THE DEVELOPER'S CONFERENCE

julho 2017

novidades do java nove

summa Technology + Business

helder da rocha

Quem sou eu? Who am I? Кто я?

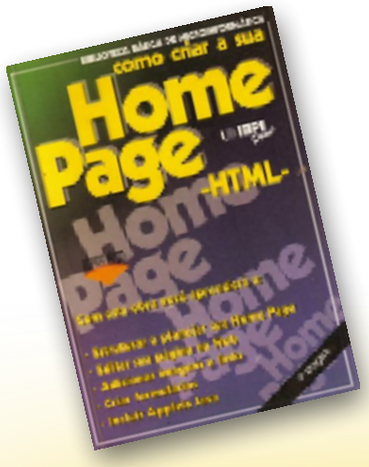
Helder da **Rocha**

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)
- Lista de **91** features http://openjdk.java.net/projects/jdk9/
- Último release candidate já está disponível
- Lançamento previsto: **21 de setembro**

História das versões Java

- JDK Alfa e Beta: **1994**
- JDK 1.0 **1996** (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?

- Antes de 20/09, baixe um early-release em http://jdk.java.net/9/

- Para programar
  - JDK + linha de comando
  - NetBeans
  - Eclipse
  - IntelliJ

9 novidades

(na verdade um pouco mais)

- 1 Arquitetura Modular
- 2 Coleções
- 3 Concorrência
- 4 Interfaces
- 5 Process API
- 6 Exceções
- 7 Performance
- 8 JShell
- 9 Outras novidades

1

Módulos

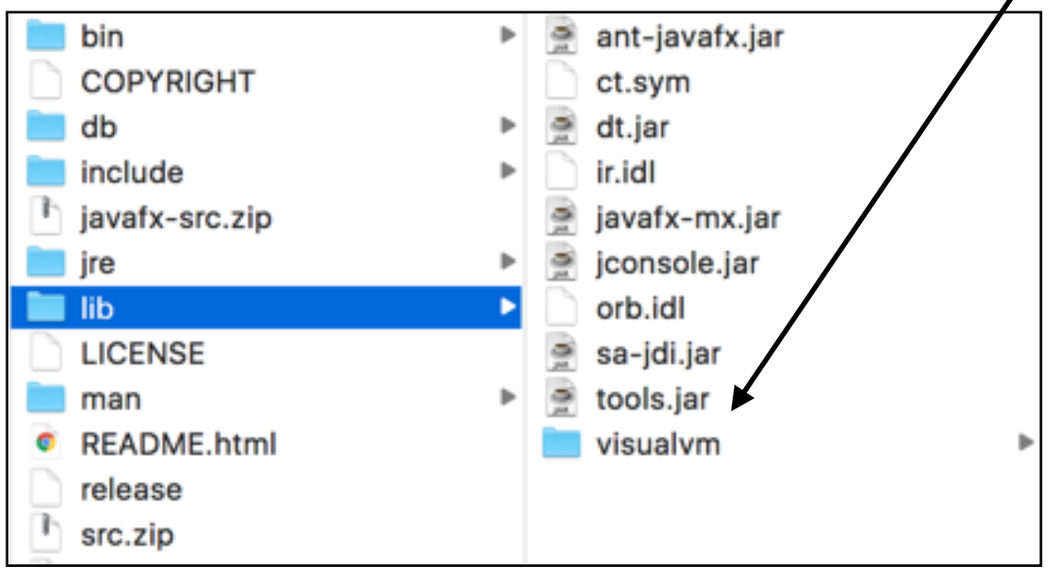
JSR 376

1 Arquitetura modular

- Aumentar a **coesão**, **reuso**, **encapsulamento**, **eficiência**
- Modularização em Java 9:
  - **JDK e JRE**: divide JDK (tools.jar) e Runtime (rt.jar) em vários 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**

