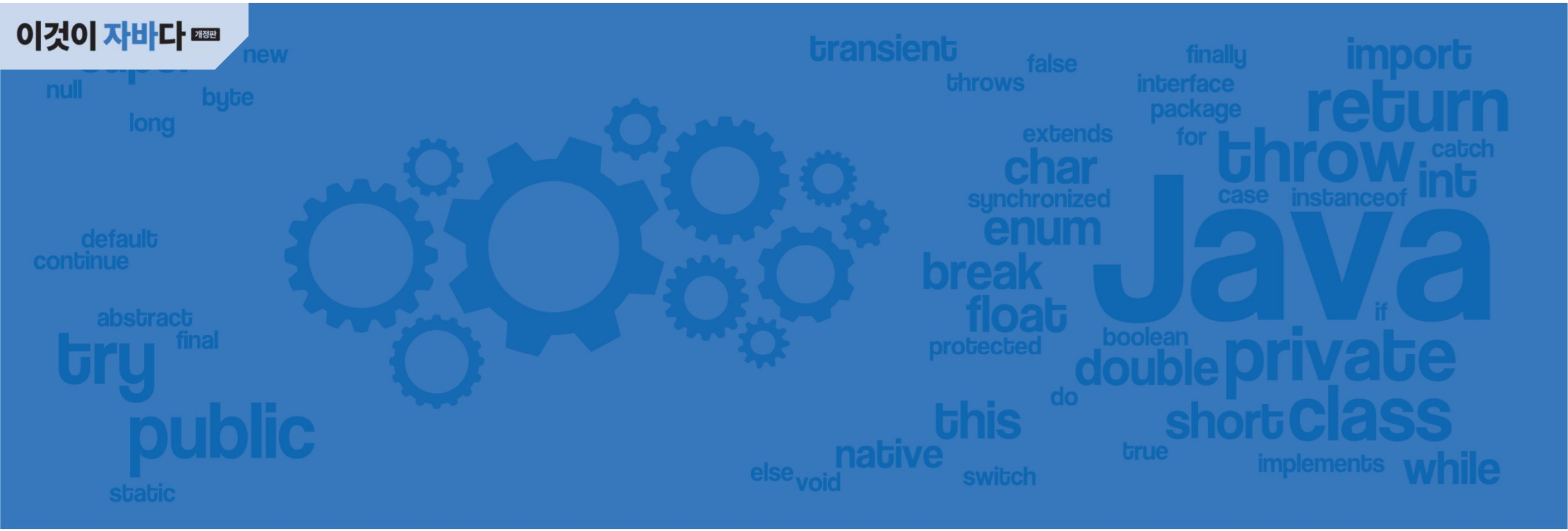


Chapter 10 라이브러리와 모듈

Contents

10.1 라이브러리

10.2 모듈

10.3 응용프로그램 모듈화

10.4 모듈 배포용 JAR 파일

10.5 패키지 은닉

10.6 전이 의존

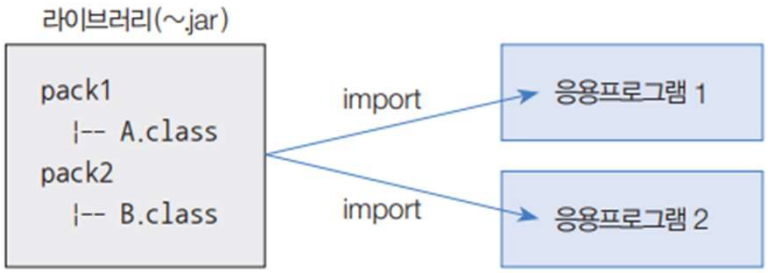
10.7 집합 모듈

10.8 리플렉션 허용

10.9 자바 표준 모듈

라이브러리 추가하기

- 프로그램 개발 시 활용할 수 있는 클래스와 인터페이스들을 모아놓은 것
- 일반적으로 JAR 압축 파일(~.jar) 형태. 클래스와 인터페이스의 바이트코드 파일(~.class)들이 압축



- 라이브러리 JAR 파일을 사용하려면 ClassPath(클래스를 찾기 위한 경로)에 추가
- 콘솔(명령 프롬프트 또는 터미널)에서 프로그램을 실행할 경우: java 명령어를 실행할 때 -classpath로 제공. 또는 CLASSPATH 환경 변수에 경로 추가
- 이클립스 프로젝트에서 실행할 경우: 프로젝트의 Build Path에 추가

