

## 20. 모듈 시스템 (Java 9+)

### module-info.java

#### module-info.java란?

module-info.java는 모듈 선언 파일로, Java 9 이상의 모듈 시스템(JPMS: Java Platform Module System)을 사용하는 프로젝트의 최상위에 위치함.

이 파일을 통해 모듈의 의존성, 공개 API, 서비스 제공 및 사용 등을 선언할 수 있다.

```
1 module com.example.myapp {  
2     requires java.sql;  
3     exports com.example.myapp.api;  
4 }
```

#### 기본 구성 요소

키워드	의미
<code>module</code>	모듈 선언 시작
<code>requires</code>	다른 모듈에 대한 의존성 선언
<code>exports</code>	외부에 공개할 패키지 지정
<code>opens</code>	리플렉션 기반 접근을 허용할 패키지 지정
<code>uses</code>	SPI(Service Provider Interface) 사용 선언
<code>provides ... with</code>	SPI 구현체 등록

#### 기본 예제

```
1 module com.example.myapp {  
2     requires java.sql;  
3     requires com.google.gson;  
4  
5     exports com.example.myapp.api;  
6 }
```

- `requires java.sql`: SQL 기능이 필요함
- `exports com.example.myapp.api`: 해당 패키지만 외부에서 접근 가능

## ✓ 고급 키워드 설명

### 1. exports

- 외부 모듈에서 사용할 수 있는 **공개 API 패키지** 선언
- `exports` 하지 않으면 해당 패키지는 **은닉됨** (다른 모듈에서 import 불가)

```
1 | exports com.example.utils;
```

### 2. requires

- 다른 모듈에 **의존한다고 명시**
- `requires transitive`: 내가 의존하는 모듈을 **나를 사용하는 쪽도 자동으로 가져가게 함**

```
1 | requires java.logging;  
2 | requires transitive my.library;
```

### 3. opens

- `exports` 와 달리, **리플렉션을 허용할** 패키지를 지정
- `Jackson`, `JPA`, `Spring` 같은 프레임워크가 내부 필드를 `reflect` 할 수 있게 함

```
1 | opens com.example.model to com.fasterxml.jackson.databind;
```

- 특정 모듈만 리플렉션 허용 (`to` 사용)
- 전체 모듈에 열려면 그냥 `opens com.example.model;`

### 4. uses / provides

#### SPI(Service Provider Interface)를 활용할 때 사용

```
1 | uses com.example.spi.MyService;
```

```
1 | provides com.example.spi.MyService with com.example.impl.MyServiceImpl;
```

## 디렉토리 구조 예시

```
1 src/
2   └─ com.example.myapp/
3       └─ module-info.java
4       └─ api/
5           └─ HelloService.java
6       └─ impl/
9           └─ HelloServiceImpl.java
```

```
1 // module-info.java
2 module com.example.myapp {
3     exports com.example.myapp.api;
4     requires java.base; // java.base는 생략 가능 (기본 포함)
5 }
```

## 주의사항

항목	설명
<code>exports</code> 없는 패키지는 외부에서 보이지 않음	public 클래스라도 <code>exports</code> 필요
<code>opens</code> 없이 리플렉션으로 private 필드 접근 불가	JPA, Jackson 등의 프레임워크에 중요
모듈 간 순환 의존 불가	모듈 A ↔ 모듈 B는 불가
split package 금지	하나의 패키지를 여러 모듈에 나누어 가질 수 없음

## 모듈 시스템 도입 이유

목적	설명
캡슐화 강화	내부 구현을 모듈 외부에 감춤
명확한 의존성	어떤 모듈이 무엇을 사용하는지 선언적 관리
런타임 최적화	필요한 모듈만 JVM에 로딩
보안 강화	무분별한 리플렉션 방지, 접근 제한 가능

## Maven과 모듈 시스템

Maven에서 모듈 시스템을 쓸 경우, `module-info.java`를 `src/main/java`에 위치시키면 됨.  
추가적인 설정 없이도 컴파일되지만, JAR 단위로 모듈 이름 충돌 주의 필요.

