THE DEVELOPER'S CONFERENCE

porto alegre,
novembro 2017

a arquitetura modular do java 9

summa
Technology + Business

helder da rocha

Quem sou eu? Who am I? Кто я?

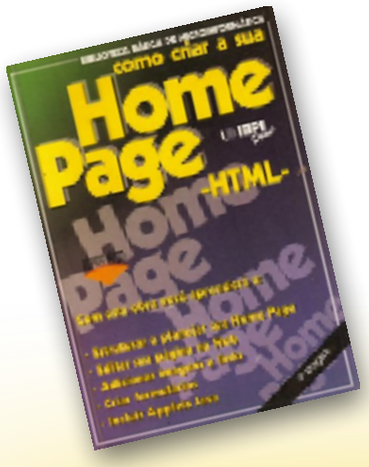
Helder da **Rocha**

Tecnologia * Ciência * Arte

HTML & tecnologias Web desde 1995

Autor de cursos e livros sobre

Java, XML e tecnologias Web



argonavis.com.br

helderdarocha.com.br

Java 9

- **JSR 379** do Java Community Process
- **Comunidade** propõe e vota features: JEPs (JDK Enhancement Proposals)
- **91** novidades http://openjdk.java.net/projects/jdk9/
- Lançado em **21 de setembro de 2017**
- Atualmente disponível (novembro 2017) na versão **9.01**

História das versões Java

- JDK Alfa e Beta: **1994**
- JDK 1.0 **1995** (8 pacotes!)
- JDK 1.1 **1996**
- Java 2 (J2SE 1.2) **1998**
- J2SE 1.3 **2000**
- J2SE 1.4 **2002**
- J2SE 5.0 **2005**
- Java SE 6 **2006**
- Java SE 7 **2011**
- Java SE 8 **2014**
- Java SE 9 **2017**

Como instalar e programar?

- Baixe de www.oracle.com/technetwork/java/javase/downloads
- Para programar em Java 9
  - JDK + linha de comando
  - NetBeans
  - Eclipse
  - IntelliJ

Modularidade

- Técnica para **reduzir a complexidade** (organizar para poder usar apenas o código necessário)

- Aumenta a **coesão**, **reuso**, **encapsulamento**, **eficiência**

- Módulos devem ter

  - Encapsulamento forte

  - Interfaces bem definidas

  - Dependências explícitas

Arquitetura modular do Java

- Grande impacto no desenvolvimento: envolve a estrutura de aplicações inteiras.

  - **JDK e JRE**: divide JDK (tools.jar) e Runtime (rt.jar) em blocos menores

  - **Encapsula** APIs internas (que não devem ser usadas) (ex: sun.misc.*)

  - Fornece **arquitetura modular** para que usuário possa desenvolver aplicações modulares

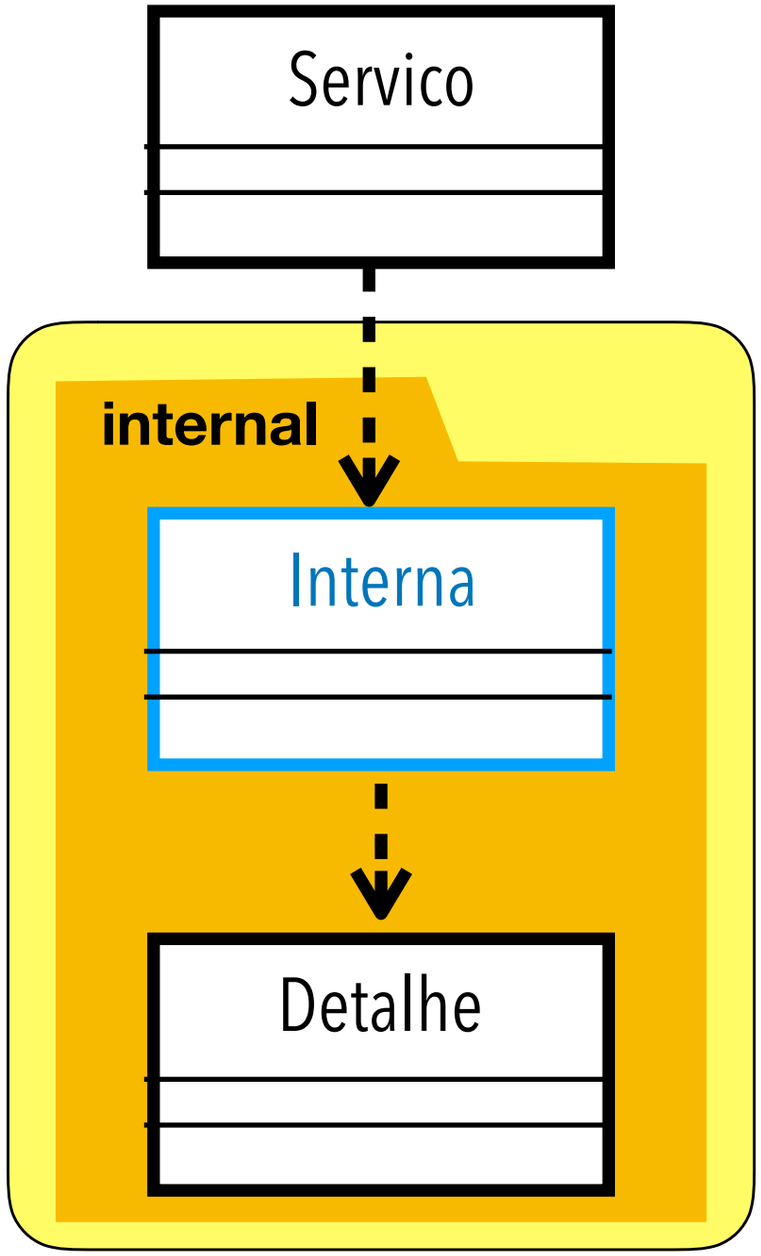
  - Aplicativo **jlink** para gerar imagens executáveis

Antes de Java 9

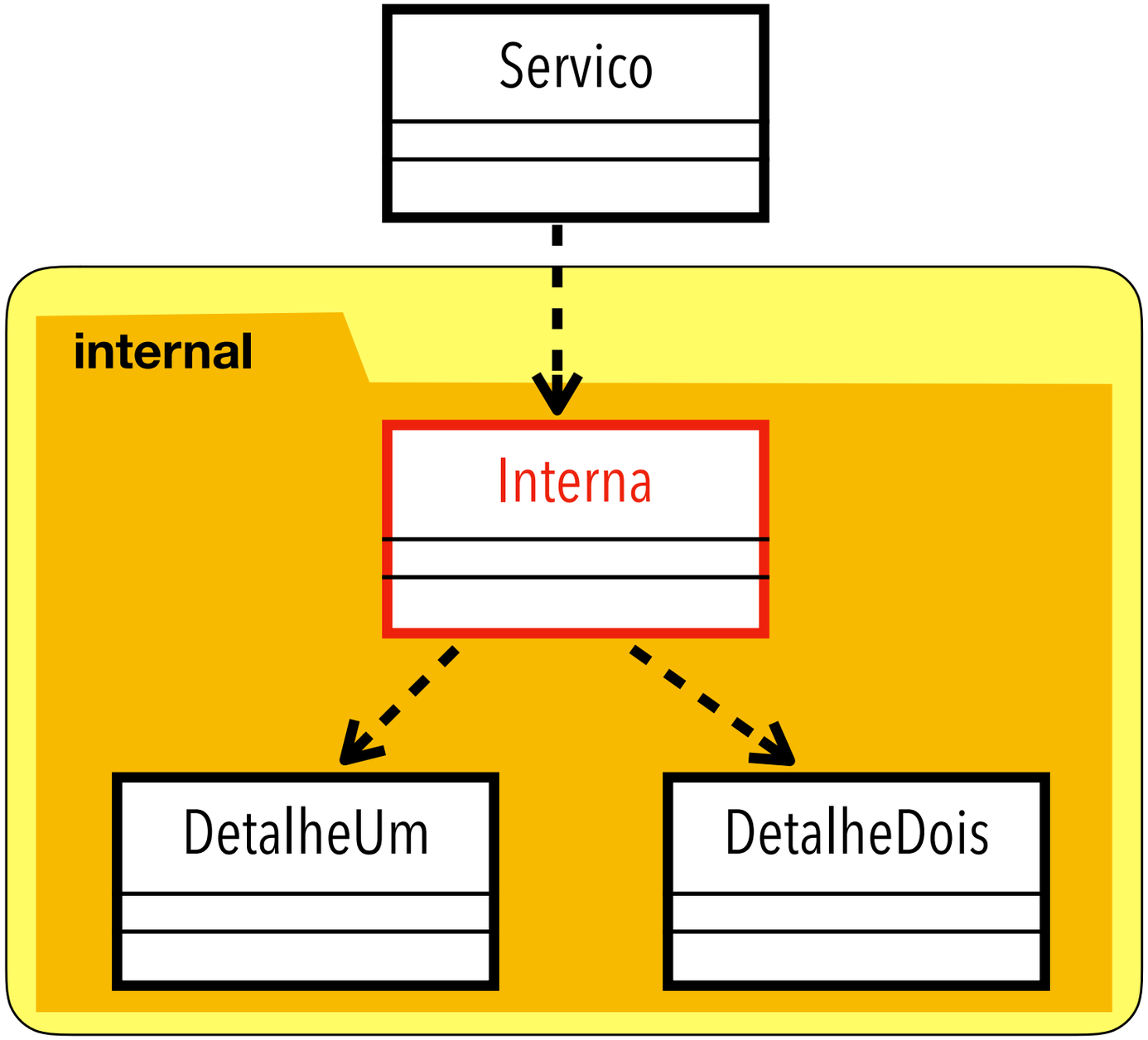
- Modificadores de acesso, pacotes, JARs
  - Dependências diretas e transitivas (**ocultas**). Para executar é preciso incluir todas as dependências, diretas e transitivas
    `java -cp ... um monte de linhas ... -jar App.jar`
    - **rt.jar** (grande) e outros JARs do JRE
    - Acesso a classes públicas de **uso interno** (ex: `sun.misc.BASE64Encoder`)
    - **Classpath/Classloader Hell** - risco de haver cópias duplicadas do mesmo JAR, geralmente dependências transitivas, em versões diferentes
- Soluções externas ao Java: **Maven** (compile-time) + **OSGi** (runtime)