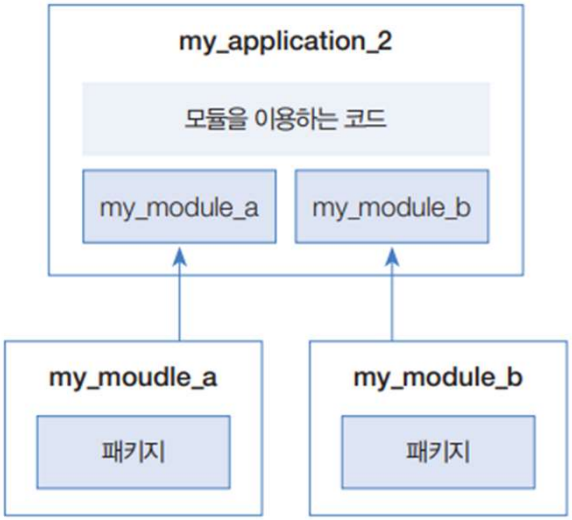
모듈

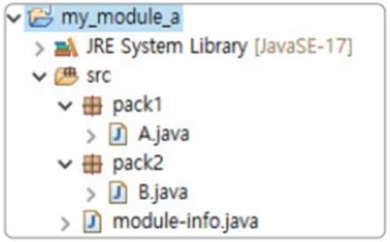
- 패키지 관리 기능까지 포함된 라이브러리. Java 9부터 지원
- 모듈은 일부 패키지를 은닉하여 접근할 수 없게끔 할 수 있음

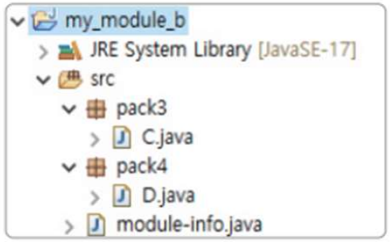
- 의존 모듈을 모듈 기술자(module-info.java)에 기술할 수 있어 모듈 간 의존 관계를 파악하기 쉬움
- 대규모 응용프로그램은 기능별로 모듈화해서 개발. 재사용성 및 유지보수에 유리

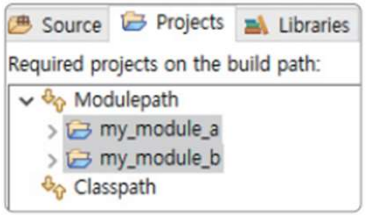
모듈화

- 모듈화: 응용프로그램을 기능별로 서브 프로젝트(모듈)로 쪼갠 다음 조합해서 개발









- 응용프로그램의 규모가 클수록 협업과 유지보수 측면에서 모듈화 유리
- 다른 응용프로그램서도 재사용 가능

모듈 배포용 JAR 파일 생성

- 다른 모듈에서 쉽게 사용할 수 있게 바이트코드 파일(.class)로 구성된 배포용 JAR 파일을 모듈별로 따로 생성할 수 있음

>>> Main.java

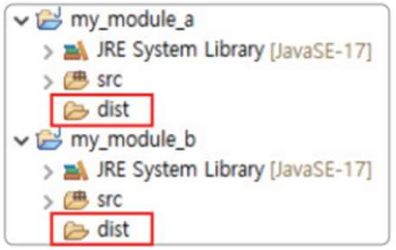
```
1   package app;
2
3   import pack1.A;
4   import pack2.B;
5   import pack3.C;
6
7   public class Main {
8     public static void main(String[] args) {
9       //my_module_a 패키지에 포함된 A 클래스 이용
10      A a = new A();
11      a.method();
12
13      //my_module_a 패키지에 포함된 B 클래스 이용
14      B b = new B();
15      b.method();
16
17      //my_module_b 패키지에 포함된 C 클래스 이용
18      C c = new C();
19      c.method();
20    }
21  }
```

