은 패키지의 소속이다.

[?](https://www.opentutorials.org/course/1194/5531)

|  |  |
| --- | --- |
| 1 | package org.opentutorials.javatutorials.packages.example3; |

패키지는 디렉터리와 대응관계에 있다. 따라서 패키지의 구조대로 src 하위에 디렉터리를 만들어보자.



packages 디렉터리에 파일 Selfcompile.java를 만든다. 파일의 내용은 아래와 같다.

[?](https://www.opentutorials.org/course/1194/5531)

|  |  |
| --- | --- |
| 1  2 | package org.opentutorials.javatutorials.packages.example3;  public class Selfcompile{} |

이제 컴파일을 해보자. 컴파일은 아래와 같이 프로젝트 디렉토리에서 진행하자.

https://s3.ap-northeast-2.amazonaws.com/opentutorials-user-file/module/516/2008.png

컴파일을 하려면 콘솔을 실행시켜야 한다. 콘솔을 프로젝트 디렉토리로 이동한다. 필자의 경우 아래와 같다.

https://s3.ap-northeast-2.amazonaws.com/opentutorials-user-file/module/516/2009.png

이제 컴파일을 하자.

[?](https://www.opentutorials.org/course/1194/5531)

|  |  |
| --- | --- |
| 1 | javac src/org/opentutorials/javatutorials/packages/example3/\*.java |

위의 명령은 현재 디렉토리를 기준으로 src/org/opentutorials/javatutorials/packages/example3/ 하위에 있는 모든 자바 파일을 컴파일한다. 컴파일 한 결과는 src/org/opentutorials/javatutorials/packages/example3/ 에 저장된다. 특별한 옵션을 주지 않으면 소스코드와 클래스 파일이 동일한 디렉토리에 위치하게 된다.

우리가 원하는 것은 클래스 파일이 bin 하위에 위치하도록 하는 것이다. 아래와 같이 컴파일 명령을 바꾸면 된다.

[?](https://www.opentutorials.org/course/1194/5531)

|  |  |
| --- | --- |
| 1 | javac src/org/opentutorials/javatutorials/packages/example3/\*.java -d bin |

-d bin은 컴파일된 결과를 bin 디렉토리 하위에 위치시킨다는 의미다. 자바 컴파일러는 자동으로 클래스의 패키지에 해당하는 디렉토리를 생성해준다.

**중복의 회피**



만약 import 한 패키지 안에 같은 이름의 클래스가 존재하고 이 클래스를 사용하고 싶다면 어떤 문제가 발생할까? 아래 코드는 import 하고 있는 두개의 패키지에 클래스 B가 존재하는 경우에 어떤 일이 발생하는가를 보여준다.

[?](https://www.opentutorials.org/course/1194/5531)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | package org.opentutorials.javatutorials.packages.example3;  import org.opentutorials.javatutorials.packages.example1.\*;  import org.opentutorials.javatutorials.packages.example2.\*;  public class D {  public static void main(String[] args) {  B b = new B();  }  } |

위의 코드는 아래와 같은 오류를 발생한다.

[?](https://www.opentutorials.org/course/1194/5531)

|  |  |
| --- | --- |
| 1  2  3  4  5 | Exception in thread "main" java.lang.Error: Unresolved compilation problems:  The type B is ambiguous  The type B is ambiguous  at org.opentutorials.javatutorials.packages.example3.D.main(D.java:8) |

클래스 B의 이름이 중복되기 때문에 애매함(ambiguous)의 문제가 발생한다. 아래와 같은 방법으로 이 문제를 우회할 수 있다.

[?](https://www.opentutorials.org/course/1194/5531)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | package org.opentutorials.javatutorials.packages.example3;  import org.opentutorials.javatutorials.packages.example1.\*;  import org.opentutorials.javatutorials.packages.example2.\*;  public class D {  public static void main(String[] args) {  org.opentutorials.javatutorials.packages.example2.B b = new org.opentutorials.javatutorials.packages.example2.B();  }  } |

**수업을 마치며**

이로써 패키지가 무엇인지, 또 복잡한 컴파일을 하는 방법 등에 대해서 알아봤다. 클래스 패스나 패키지는 자바에서는 거대담론에 속하는 주제다. 로직들을 관리하는 가장 큰 틀의 체계들인 셈이다. 이제 거대 주제는 여기까지 알아보고 다시 객체 지향에 대해서 알아보자.