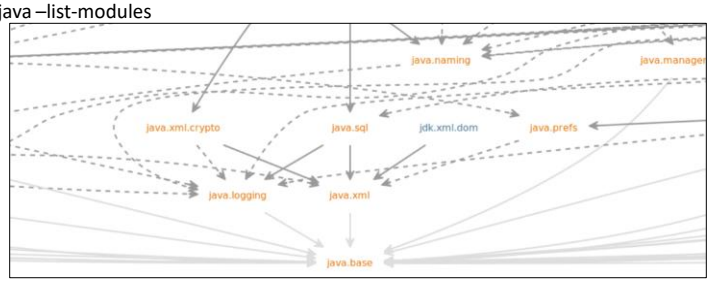
Module

sanghyuck.na@lge.com

1

Modular System[9]

- Java system
  - `95년부터 JAVA SE는 하나의 거대한 일체형 라이브러리 였습니다.
  - JAR와 그 세부 package에 대한 접근자, 공개자를 세분화 하지 못했습니다.
- `17 Sep, Java9 JDK적용
  - 277: JAVA Module System(`05 Java7) 제안, 376: Module System('17 Java9)로 대체
  - 플랫폼에 관한 사항 - 200: The Modular JDK, 201: Modular Source Code, 220: Modular Run-Time Images, 260: Encapsulate Most Internal APIs
  - 모듈에 관한사항 – 261: Module System
- Module graph
  - java.base는 명시적 선언, 관계없이 기본 참조
  - java –list-modules



2

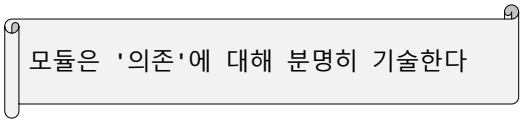
Goal[JSR376]

- 신뢰할 만한 구성
  - Modularity는 어떤 방식이든 Compile time이나 Runtime에서 모두 의존성을 확실히 선언하는 것에 대한 수단을 제공합니다.
- 강한캡슐화
  - 모듈 내 패키지는 사용하려는 모듈에 분명히 패키지에 대한 'exports'선언이 되어 있어야 접근 가능합니다.
- 확장가능한 Java 플랫폼
  - 약 95개 모듈로 세분화 되면서, 비지니스개발자는 타겟이 실제로 사용하는 모듈로만 자신의 앱을 구성하여 JAVA플랫폼을 배포 가능합니다(jlink)
- 향상된 플랫폼 무결성
  - Internal API는 캡슐 화되어 숨겨져서 오래된 app은 JAVA9으로 migration해야 합니다.
- 성능개선
  - 특정 모듈에 필요한 타입을 미리 JVM은 인지할 수 있음으로써 다양한 실행속도와 메모리재배치 최적화기법이 가능하게 됐습니다.

3

Module

- 정의
  - Module descriptor (module-info.java)와 연관된 Package, Resource를 묶은 한 그룹
  - Package는 JAVA package와 동일하며 Resource는 클래스 이 외에 파일입니다.
- Module descriptor module-info.java
  - package상단에 위치하여 모듈 및 패키지를 설명하는 파일
- Module구성요소
  - Module name
  - Module's dependencies
  - Public packages: 외부에서 접근할 수 있는 package리스트
  - Services it offered: 외부에서 사용될 Service class구현체
  - Service it consumed: 외부에서 사용할수 있는 Service
  - Reflection permissions: JAVA Reflection으로 사용을 허가할 private member리스트

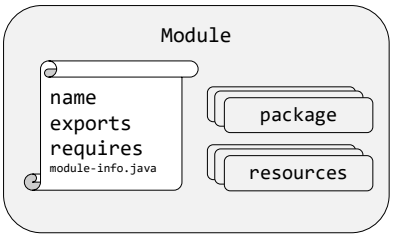


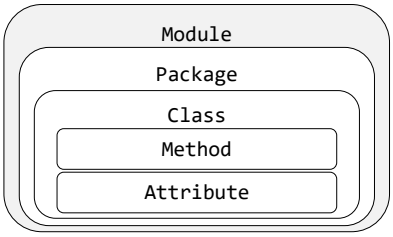
4

module-info.java

- 기술내용
  - 외부에 공개 exports
  - 특정 모듈에만 공개 exports to
  - 컴파일타임에만 필요 requires static
  - 묵시적 의존 requires transitive
  - 서비스 사용 uses
  - 리플렉션으로 공개 opens

```
module com.addresschecker {
}
```