Classpath hell

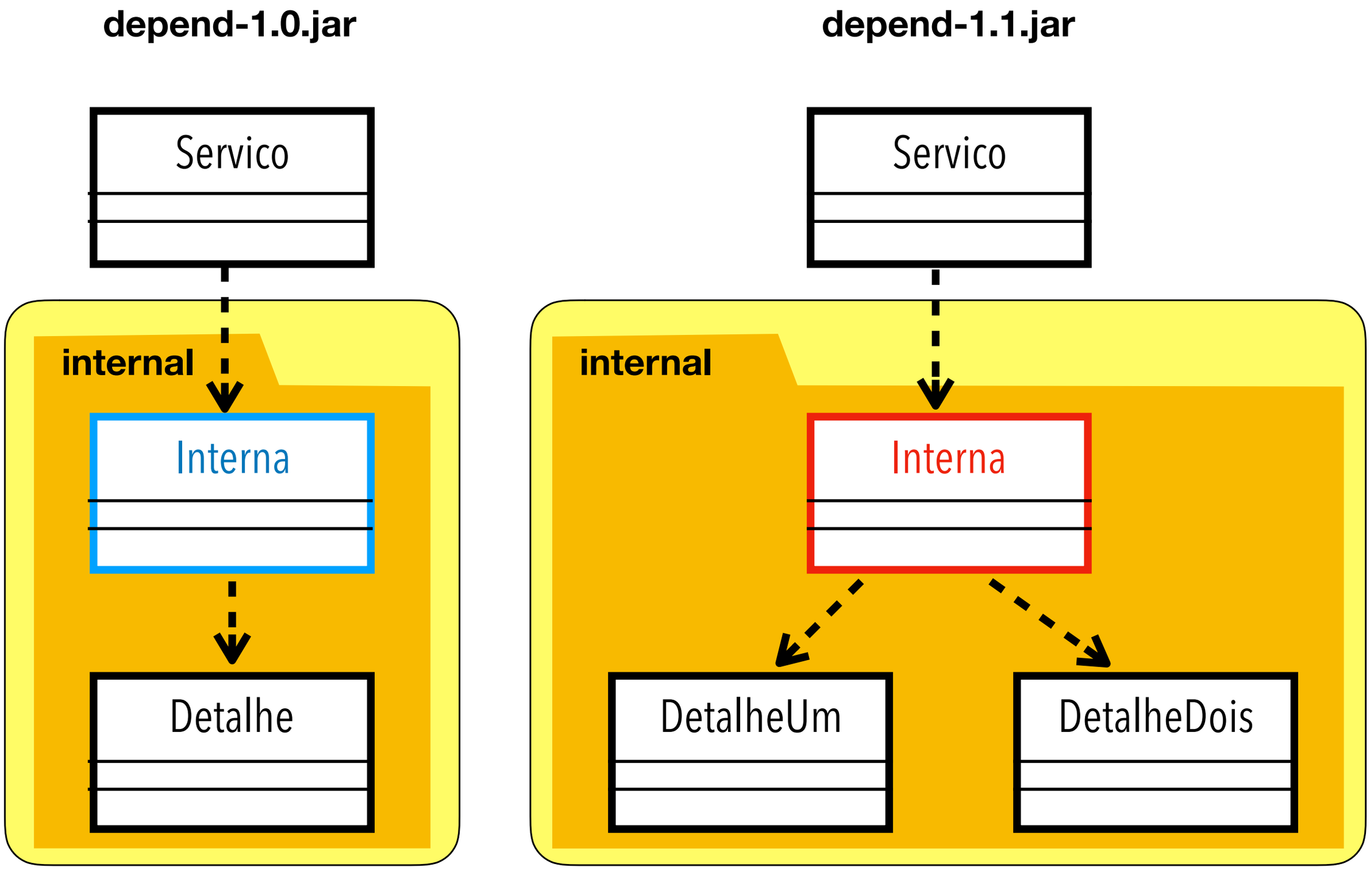
depend-1.0.jar



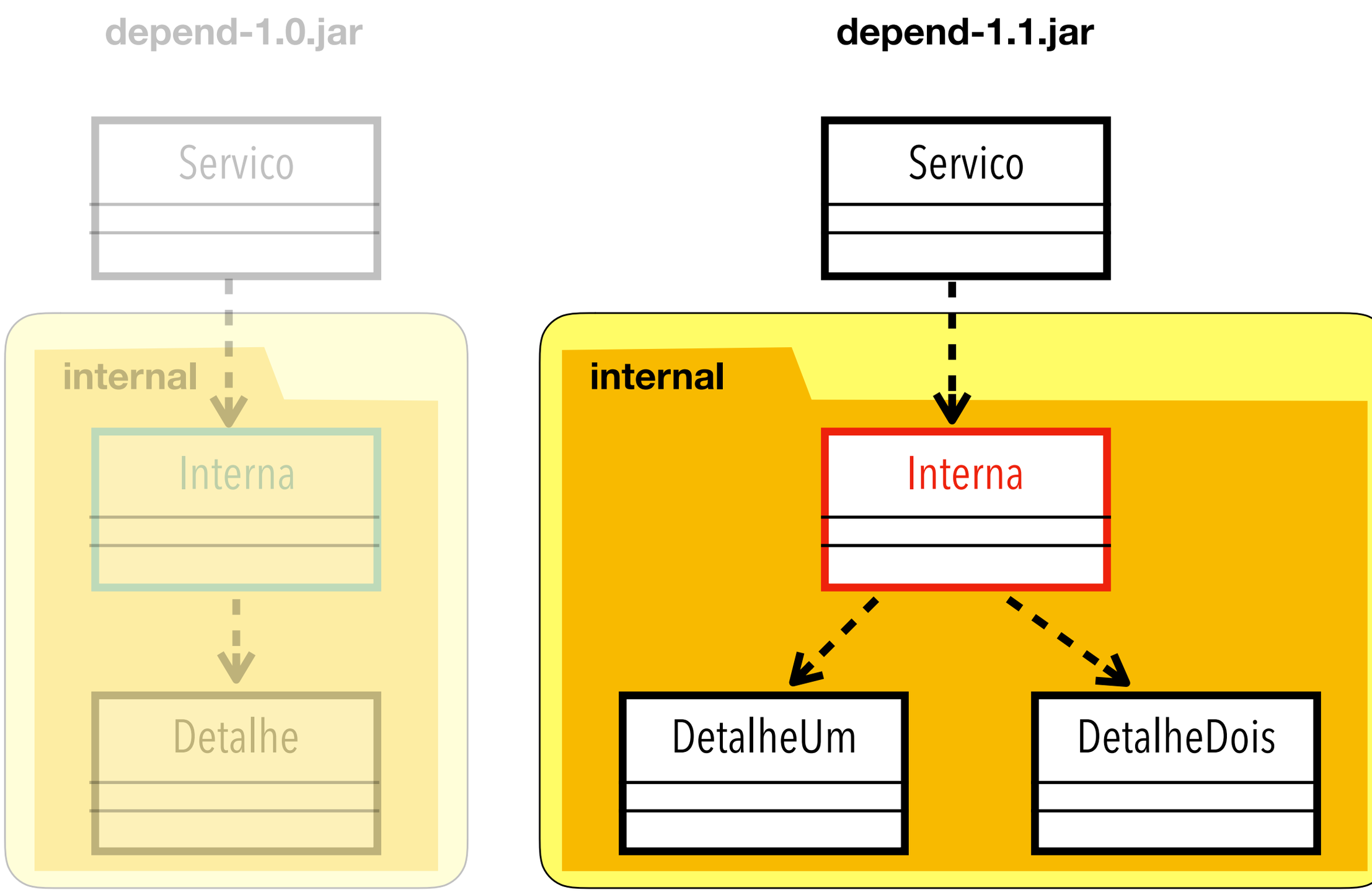
depend-1.1.jar



java -cp depend-1.1.jar;outros-1.x.jar;depend-1.0.jar -jar App.jar

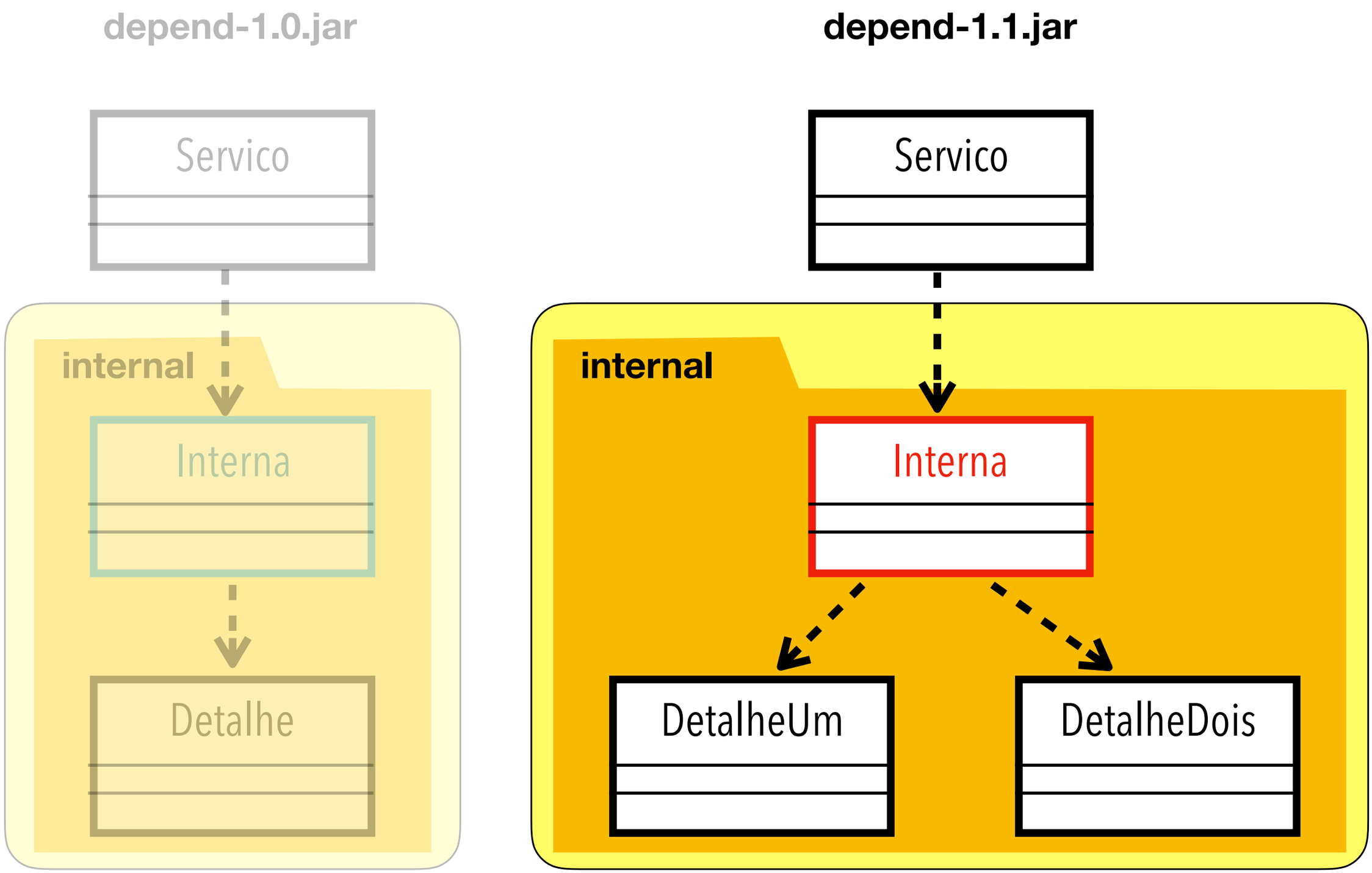


java -cp depend-1.1.jar;outros-1.x.jar;depend-1.0.jar -jar App.jar

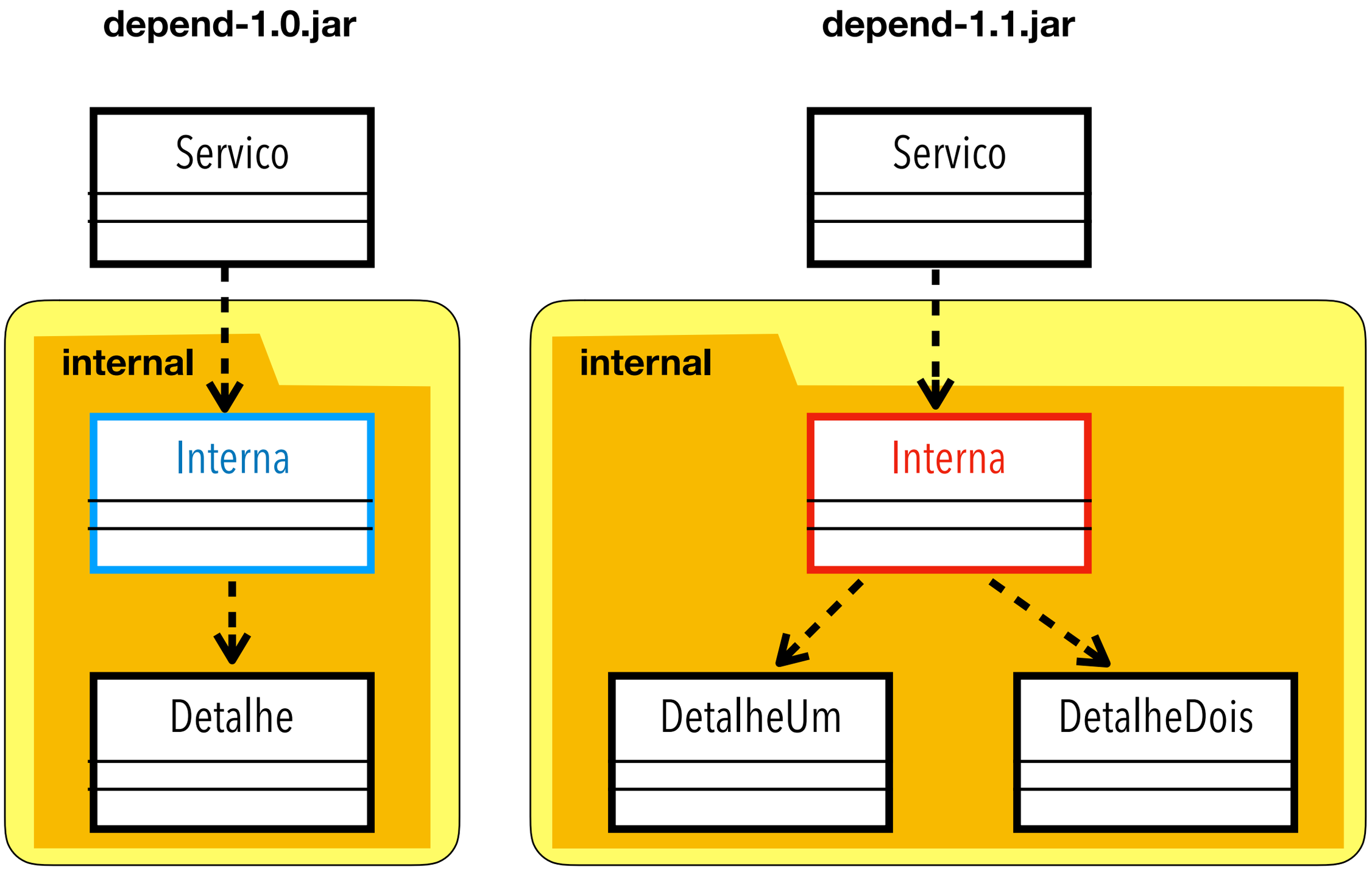


java -cp depend-1.1.jar;outros-1.x.jar;depend-1.0.jar -jar App.jar