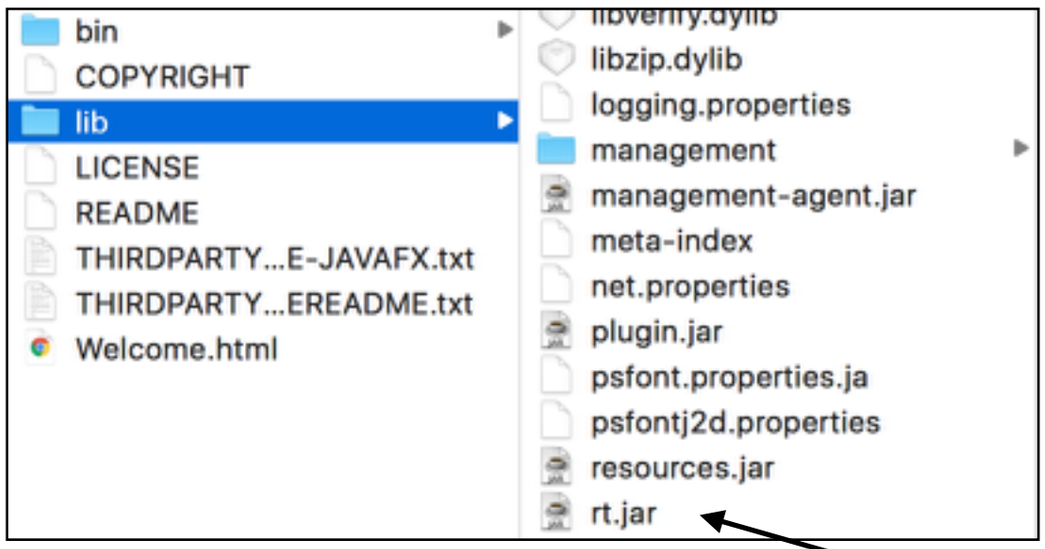
Estrutura do JDK

JDK 8



pacotes do JDK

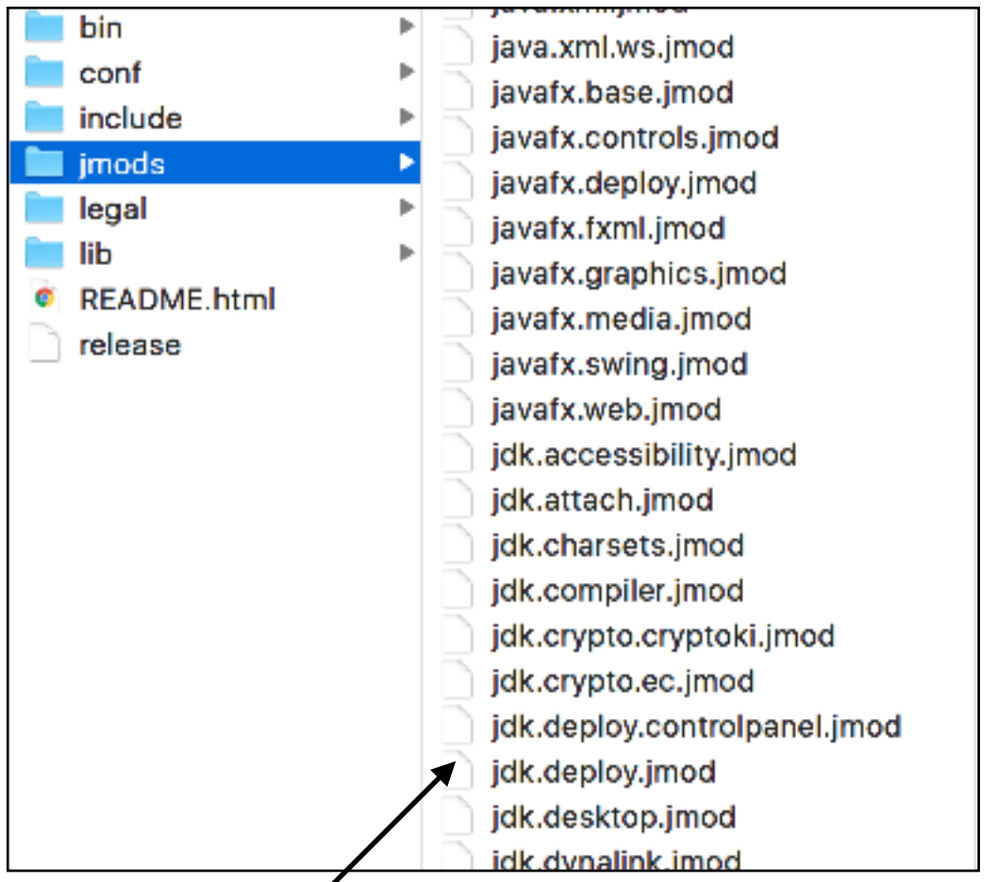
JDK 8/jre



pacotes do JRE

módulos do JRE