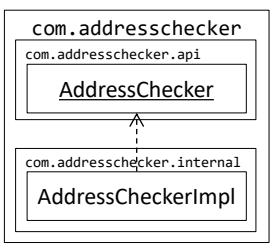


5

exports

- 공개 모듈의 패키지 지시자
  - exports 지시자로 공개할 패키지 지정하여 노출된(Exported) 패키지를 다른 패키지가 사용할 수 있게 합니다.
  - Comma로 분리해서 여래 개 모듈을 작성할 수 있습니다.
  - Qualified export란 "to"로 해당 모듈패키지를 사용할 대상 모듈 지정하는 기법

```
module com.addresschecker {
    exports com.addresschecker.api;
}
```




```
String value = args.length > 0 ? args[0] : "";
boolean isZipCode = new
AddressCheckerImpl().checkZipCode(value);
if (isZipCode)
    System.out.println(
          value + " is a valid zip code");
else
    System.out.println(
          value + " is not a valid zip code");
```

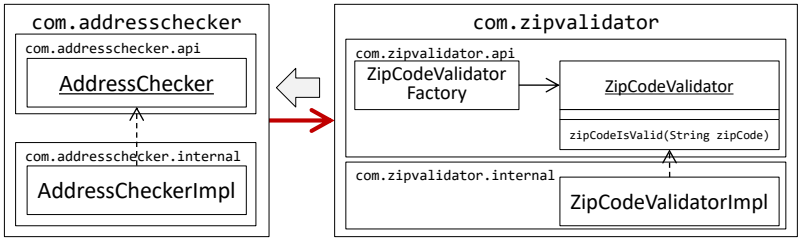
6

requires

- 의존하는 모듈 지정 지시자
  - 이 관계는 Module dependency입니다. 각 모듈은 분명히 의존하는 모듈명을 이 지시자로 지정해야 합니다.
  - A requires B의 의미는 "모듈A는 모듈B를 접근(read)한다", "모듈B는 모듈A가 접근(read by)한다"라고 합니다.
- requires static
  - 지시되는 모듈은 compile time만 필요하다는 의미

```
module com.addresschecker {
    requires com.zipvalidator;
    exports com.addresschecker.api; }
```

```
module com.zipvalidator {
    exports com.zipvalidator.api;
}
```



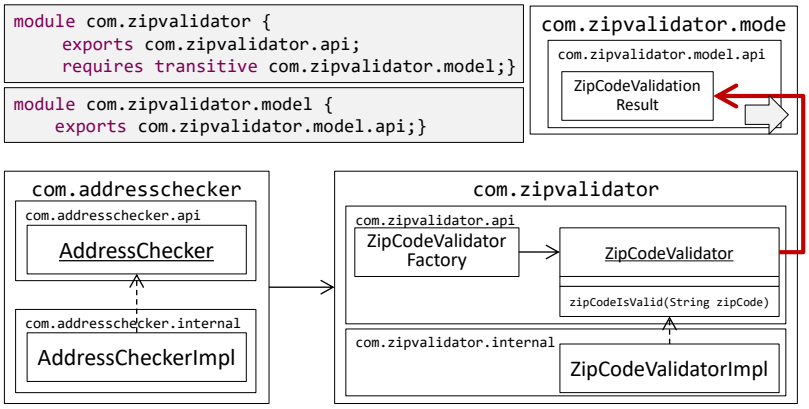
7

requires transitive

- 묵시적접근(Implied readability) 지시자
  - 이 지시자를 사용하여 모듈C에 의존성을 지정한 모듈B를 사용하는 모듈A는 묵시적으로 모듈C를 의존하게 됩니다.
  - com.zipvalidator는 com.zipvalidator.model를 requires transitive합니다. com.addresschecker는 분명히 requires com.zipvalidator.model를 지정하지 않아도 됩니다.

```
module com.zipvalidator {
    exports com.zipvalidator.api;
    requires transitive com.zipvalidator.model;}
```

```
module com.zipvalidator.model {
    exports com.zipvalidator.model.api;}
```

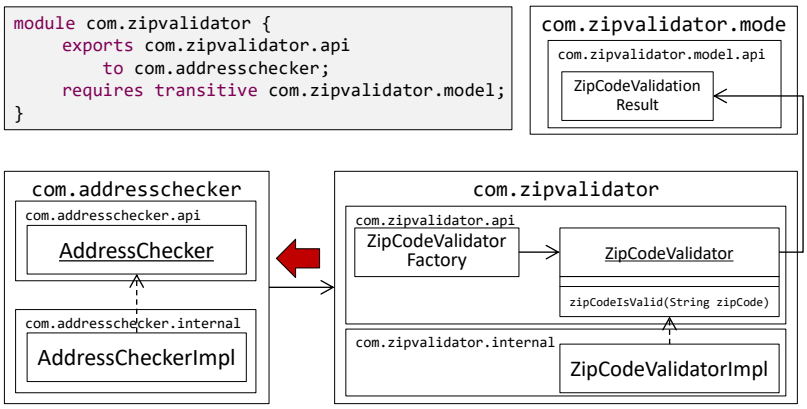


8

qualified exports

- 특정 모듈로 공개가 제한된 패키지 지시자
  - 모듈패키지를 사용할 대상 모듈 지정하는 기법
  - com.zipvalidator.api는 오직 com.addresschecker에만 export됩니다.

```
module com.zipvalidator {
    exports com.zipvalidator.api
        to com.addresschecker;
    requires transitive com.zipvalidator.model;
}
```