... Servico("dois") ...

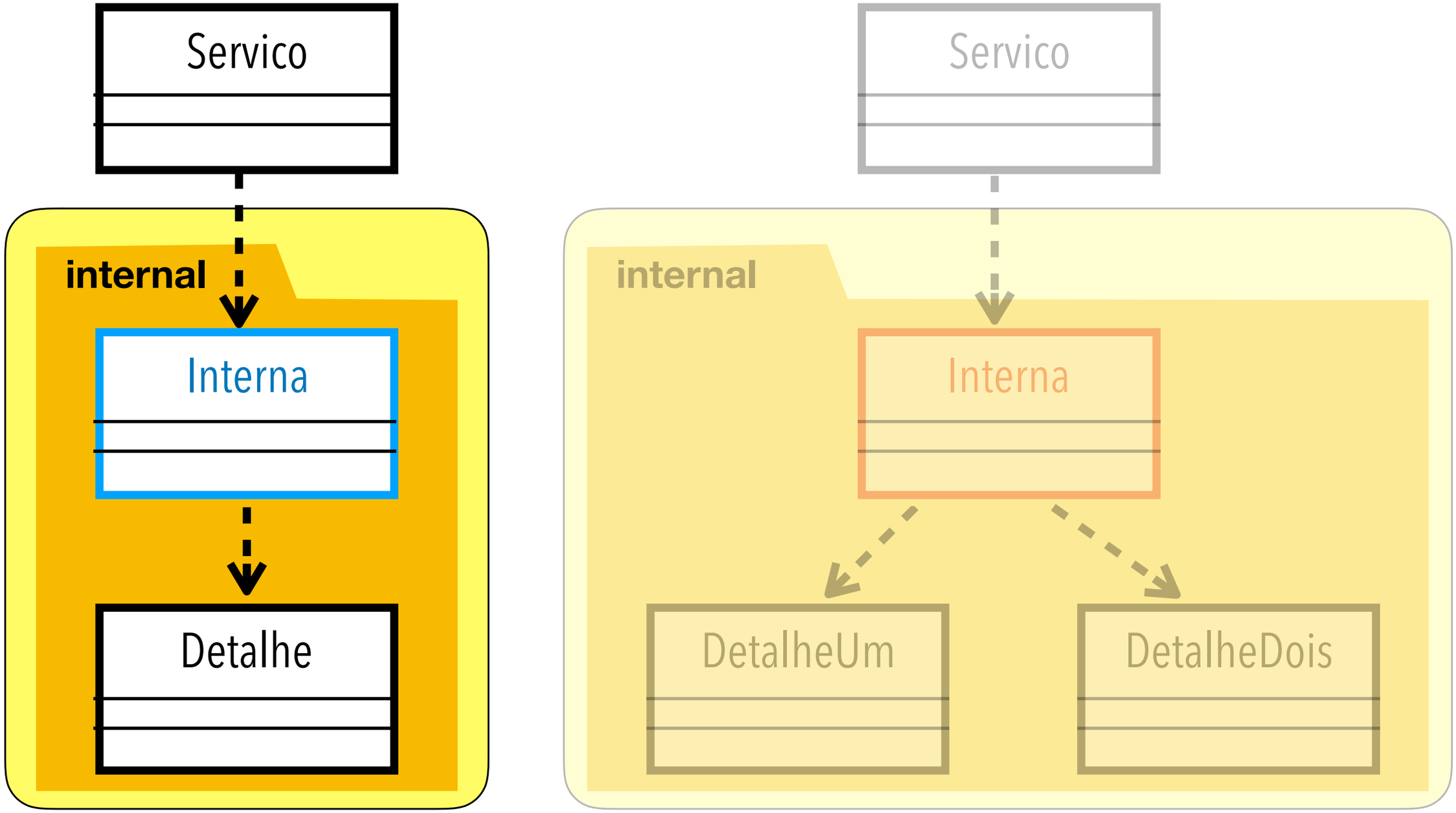


java -cp depend-1.0.jar;outros-1.x.jar;depend-1.1.jar -jar App.jar



java -cp depend-1.0.jar;outros-1.x.jar;depend-1.1.jar -jar App.jar

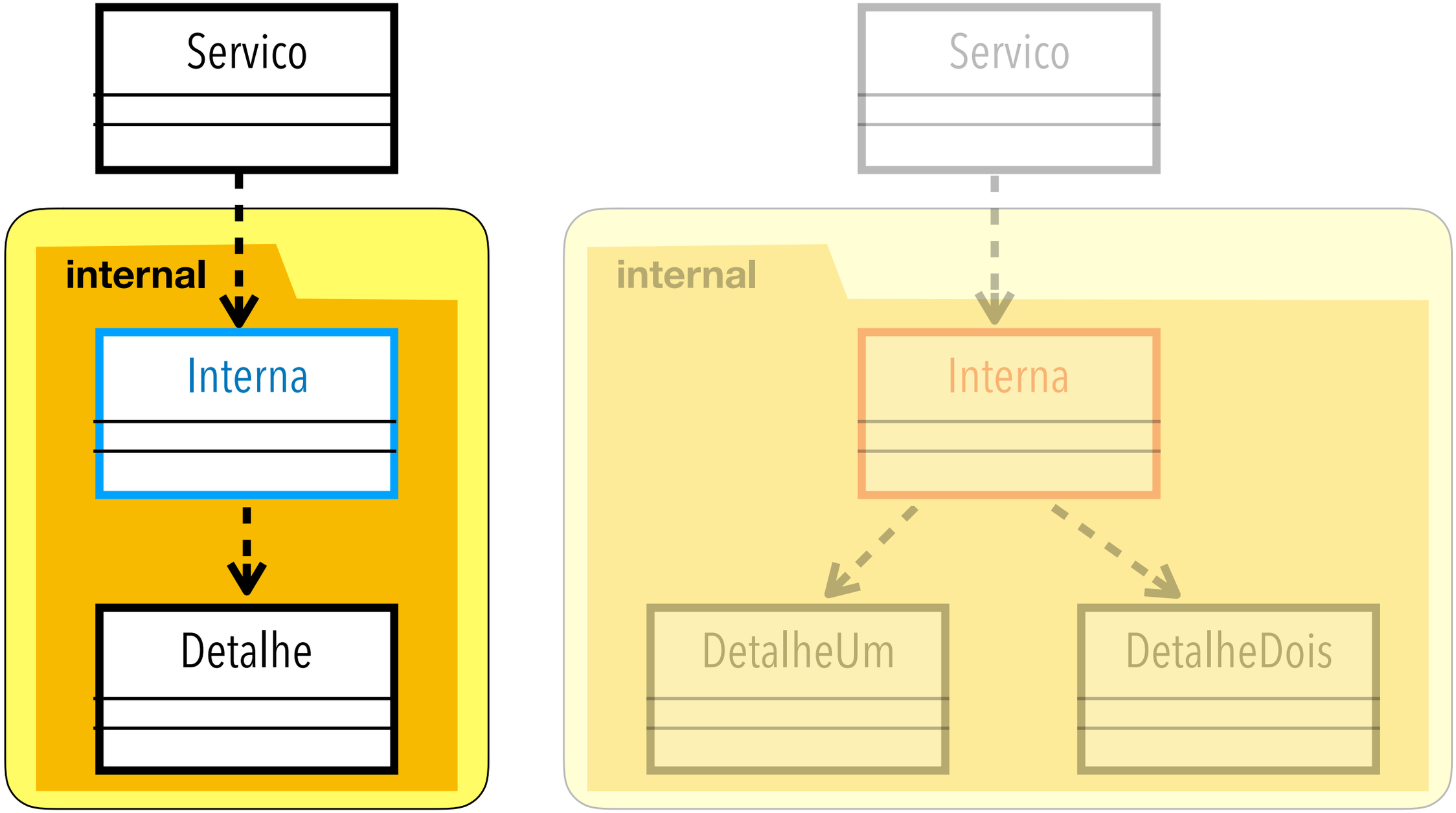
... Servico("dois") ...



java -cp depend-1.0.jar;outros-1.x.jar;depend-1.1.jar -jar App.jar

... Servico("dois") ...

java.lang.NoClassDefFoundError



O que é um módulo?