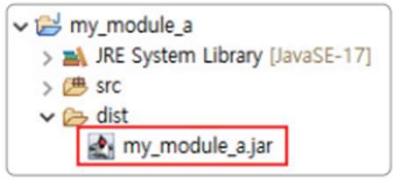
(3행, 4행) my_module_a 모듈에서 가져옴

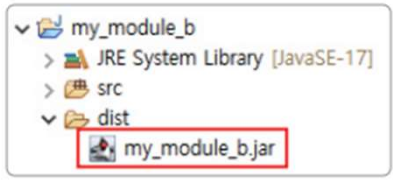
(5행) my_module_b 모듈에서 가져옴

(9~15행) my_module_a 모듈의 클래스 이용

(17~19행) my_module_b 모듈의 클래스 이용

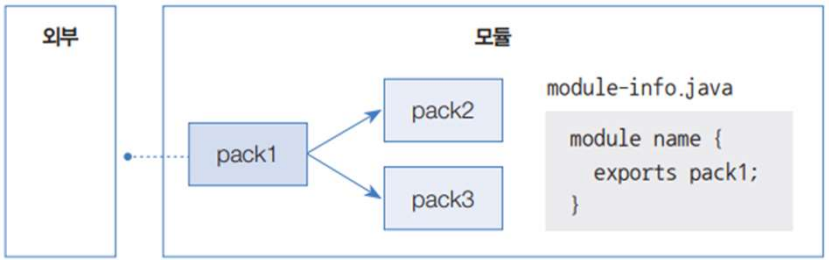






패키지 은닉

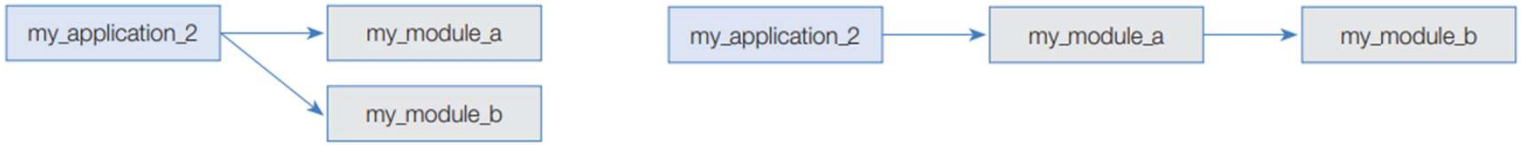
- 모듈은 모듈 기술자에서 exports 키워드를 사용해 내부 패키지 중 외부에서 사용할 패키지를 지정
- exports되지 않은 패키지는 자동적으로 은닉



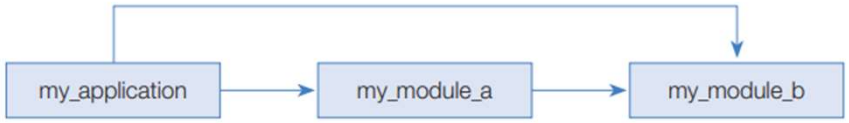
- 한 가지 패키지로 모듈 사용 방법 통일
- 다른 패키지를 수정하더라도 모듈 사용 방법이 바뀌지 않아 외부에 영향을 주지 않음

의존 설정 전이하기

- my_application_2 프로젝트는 직접적으로 두 모듈 my_module_a, my_module_b를 requires하고 있는 의존 관계
- my_application_2는 my_module_a에 의존하고, my_module_a는 my_module_b에 의존하는 관계로 변경하면 컴파일 오류 발생



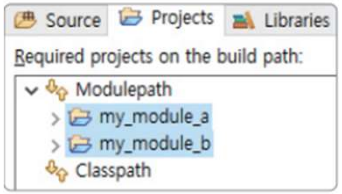
- 의존 설정 전이: my_module_a의 모듈 기술자에 transitive 키워드와 함께 my_module_b를 의존 설정하면 해결



집합 모듈

- 여러 모듈을 모아놓은 모듈. 자주 사용되는 모듈들을 일일이 requires하지 않아 편리.
- 집합 모듈은 자체적인 패키지를 가지지 않고, 모듈 기술자에 전이 의존 설정만 함

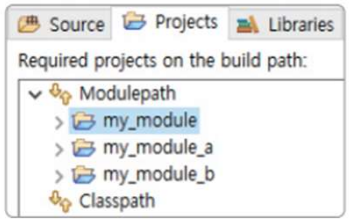
```
module my_module {
  requires transitive my_module_a;
  requires transitive my_module_b;
}
```