9

open

- Reflection 사용 지시자
  - Reflection으로 접근할 수 있는 공개(export) 패키지 지정
  - JAVA9 이전 까지는 Reflection으로 패키지의 모든타입, 모든타입멤버, 심지어 private 임에도 접근될 수 있었습니다. 모듈개발자 의도와는 상관 없게 캡슐화는 깨졌습니다
  - Module system의 특징 중 하나는 강한캡슐화(Strong encapsulation)입니다. module-info.java의 open으로 지시하지 않는다면 reflection으로도 이제는 읽을 수 없습니다.

```
ClassLoader classLoader = AddressCheckerImpl.class.getClassLoader();
try {
    String apath ="com.zipvalidator.internal.ZipCodeValidatorImpl";
    Class aClass = classLoader.loadClass(apath);
    return ((ZipCodeValidator) aClass.newInstance())
        .zipCodeIsValid(zipCode) == ZipCodeValidationResult.OK;
} catch (Exception e) {
    throw new RuntimeException(e);
}
```

```
module com.zipvalidator {
    exports com.zipvalidator.api to com.addresschecker;
    opens com.zipvalidator.internal;
    requires transitive com.zipvalidator.model;
}
```

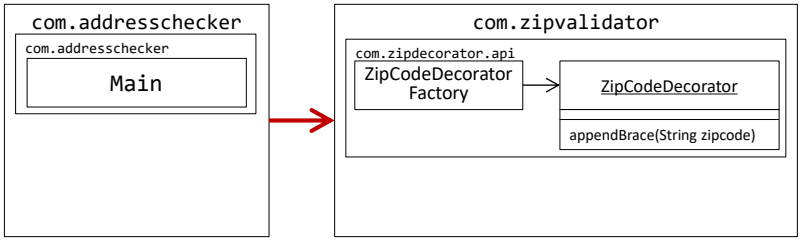
10

uses

- 사용 서비스 지정(Consumer)
  - 현재 모듈(client)가 사용할 서비스 인터페이스를 지정합니다.
  - 사용하는(use) 서비스는 대상 모듈에서 provide with로 기술되어야 하고, 해당 패키지는 exports로 외부 공개되어야 합니다.

```
module com.addresschecker { …
    uses com.zipdecorator.api.ZipCodeDecorator;}
```

```
ZipCodeDecorator deco = ZipCodeDecoratorFactory.newDecorator();
System.out.println(deco.appendBrace(zipcode));
```

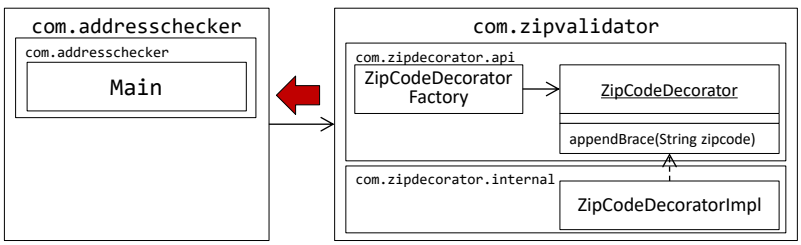


11

provides with

- 서비스 구현체를 기술하는 지시자(Provider)
  - 서비스를 구현하는 구체적인 클래스 기술
  - provides [interface | abstract-class] with [service-provider] : service-provider는 interface나 abstract-class를 extend나 implements하는 provider class를 기술합니다.

```
module com.zipvalidator {…
    exports com.zipdecorator.api;
    provides com.zipdecorator.api.ZipCodeDecorator
            with com.zipdecorator.internal.ZipCodeDecoratorImpl;
}
```



12

Service Loader[6]

- Service-provider 동적로딩
  - Service과 Service-provider를 직접적으로 생성하지 않고 서로 분리로딩 기법
  - Service란 잘 알려진 inteface나 abstract class
  - Service provider는 그 service를 구체화한 구현체
  - Service provider는 JAVA platform이나, jar 파일로 설치될 수 있습니다.

```
ServiceLoader<ZipCodeDecorator> sl = ServiceLoader
        .load(ZipCodeDecorator.class);
Iterator<ZipCodeDecorator> iter = sl.iterator();
if(iter.hasNext()) {
   ZipCodeDecorator provider = iter.next();
   System.out.println(provider.appendBrace(zipcode));
}
```

13

Module Utility

- 설치 된 JRE 모듈조회
- 지정 모듈의 module-info.java 조회
- Java class의 의존성 분석도구

```
> java --list-modules

> java -d java.base
java.base@13
exports java.io
uses sun.util.spi.CalendarProvider
…

> jdeps -s jrt-fs.jar
jrt-fs.jar -> java.base
```

14

정리

- Modular system

17