JDK 9



módulos do JDK

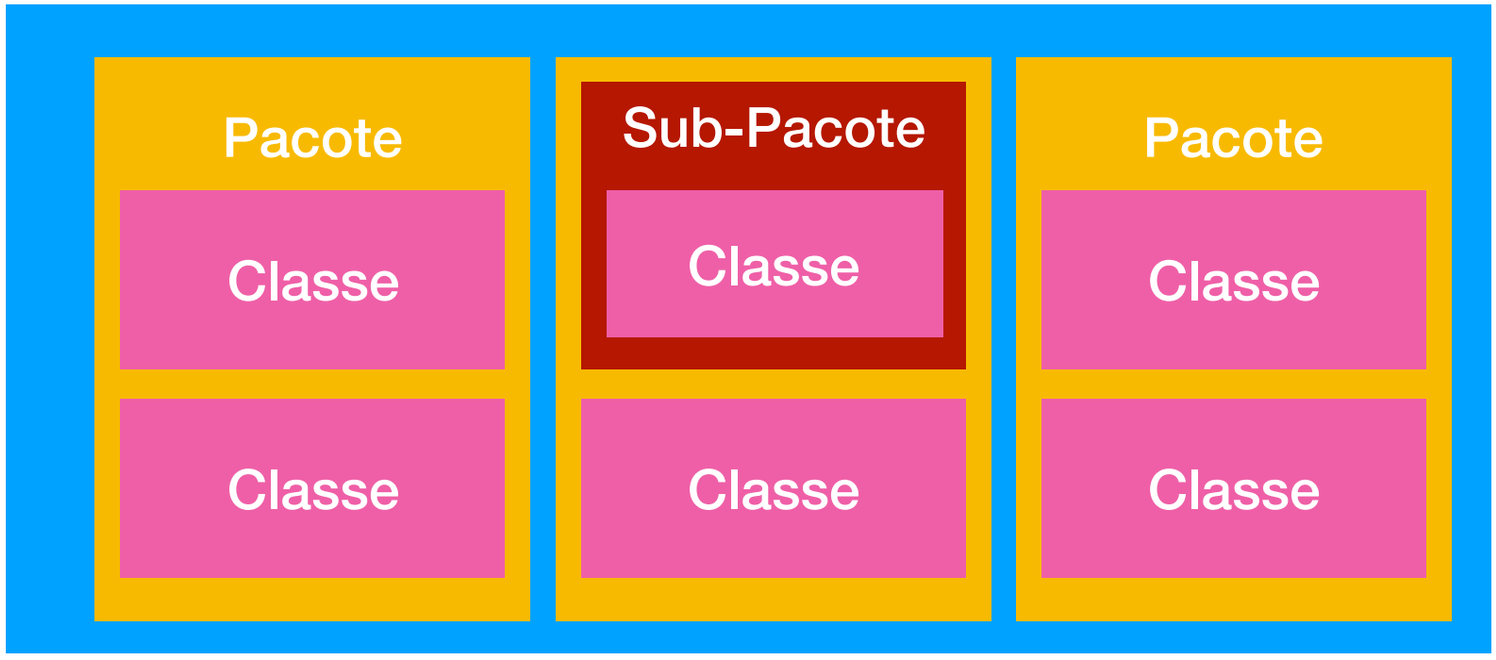
- Pouco mais de **90** módulos (1/3 java.*, 2/3 jdk.* e 8 módulos javafx)

O que é um módulo?