- **Conjunto de pacotes**
- Contém arquivo de metadados (**module-info.java**) e possivelmente outros recursos se necessário.
- Na JVM, **java.base** é o **módulo raiz** e não depende de nenhum outro módulo
- Outros módulos dependem de **java.base** por **default**

java --list-modules

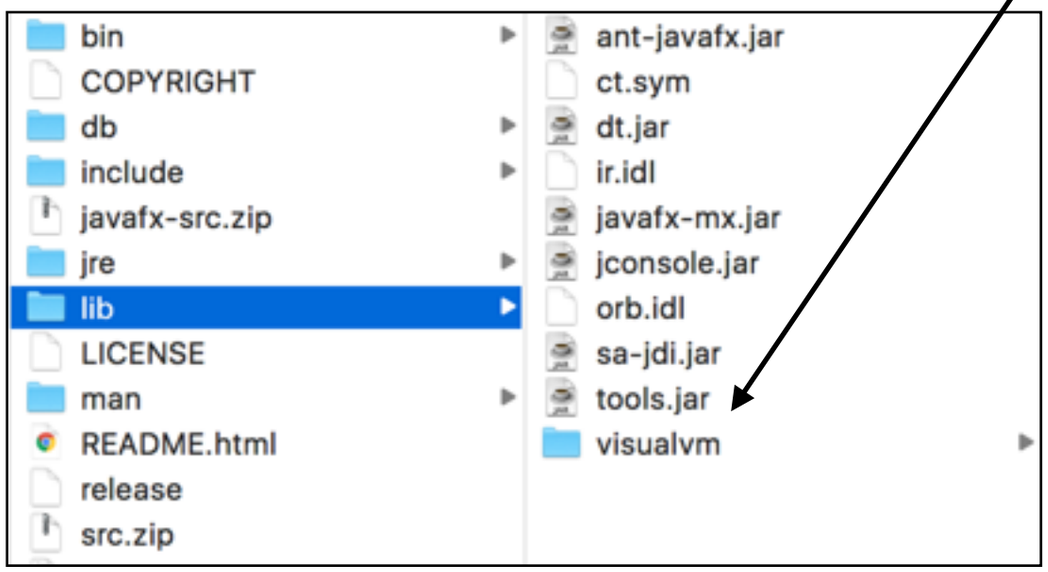
java.activation@9-ea
java.annotations.common@9-ea
java.base@9-ea
java.compiler@9-ea
java.corba@9-ea
java.datatransfer@9-ea
java.desktop@9-ea
java.instrument@9-ea
java.jnlp@9-ea
java.logging@9-ea
java.management@9-ea
java.naming@9-ea
java.prefs@9-ea
java.rmi@9-ea
java.scripting@9-ea
java.se@9-ea
java.se.ee@9-ea
java.security.jgss@9-ea
java.security.sasl@9-ea
java.smartcardio@9-ea
java.sql@9-ea
java.sql.rowset@9-ea
java.transaction@9-ea
java.xml@9-ea
java.xml.bind@9-ea
java.xml.crypto@9-ea
java.xml.ws@9-ea

javafx.base@9-ea
javafx.controls@9-ea
javafx.deploy@9-ea
javafx.fxml@9-ea
javafx.graphics@9-ea
javafx.media@9-ea
javafx.swing@9-ea
javafx.web@9-ea

jdk.accessibility@9-ea
jdk.attach@9-ea
jdk.charsets@9-ea
jdk.compiler@9-ea
jdk.crypto.cryptoki@9-ea
jdk.crypto.ec@9-ea
jdk.deploy@9-ea
jdk.deploy.controlpanel@9-ea
jdk.desktop@9-ea
jdk.dynalink@9-ea
jdk.editpad@9-ea
jdk.hotspot.agent@9-ea
jdk.httpserver@9-ea
jdk.incubator.httpclient@9-ea
jdk.internal.ed@9-ea
jdk.internal.le@9-ea
jdk.internal.opt@9-ea
jdk.jartool@9-ea
jdk.javadoc@9-ea
jdk.javaws@9-ea
jdk.jcmd@9-ea
jdk.jconsole@9-ea
jdk.jdeps@9-ea
jdk.jdi@9-ea
jdk.jdwp.agent@9-ea
jdk.jfr@9-ea
jdk.jlink@9-ea
jdk.jshell@9-ea
jdk.jsobject@9-ea
jdk.jstatd@9-ea
jdk.jvmstat@9-ea
jdk.localedata@9-ea
jdk.management@9-ea
jdk.naming.dns@9-ea
jdk.naming.rmi@9-ea
jdk.net@9-ea
jdk.pack@9-ea
jdk.packager@9-ea
jdk.packager.services@9-ea
jdk.plugin@9-ea
jdk.plugin.dom@9-ea
jdk.plugin.server@9-ea
jdk.policytool@9-ea
jdk.rmic@9-ea
jdk.scripting.nashorn@9-ea
jdk.scripting.nashorn.shell@9-ea
jdk.sctp@9-ea
jdk.security.auth@9-ea
jdk.security.jgss@9-ea
jdk.snmp@9-ea
jdk.unsupported@9-ea
jdk.vm.ci@9-ea
jdk.xml.bind@9-ea
jdk.xml.dom@9-ea
jdk.xml.ws@9-ea
jdk.zipfs@9-ea