## ✓ 요약

구성	예시
모듈 선언	<code>module com.example.app { ... }</code>
모듈 의존성	<code>requires java.sql;</code>
공개 패키지	<code>exports com.example.api;</code>
리플렉션 허용	<code>opens com.example.model;</code>
서비스 사용/제공	<code>uses, provides ... with</code>

## exports, requires 키워드

### 개념

`exports`는 현재 모듈이 외부에 공개하고자 하는 패키지를 지정할 때 사용한다.

`exports`하지 않으면, `public` 클래스여도 외부에서 접근할 수 없다.

### 문법

```
1 | exports <패키지명>;
```

또는 특정 모듈에만 공개할 수도 있음:

```
1 | exports <패키지명> to <모듈명>;
```

### 예시

```
1 | // module-info.java
2 | module com.example.library {
3 |     exports com.example.api;
4 | }
```

이 경우, `com.example.api` 패키지 안의 `public` 클래스들만 다른 모듈에서 사용할 수 있다.

```
1 | exports com.example.internal; // ✗ 안 하면 외부 모듈에서 접근 불가
```

### 부분 공개 (to 키워드)

```
1 | exports com.example.secret to com.example.friend;
```

- `com.example.friend` 모듈에서만 `com.example.secret` 패키지를 사용할 수 있음
- 부분적 API 공개, 보안 강화에 유용

## ✓ 2. `requires` 키워드 — "필요한 모듈"

### 개념

`requires` 는 현재 모듈이 동작하는 데 의존하는 외부 모듈을 명시할 때 사용한다.  
자바 모듈 시스템에서는 클래스 경로가 아니라 **모듈 이름**으로 의존성을 정의한다.

### 문법

```
1 | requires <모듈명>;
```

예:

```
1 | requires java.sql;  
2 | requires com.fasterxml.jackson.databind;
```

### `requires transitive`

- 내가 의존하는 모듈을 나를 사용하는 다른 모듈에도 전이(전달)
- 라이브러리 개발자 입장에서 자주 쓰임

```
1 | module my.framework.core {  
2 |     requires transitive my.framework.logging;  
3 | }
```

→ 이 경우, `my.framework.core` 를 사용하는 쪽은 자동으로 `my.framework.logging` 도 같이 사용할 수 있음

### `requires static`

- 컴파일 시에는 필요하지만, 런타임에는 없어도 되는 모듈
- 예: `annotation processor`, `IDE plugin`, `test API` 등

```
1 | requires static com.google.auto.service;
```

## 🔍 예제: 실전 구조

```
1 | module com.myapp.module {  
2 |     requires java.sql;  
3 |     requires com.google.gson;  
4 |  
5 |     exports com.myapp.service;  
6 |     exports com.myapp.model to com.myapp.frontend;  
7 | }
```

- `java.sql`, `gson` 기능 사용 가능
- `com.myapp.service` 는 모든 모듈에 공개
- `com.myapp.model` 은 `com.myapp.frontend`에만 공개

## ✅ 차이 요약

키워드	목적	적용 대상	주의점
<code>exports</code>	패키지 공개	내 모듈 안의 패키지	선언하지 않으면 <code>public</code> 이어도 외부에서 접근 불가
<code>requires</code>	외부 모듈 사용 선언	외부 모듈 이름	순환 의존 불가, 자동 모듈은 이름 불안정

## 🔒 왜 명시적으로 해야 하나?

자바 9부터는 캡슐화를 강화하고 불필요한 의존성 제거, 보안 강화, 경량화를 위해 이렇게 선언적으로 공개 범위와 의존성을 명시하게 된 거야.

## 📌 정리 요약

- `exports` 는 내부 패키지를 외부에 공개할 때 사용
- `requires` 는 다른 모듈에 의존할 때 사용
- `exports ... to` 로 특정 모듈에만 제한 공개 가능
- `requires transitive`, `requires static` 은 전이 의존성/컴파일 전용 의존성을 제어할 수 있음

## 모듈 간 의존성

### ✅ 모듈 간 의존성이란?

한 모듈이 다른 모듈의 `public API`에 접근하기 위해 선언하는 관계이다.