>>> module-info.java

```
1   module my_application_2 {
2     //requires my_module_a;
3     //requires my_module_b;
4     requires my_module;
5   }
```

my_module 모듈에만 의존



리플렉션

- 실행 도중에 타입(클래스, 인터페이스 등)을 검사하고 구성 멤버를 조사하는 것
- 은닉된 패키지는 기본적으로 다른 모듈에 의해 리플렉션을 허용하지 않음
- 모듈은 모듈 기술자를 통해 모듈 전체 또는 지정된 패키지에 대해 리플렉션을 허용할 수 있고, 특정 외부 모듈에서만 리플렉션을 허용할 수도 있음

모듈 전체를 리플렉션 허용

```
open module 모듈명 {
  ...
}
```

지정된 패키지에 대해 리플렉션 허용

```
module 모듈명 {
  ...
  opens 패키지1;
  opens 패키지2;
}
```

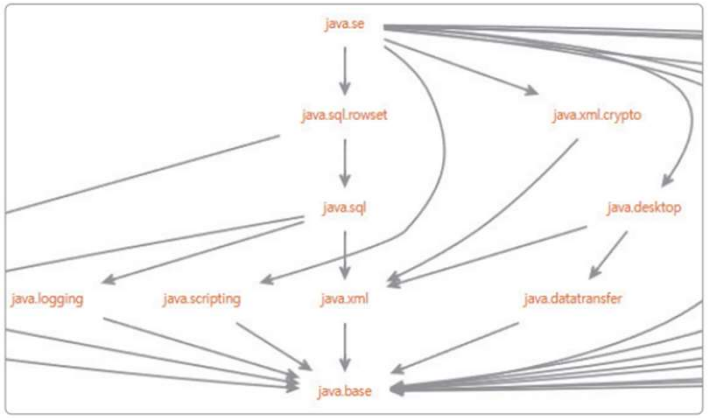
지정된 패키지에 대해 특정 외부 모듈에서만 리플렉션 허용

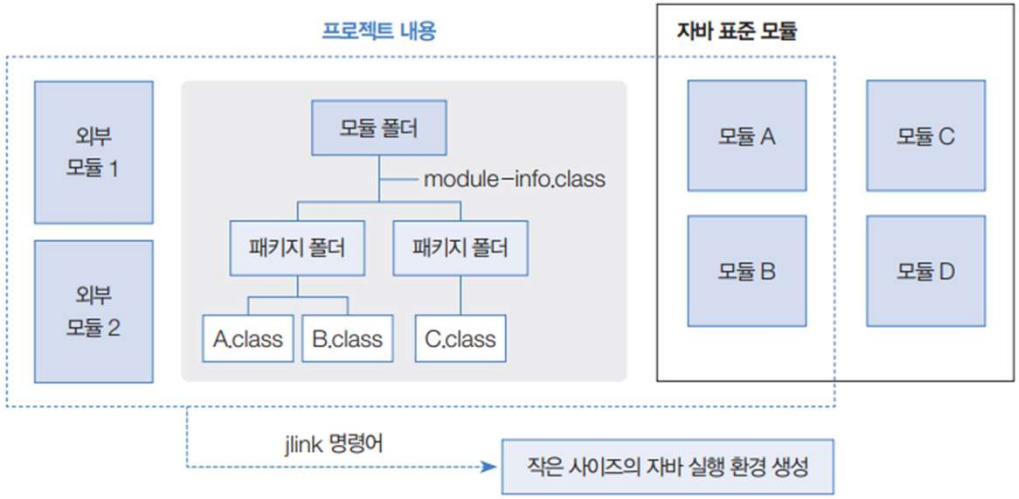
```
module 모듈명 {
  ...
  opens 패키지1 to 외부모듈명, 외부모듈명, ···;
  opens 패키지2 to 외부모듈명;
}
```

표준 라이브러리

- JDK가 제공하는 표준 라이브러리는 Java 9부터 모듈화됨
- 응용프로그램을 실행하는 데 필요한 모듈만으로 구성된 작은 사이즈의 자바 실행 환경(JRE)
- Java 17의 전체 모듈 그래프(화살표는 모듈간의 의존 관계를 표시)

  https://docs.oracle.com/en/java/javase/17/docs/api/java.se/module-summary.html






transient
false
finally
import
new
byte
throws
interface
null
long
package
return
extends
for
throw
catch
char
int
synchronized
case
instanceof
enum
default
continue
abstract
final
boolean
private
double
try
public
do
this
short
class
native
true
implements
while
static
else
void
switch

Thank you!