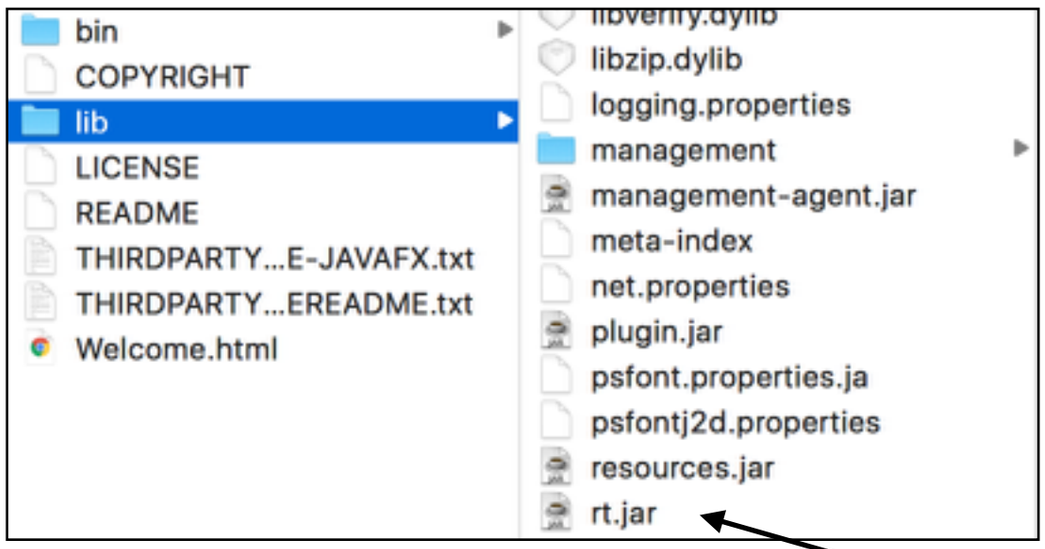
Estrutura do JDK

JDK 8



pacotes do JDK

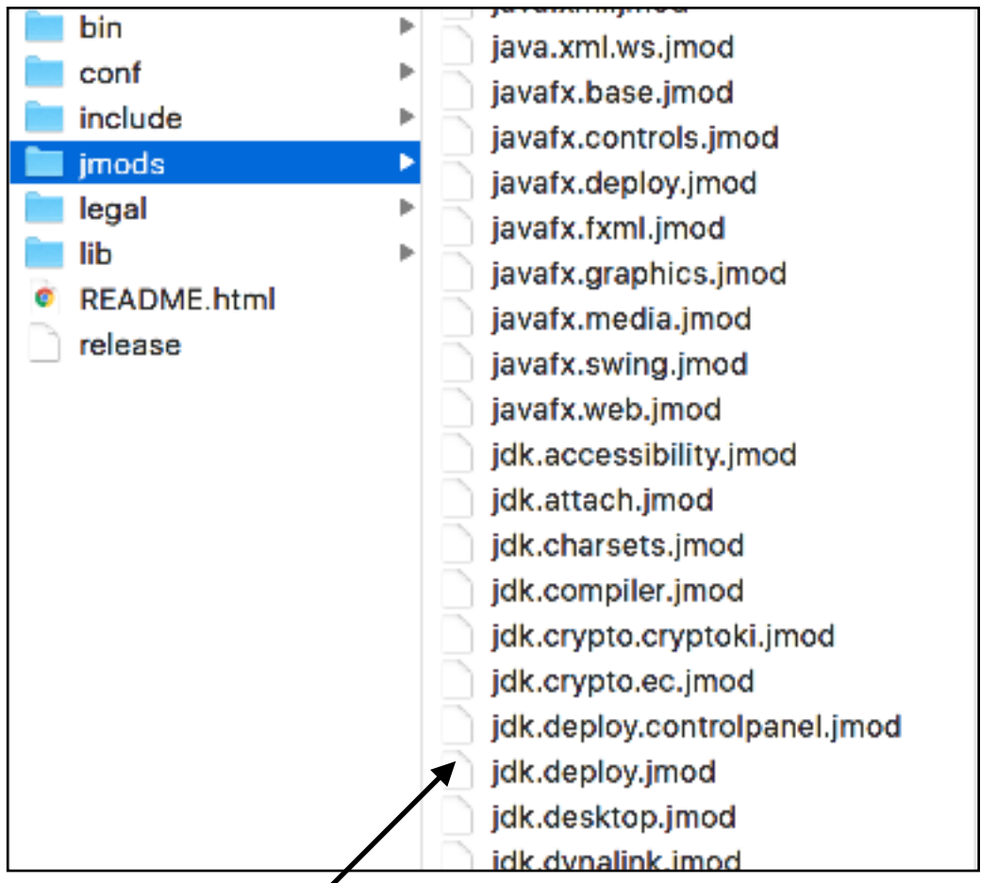
JDK 8/jre



pacotes do JRE

módulos do JRE

JDK 9

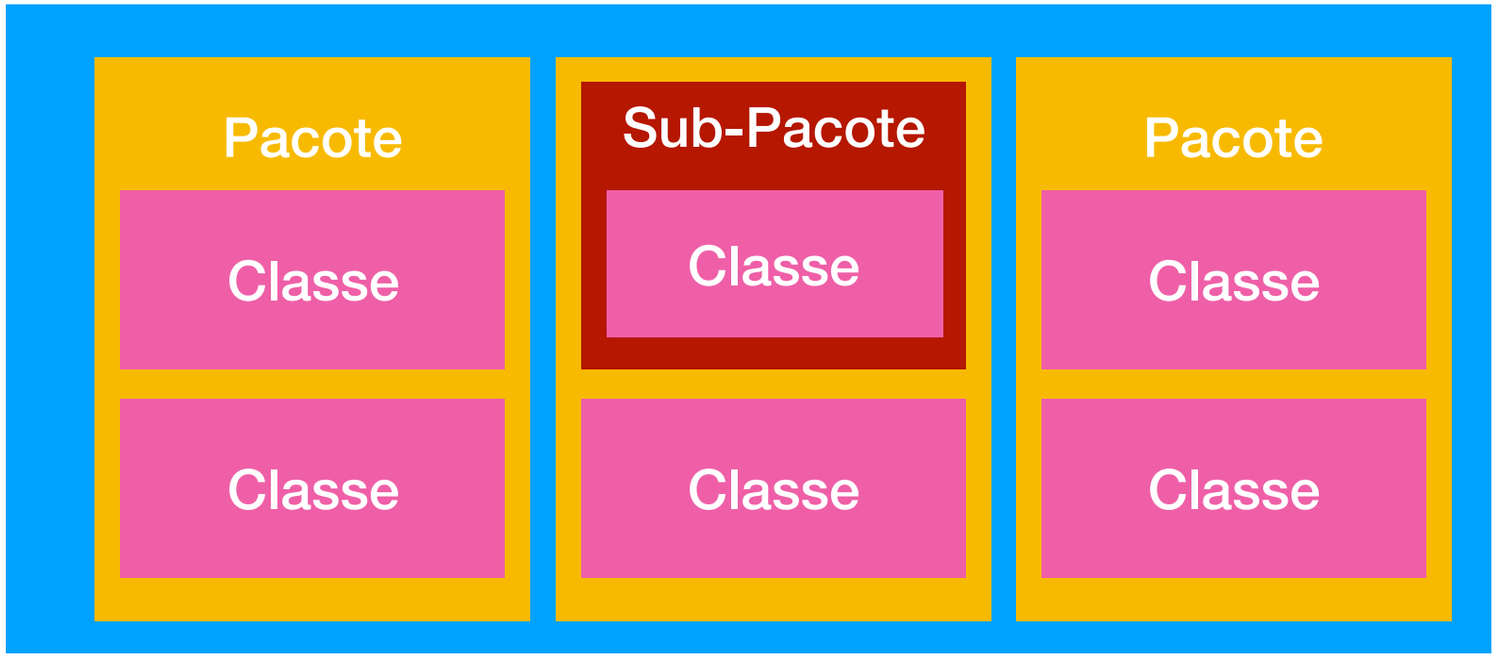


módulos do JDK

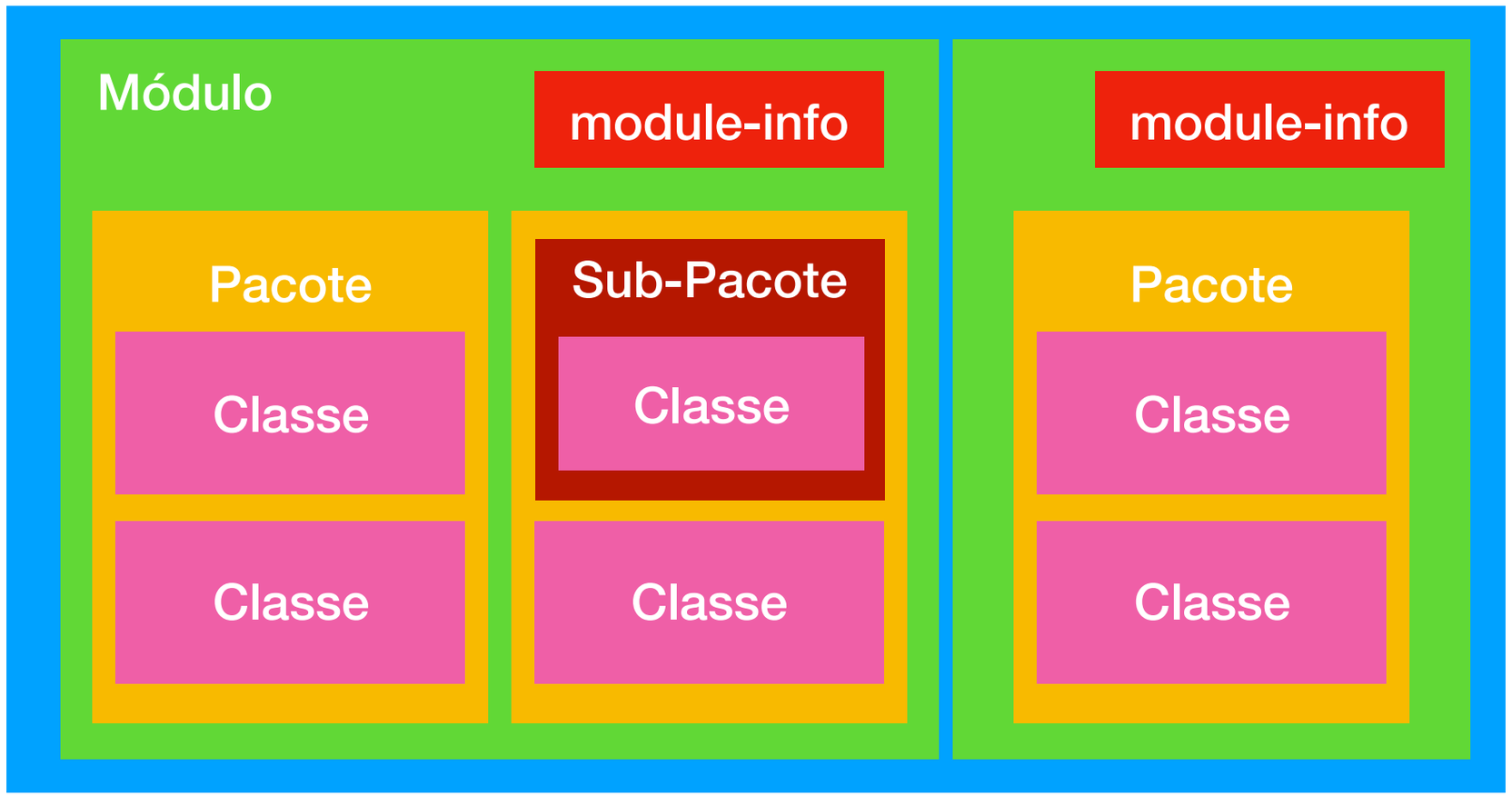
- Pouco mais de **90** módulos (1/3 java.*, 2/3 jdk.* e 8 módulos javafx)

Estrutura de uma aplicação

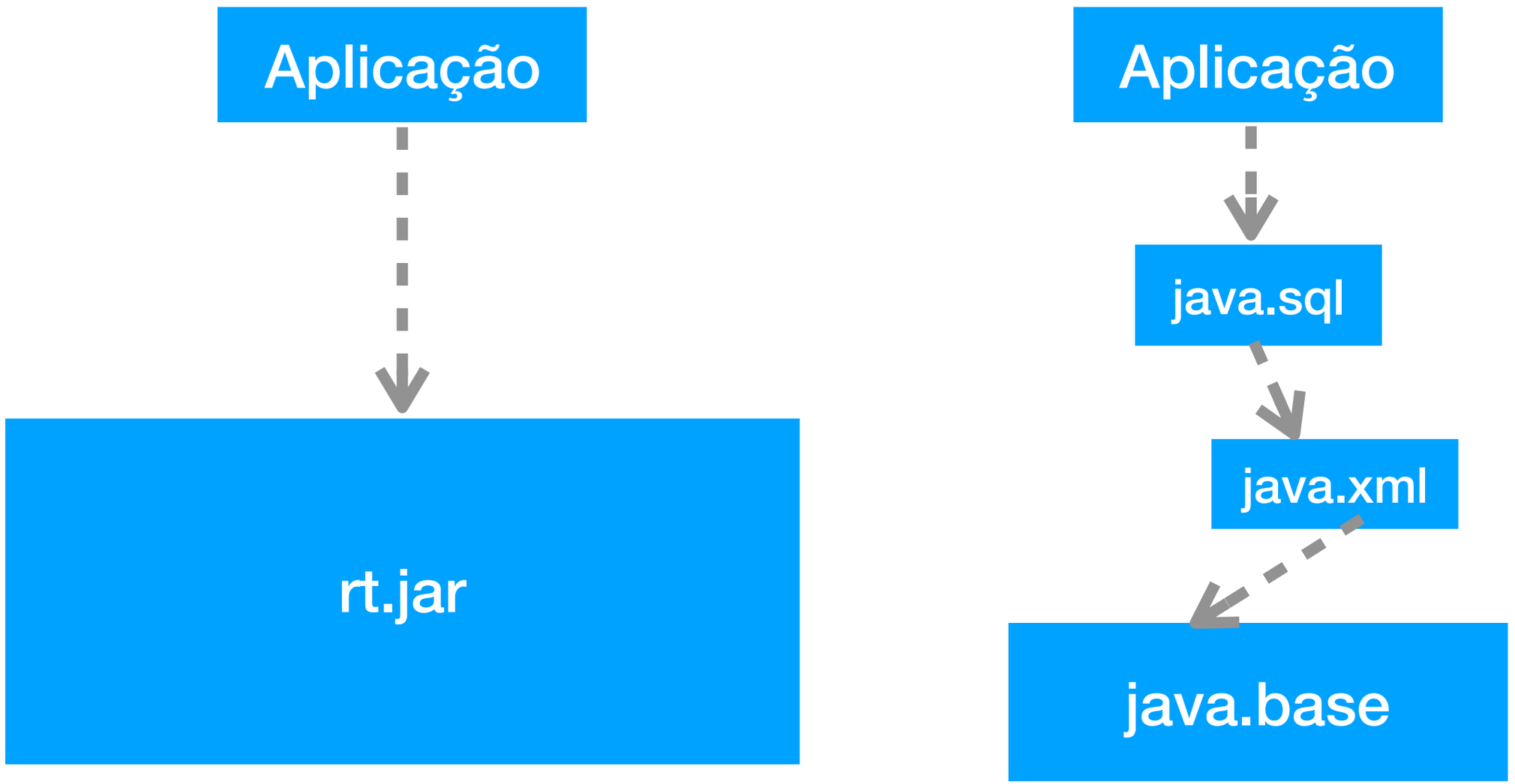
Java 8



Java 9



Aplicações ficam menores!

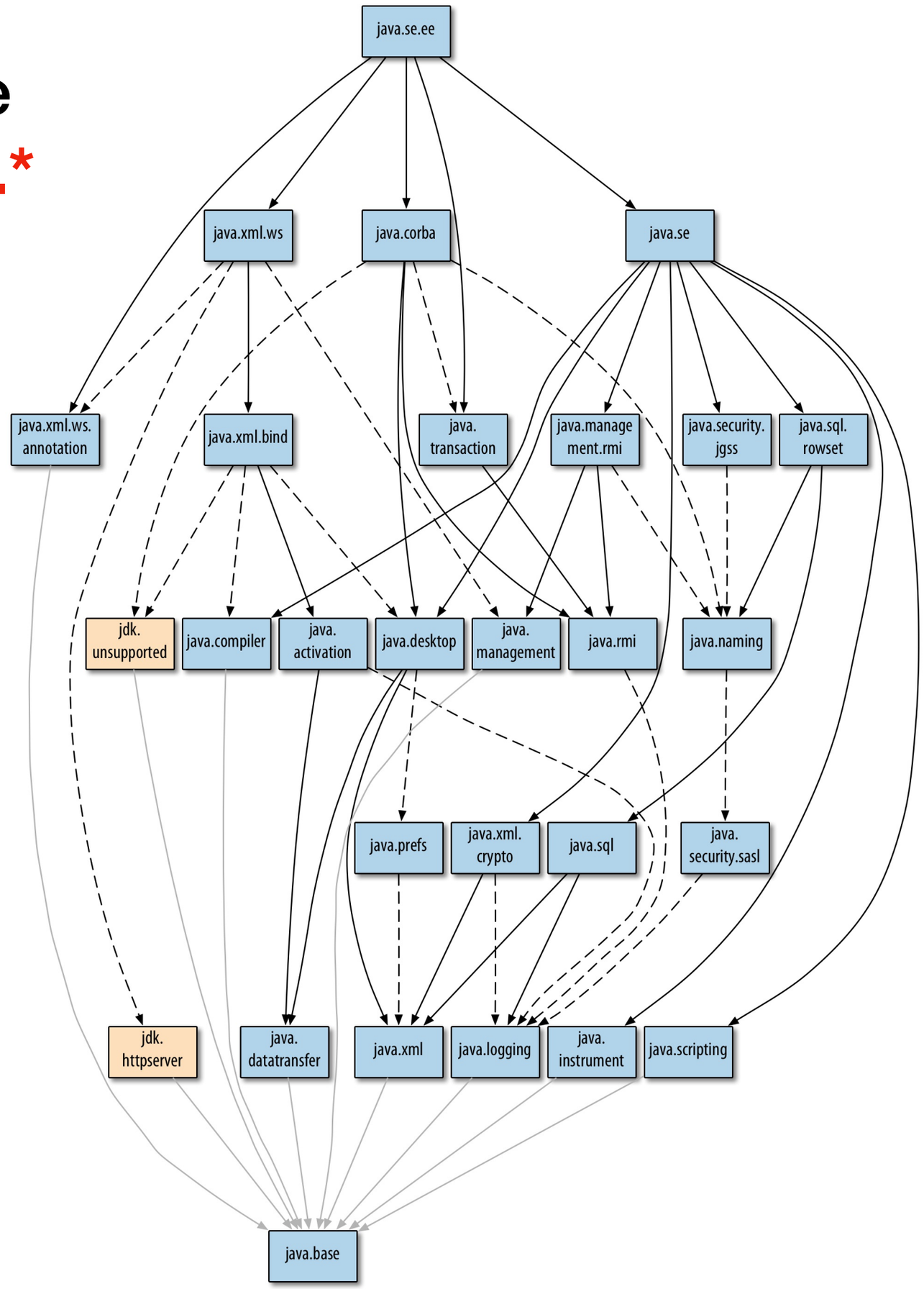


Como gerar um runtime

```
jlink --module-path $JDKMODS:$MYMODS
      --add-modules teste.mod
      --launcher exec
      --output myimage
```

```
$ myimage/bin/java -list-modules
java.base@9
java.logging@9
java.sql@9
java.xml@9
teste.mod
teste.mod.d1
```

Hierarquia de módulos java.*



module-info.java

- **Module Descriptor**: arquivo Java com meta-informação
- Define quais **pacotes** disponibiliza e de quais **módulos** depende
- Todo módulo possui um **module-info.java**
- Sintaxe: **`module`** ***`nome-do-módulo { ... }`***

```
module tdc.java9.exemplos {

}
```

requires

- Se um **módulo** precisa usar os **pacotes** contidos em outro módulo, ele precisa requerê-los:

```
module nome-do-modulo {
    requires outro-modulo;
}
```

- Um módulo pode requerer nenhum ou vários módulos

- É preciso saber **em que módulo está** um determinado pacote que se deseja usar!

exports

- Se os **pacotes** de um módulo foram criados para reuso, o módulo precisa exportá-los:

```
module nome-do-modulo {
    requires outro-modulo;
    exports pacote.nome-do-pacote;
}
```

- Um módulo pode exportar vários pacotes ou nenhum.