이 의존 관계는 `requires`, `requires transitive`, `exports`, `exports to` 같은 키워드로 설정된다.

### ✅ 기본 예시 구조

```
1 | 📦 com.example.api
2 |   └─ module-info.java
3 |   └─ ApiService.java
4 |
5 | 📦 com.example.app
6 |   └─ module-info.java
7 |   └─ MainApp.java
```

## com.example.api/module-info.java

```
1 module com.example.api {
2     exports com.example.api;
3 }
```

## com.example.app/module-info.java

```
1 module com.example.app {
2     requires com.example.api;
3 }
```

- `com.example.app`은 `com.example.api`의 클래스를 사용할 수 있음
- 단, `com.example.api` 모듈에서 `exports`한 패키지에 한해서만 접근 가능

## ✅ 의존 관계 유형

### 1. requires

- 기본적인 의존 선언
- 현재 모듈이 다른 모듈의 API를 사용할 수 있게 함

```
1 requires com.example.api;
```

### 2. requires transitive

- 전이적 의존성 (Transitive Dependency)
- $A \rightarrow B \rightarrow C$  관계일 때, A가 B를 `requires transitive`하면 C가 A만 `requires`해도 B에 접근 가능

#### 예시

```
1 module framework.core {
2     requires transitive framework.logging;
3 }
```

- `framework.core`를 사용하는 모듈은 `framework.logging`도 자동 사용 가능

### 3. requires static

- 컴파일 시에는 필요하지만, 런타임에는 없어도 되는 모듈
- 주로 어노테이션 프로세서, IDE 전용 API 등에 사용됨

```
1 requires static com.google.auto.service;
```

## ✓ 모듈 간 접근 제어

### exports

- 모듈 내 패키지를 외부에 공개
- 공개하지 않으면, public 클래스도 외부에서 접근 불가

```
1 | exports com.example.service;
```

### exports ... to

- 특정 모듈에만 부분 공개

```
1 | exports com.example.secret to com.example.trusted;
```

### opens

- 리플렉션 기반 프레임워크(Jackson, Spring 등)를 위한 접근 허용
- 컴파일 타임이 아닌 런타임 리플렉션에 필요한 경우 사용

```
1 | opens com.example.model;
```

### opens ... to

- 특정 모듈에만 리플렉션 열기

```
1 | opens com.example.internal to jackson.databind;
```

## ✓ 순환 의존성 금지

Java 모듈 시스템은 모듈 간의 순환 의존을 허용하지 않는다.

```
1 | module A {  
2 |     requires B;  
3 | }  
4 |  
5 | module B {  
6 |     requires A; // ✖ 컴파일 오류  
7 | }
```

→ 이런 경우는 인터페이스 분리 또는 공통 인터페이스 모듈을 만들어 해결해야 한다.



## ✓ 모듈 그래프 예시

```
1 | [com.myapp] → requires → [com.lib.core] → requires transitive → [com.lib.utils]
```

- `com.myapp`은 `com.lib.utils`를 명시적으로 `requires`하지 않아도 됨
- 단, `com.lib.utils`는 반드시 `exports`되어 있어야 사용 가능함

## 🧠 정리 요약표

선언 유형	의미	대상
<code>requires</code>	다른 모듈에 대한 기본 의존	전체 모듈
<code>requires transitive</code>	이 모듈을 사용하는 모듈에게도 전이 공개	전체 모듈
<code>requires static</code>	컴파일 전용 의존성	전체 모듈
<code>exports</code>	패키지를 외부에 공개	모든 모듈
<code>exports to</code>	특정 모듈에만 공개	일부 모듈
<code>opens</code>	리플렉션 접근 허용	모든 모듈
<code>opens to</code>	특정 모듈만 리플렉션 허용	일부 모듈

## 🔧 도구로 의존성 시각화

- `jdeps` 명령어로 모듈 간 의존성 확인 가능

```
1 | jdeps --module-path mods -s mods/com.example.app.jar
```