- Se um pacote não for exportado ele só pode ser usado dentro do módulo!

Legibilidade e Acessibilidade

- Readability e Acessibility

- **Legibilidade**: Se Módulo A depende diretamente Módulo B, o Módulo B **pode ser lido** (is readable) pelo módulo A

  - Cláusula **requires**

  - Ex: **java.base** pode ser lido por **java.xml**

  - Grafo (de readability) não deve conter ciclos!

- **Acessibilidade** (dos tipos públicos de um pacote): Módulo é legível e **exporta** o pacote

Módulos aprofundam as regras de encapsulamento

- Um módulo pode conter nenhuma ou várias cláusulas **requires** e **exports**

- Apenas classes e membros **públicos** em **pacotes exportados** são acessíveis

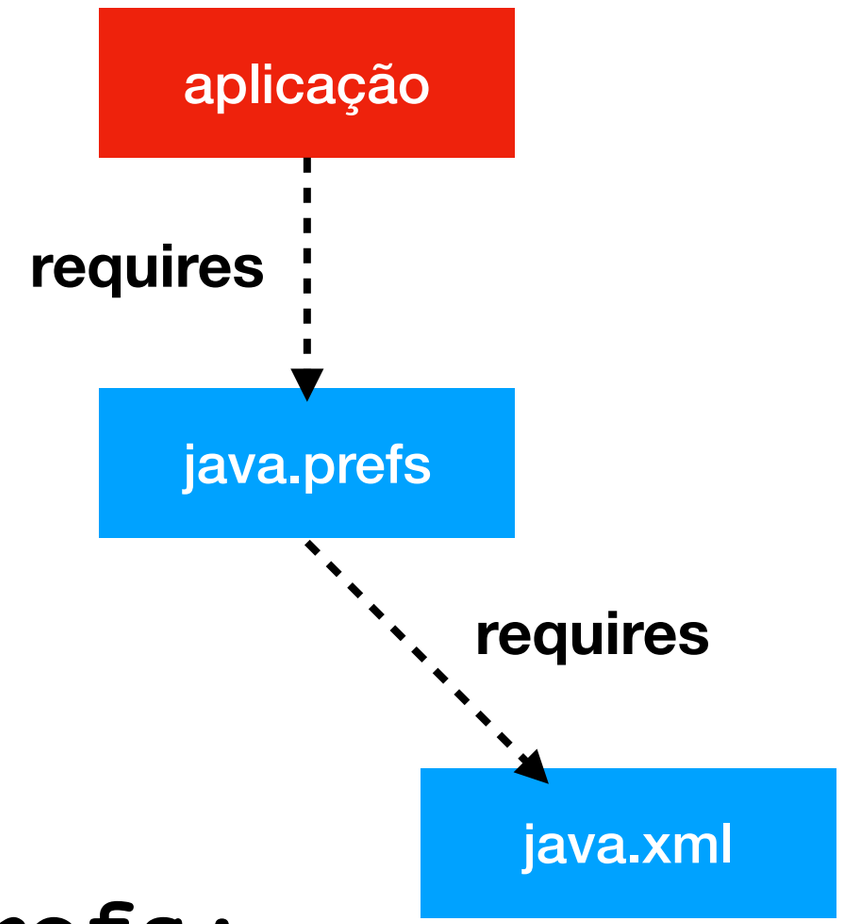
- O acesso acontece **apenas nos módulos que requerem** o módulo que exporta os pacotes

- Esse acesso refere-se a tempo de compilação e execução (há como controlar isto também)

Legibilidade

- A legibilidade **não é transitiva** por default

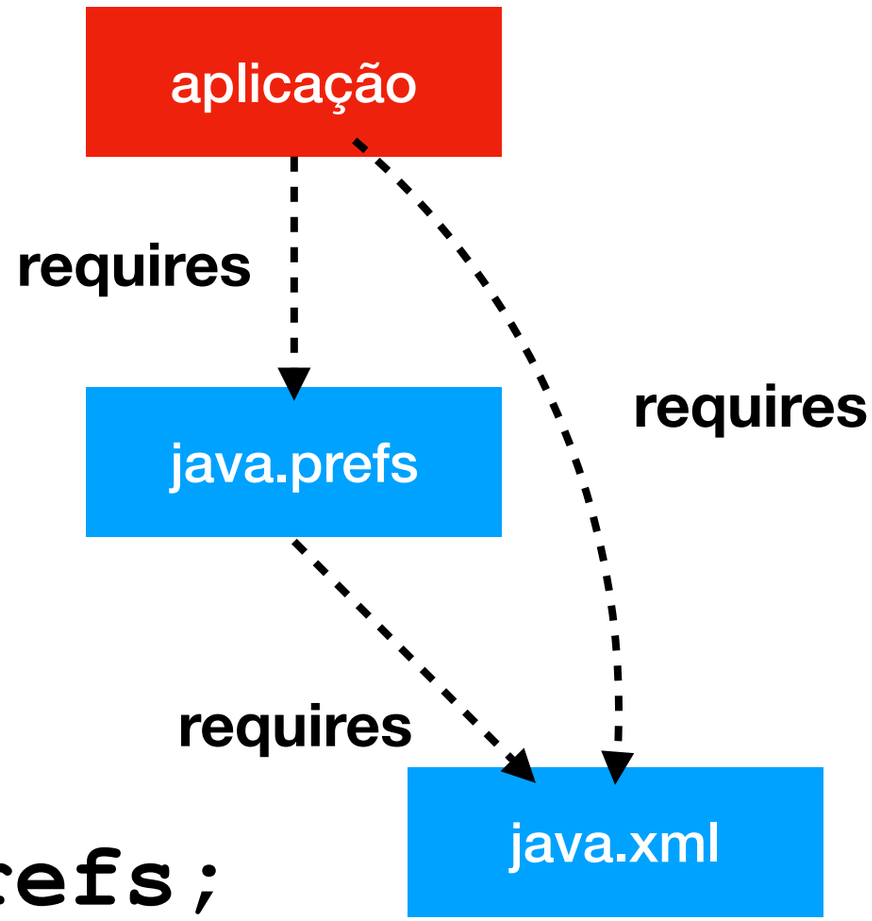
- Aplicação ao lado não tem acesso à API do módulo java.xml




```
module aplicacao {
    requires java.prefs;
}
```

Legibilidade

- Para ter acesso, a Aplicação **precisa também declarar requires** para o módulo java.xml




```
module aplicacao {
    requires java.prefs;
    requires java.xml;
}
```

java.base

- Raiz de todos os módulos
- Todo módulo **implicitamente** requer java.base
- Define e exporta todos os pacotes fundamentais da plataforma Java

```
module java.base {
    exports java.io;
    exports java.lang;
    exports java.lang.annotation;
    exports java.lang.invoke;
    exports java.lang.ref;
    exports java.lang.reflect;
    exports java.math;
    exports java.net;
    ...
}
```

Transitividade de dependências

- Dependência **não transitiva** - a dependência é necessária para dar suporte à **implementação** interna do módulo

- Dependência **transitiva** - a dependência é necessária para dar suporte à **API** do módulo

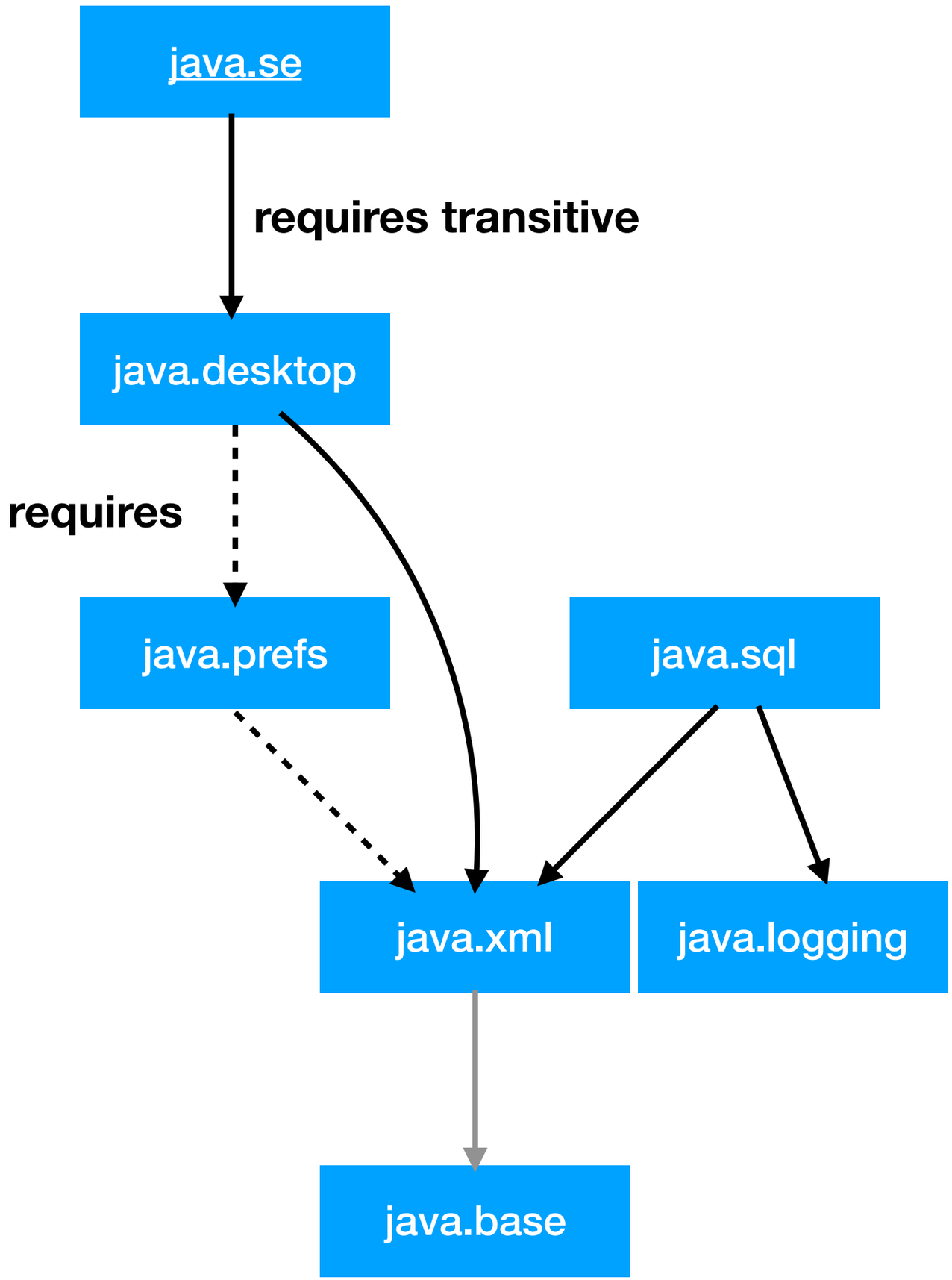
requires transitive

- Legibilidade transitiva (transitive/implied readability)
- Permite a extensão da legibilidade para módulos adicionais dos quais o módulo depende
- Ex:

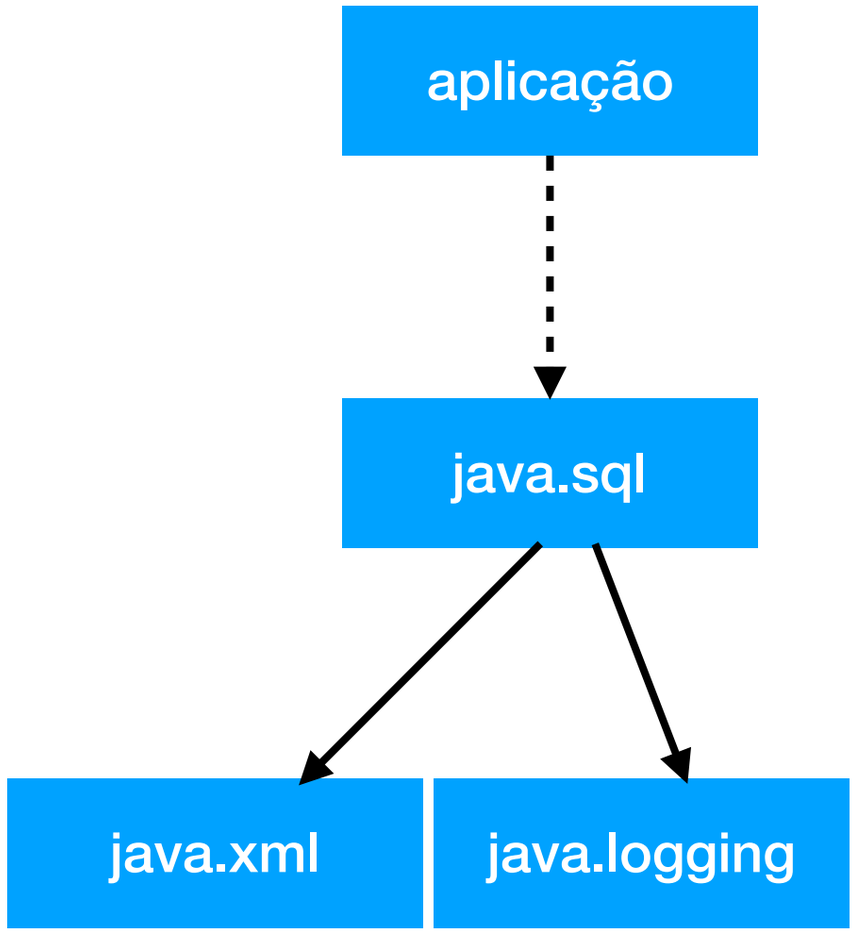
```
module java.sql {
    requires transitive java.logging;
    requires transitive java.xml;
    exports java.sql;
    exports javax.sql;
    exports javax.transaction.xa;
}
```

Legibilidade transitiva

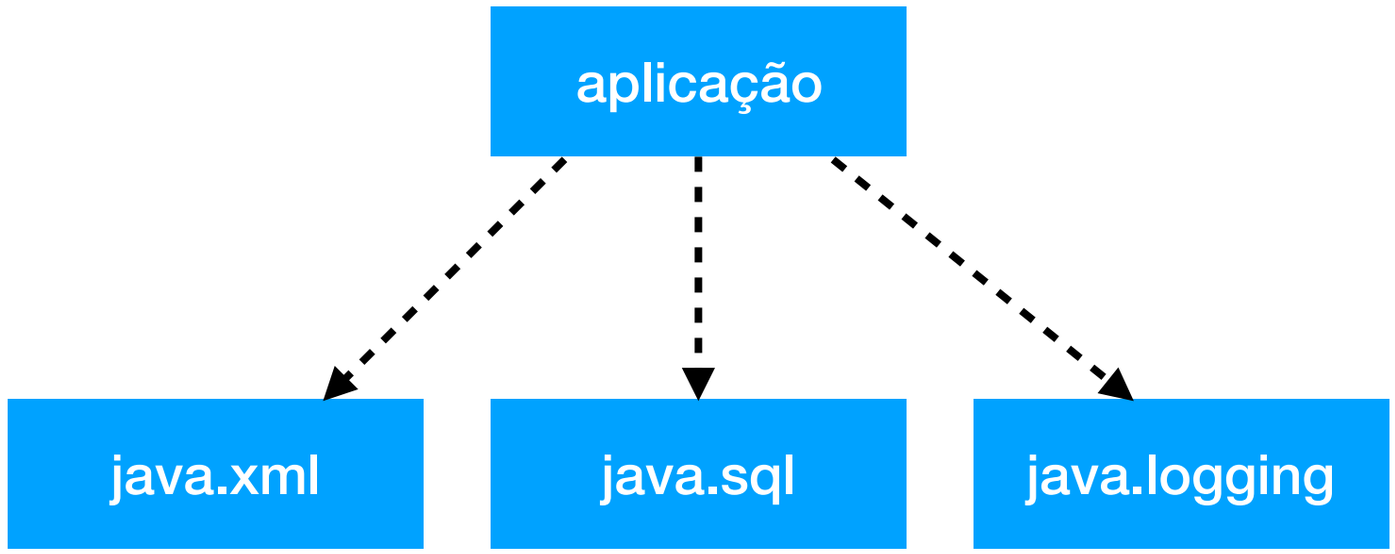
- Setas sólidas representam ligações transitivas



requires transitive



Declaração



Efeito

java.se e java.se.ee

- Módulos **agregadores** (agrupam outros módulos)

- Declaram todos os módulos descendentes com **requires transitive** (permite que a aplicação use tudo)

```
module java.se  {
    requires transitive java.desktop;
    requires transitive java.sql;
    ...
}
```

- Não é boa prática usá-los sempre

Module Path

- Local onde se encontram **módulos** (análogo ao classpath)

- É diferente do classpath: não localiza classes, mas **módulos inteiros** - dependências são especificadas com clareza em metadados

- Dependências transitivas são computadas na inicialização e construção do **grafo de módulos**

Módulos sem nome

- Tentativa de carregar uma classe que não se encontra em nenhum módulo conhecido
- Define o módulo raiz como java.se (mas não inclui java.se.ee - se aplicação usar JAXB ou CORBA não vai funcionar)
- Sistema carrega do **classpath** (se possível) e inclui o tipo como parte do **módulo sem nome**
- O módulo sem nome exporta automaticamente todos os seus pacotes
- Módulos com nome não podem declarar dependência em módulos sem nome
- Permite a execução e uso de aplicações Java SE 8

Módulos automáticos

- JARs incluídos no **modulepath** criam um módulo de mesmo nome

- **Módulos automáticos** são módulos com nome (o nome é definido automaticamente)

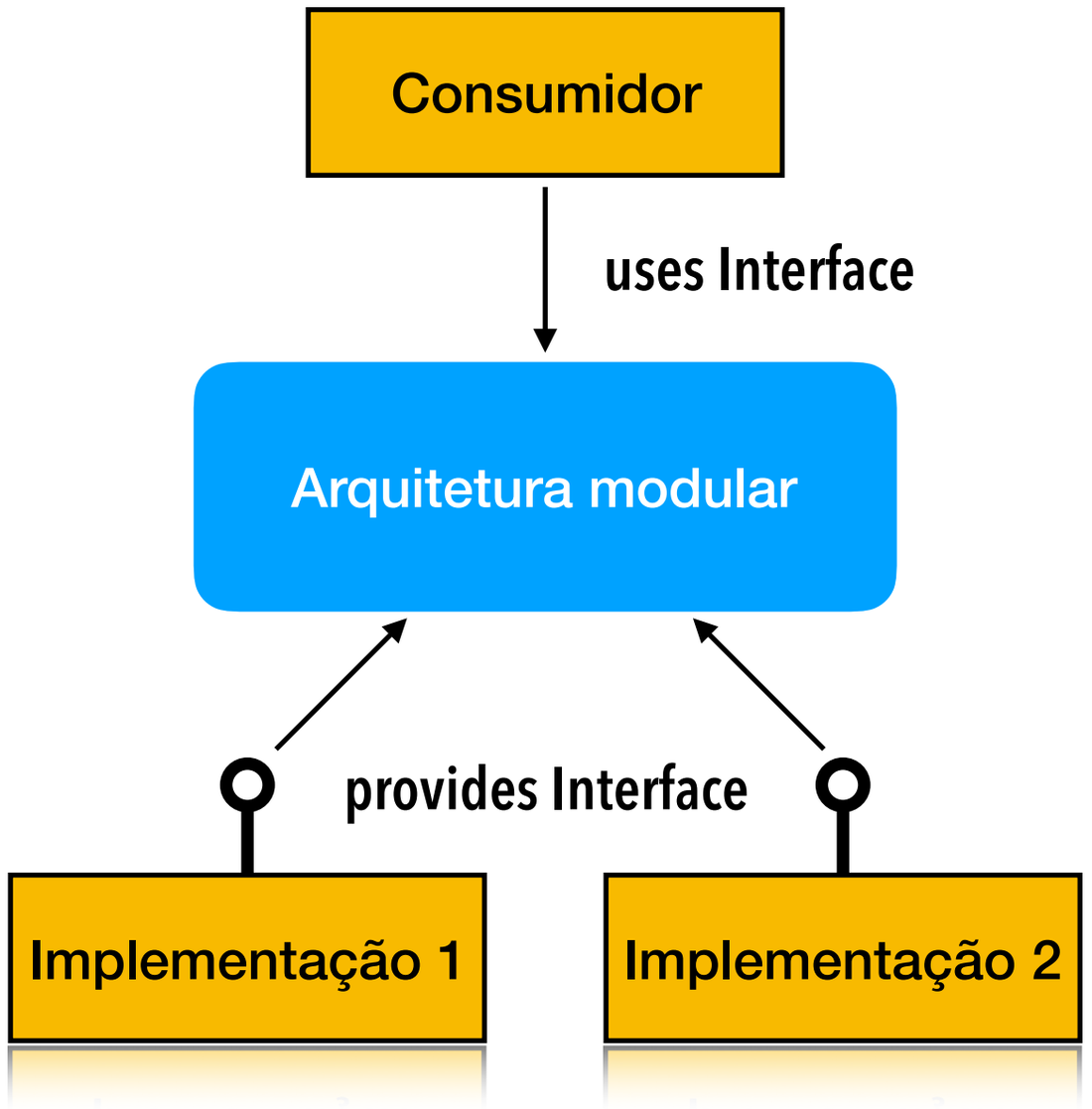
- Permite que uma aplicação modular use JARs que não têm module-info.jar

- Exportam **todos** os seus pacotes e requer **todos** os outros pacotes do modulepath.

- Pode-se declarar dependências de módulos automáticos

Serviços

- Service Provider Interface (SPI)
- Usa API **ServiceLoader** (foi adaptada para módulos)
- Módulo que oferece o serviço declara **provides** interface-do-servico **with** implementacao
- Módulo que consome o serviço declara uses interface-do-servico




```
ServiceLoader<Interface> servico = ServiceLoader.load(Interface.class);
Interface obj = servico.next(); // instancia o serviço + cache (reload() renova)
```

Exemplo usando SPI

- Módulo que usa provedor, declara **uses**

```
module java.sql {
    ...
    exports java.sql;
    exports javax.sql;
    uses java.sql.Driver;
}
```

- Módulo que fornece, declara **provides**

```
module com.mysql.jdbc {
    requires java.sql;
    exports com.mysql.jdbc;
    provides java.sql.Driver with com.mysql.jdbc.Driver;
}
```

requires static

- Permite que um módulo dependa de outro para compilação, mas não em tempo de execução
- Existe risco de NoClassDefFoundError

```
module stats.dados {
    requires stats.dados;
    requires static graficos.3d;
    exports stats.leituras;
}
```

Qualified exports (to)

- Restringe a exportação de pacotes a módulos específicos

```
module casa {
    exports casa.garagem to casa.dono;
    exports casa.sala;
}
```

- Apenas **casa.dono** pode acessar casa.garagem (campos públicos)

Dificuldades em sistemas legados

- **Frameworks** que dependem de reflection (a maioria) têm problemas para acessar dados privativos em módulos

- Antes de Java 9 Reflection tudo era possível (**setAccessible(true)**)

- Solução

  - Pacotes e módulos abertos

  - Opções da JVM

Pacotes abertos

- Libera acesso total (permite reflection e acesso a membros não-públicos) em um pacote específico (apenas em tempo de execução)

```
module nome-do-modulo {
    opens pacote;
}
```

- Também suporta **to**, para restringir o acesso

```
module nome-do-modulo {
    opens pacote to mod-amigo;
}
```

Módulos abertos

- Libera todos os pacotes do módulo - todos tem acesso a todos os membros de todos os pacotes

```
open module nome-do-modulo {
    ...
}
```

Algumas opções da app java

```
java

   --module-path caminho

   --add-modules

   --add-opens

   -- module

   --illegal-access=allow
```

Ferramenta jdeps

- Permite saber quais os módulos requeridos por uma classe ou módulo

- Ferramenta fundamental para migração

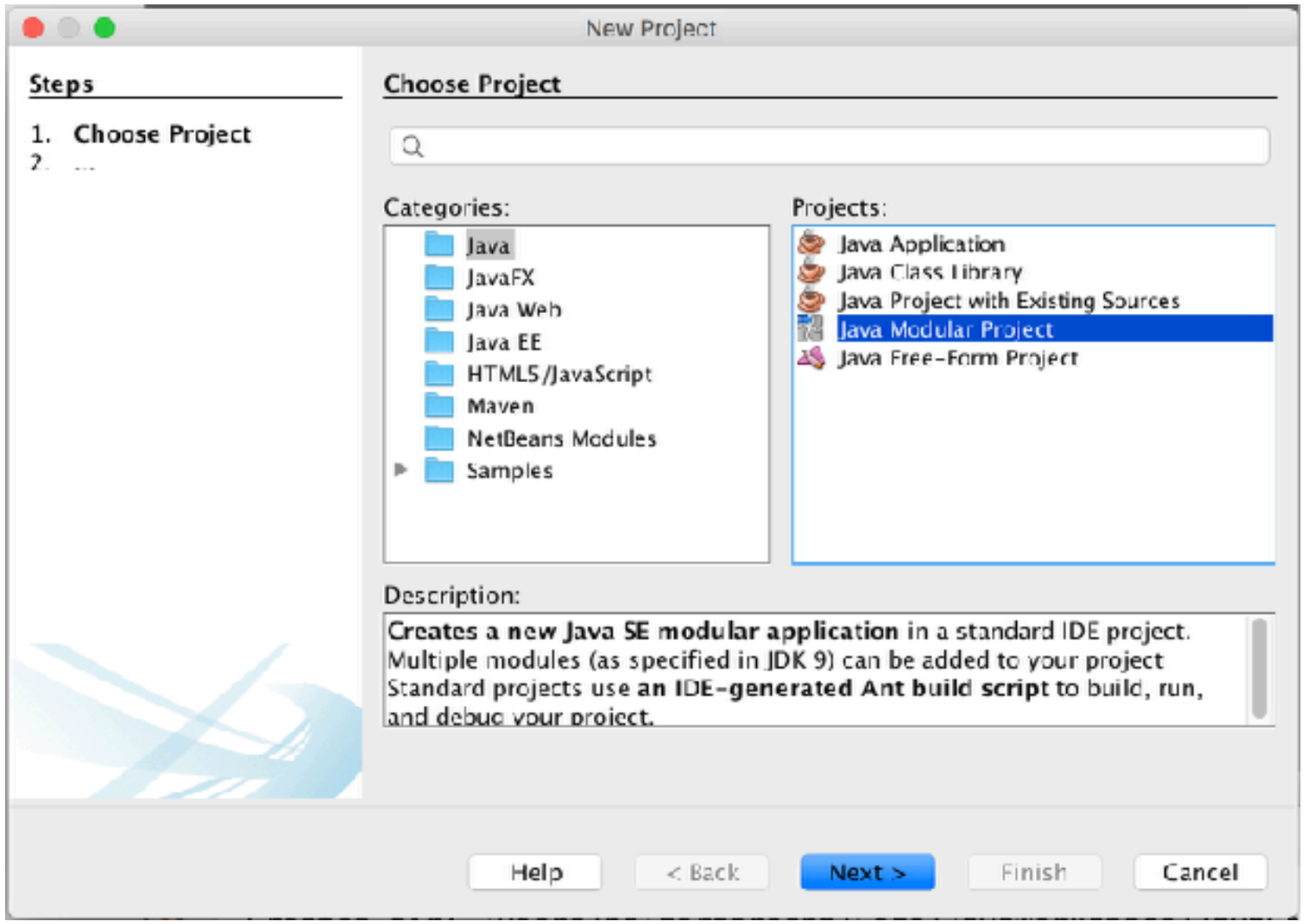
```
>jdeps -jdkinternals Sample.class
Sample.class -> JDK removed internal API
   Sample  -> sun.misc.BASE64Encoder  JDK internal API (JDK removed internal API)

Warning: JDK internal APIs are unsupported and private to JDK implementation that are
subject to be removed or changed incompatibly and could break your application.
Please modify your code to eliminate dependency on any JDK internal APIs.
For the most recent update on JDK internal API replacements, please check:
https://wiki.openjdk.java.net/display/JDK8/Java+Dependency+Analysis+Tool

JDK Internal API                          Suggested Replacement
----------------                          ---------------------
sun.misc.BASE64Encoder                    Use java.util.Base64 @since 1.8
```

Projeto modular no NetBeans

- Primeiro instale o JDK 9 no seu ambiente, depois baixe um Development build
    - http://bits.netbeans.org/download/trunk/nightly/latest/

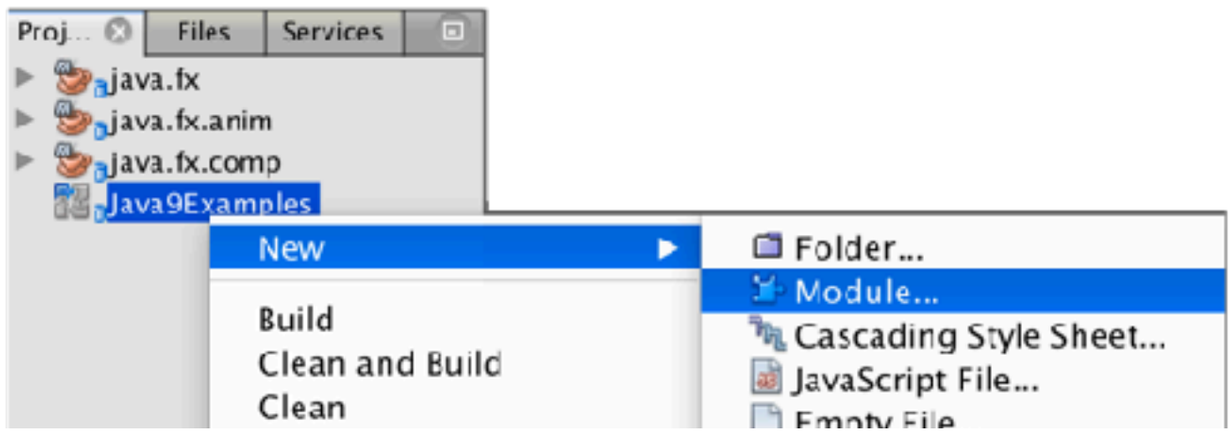
Projeto modular no NetBeans

- Depois crie um novo projeto modular

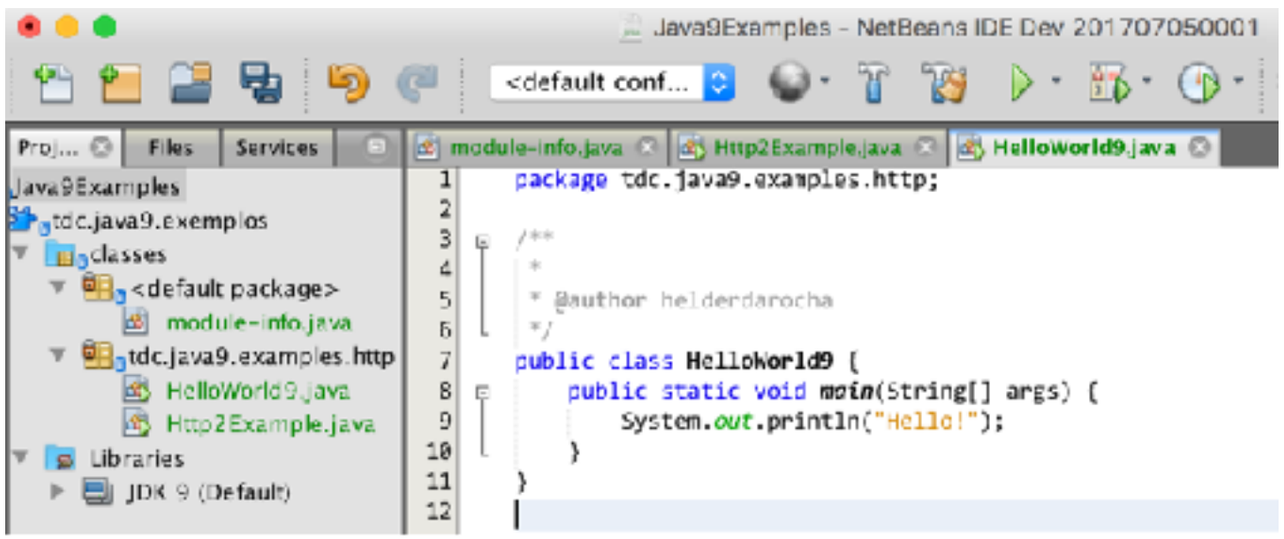


Projeto modular no NetBeans

- Crie pelo menos um módulo



- Dentro do módulo será criado uma pasta de código-fonte contendo um arquivo **module-info.java**. Na pasta você poderá criar pacotes e classes normalmente



Conclusões

- A arquitetura modular é a principal novidade do Java 9 e tem **grande impacto** na forma de desenvolver aplicações

- Os benefícios são muitos e **vale a pena modularizar**, mas a migração não será tão fácil quanto java 8 (usar Java EE com módulos ainda é difícil)

- Há diversos mecanismos para facilitar a migração e permitir que aplicações que não usam módulos trabalhem juntas

THE DEVELOPER'S CONFERENCE

Julho/2017

A arquitetura modular do Java

Baixe esta palestra depois do TDC em

http://www.argonavis.com.br/download/tdc_2017_modulosJava.html

+Links para outros recursos, código, referências

helder da rocha

helder@summa.com.br