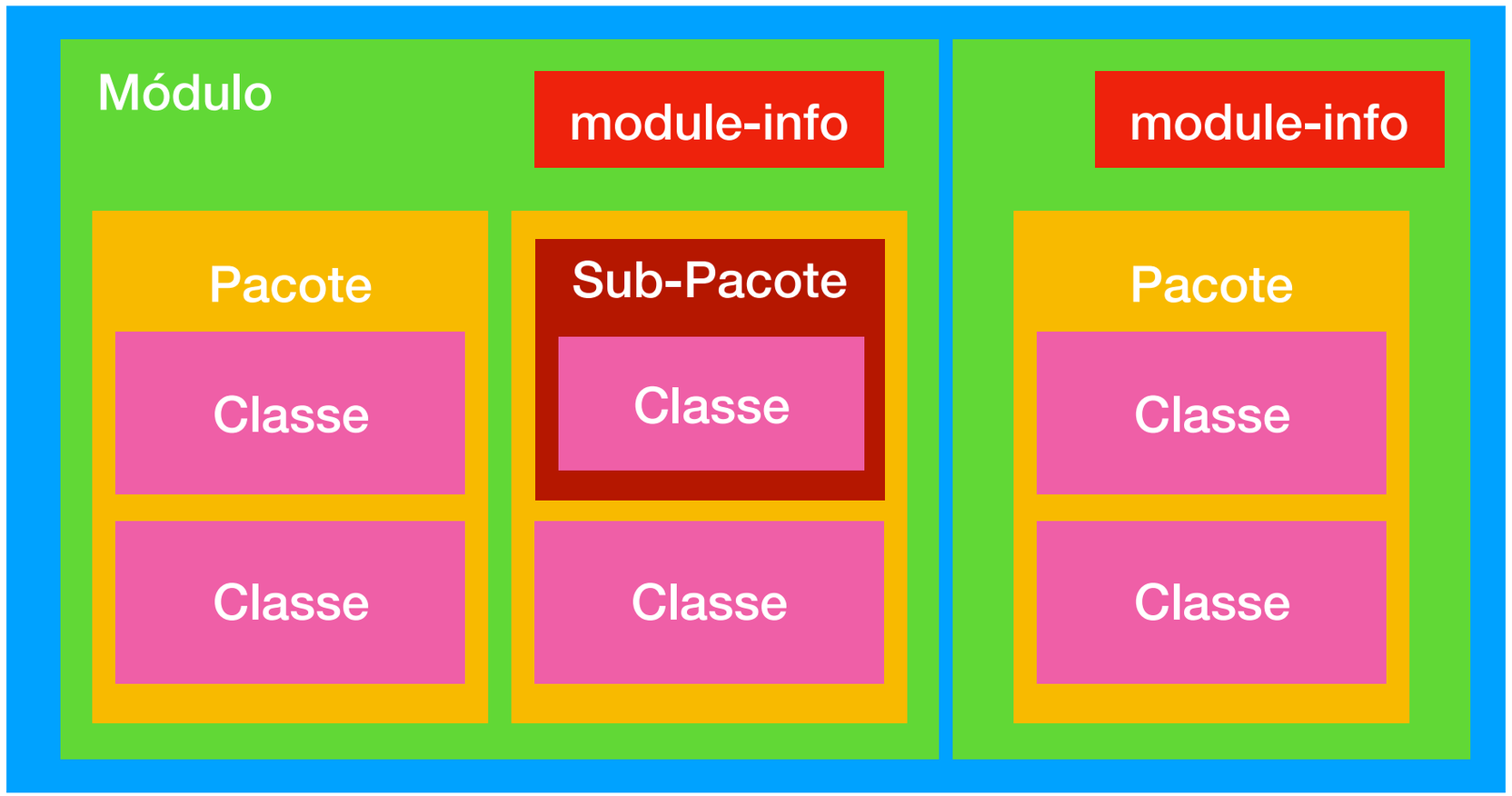
- **Conjunto de pacotes**
- Contém arquivo de metadados (**module-info.java**) e possivelmente outros recursos se necessário.
- Na JVM, **java.base** é o **módulo raiz** e não depende de nenhum outro módulo
- Outros módulos dependem de **java.base** por **default**

Estrutura de uma aplicação

Java 8



Java 9



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 {

}
```

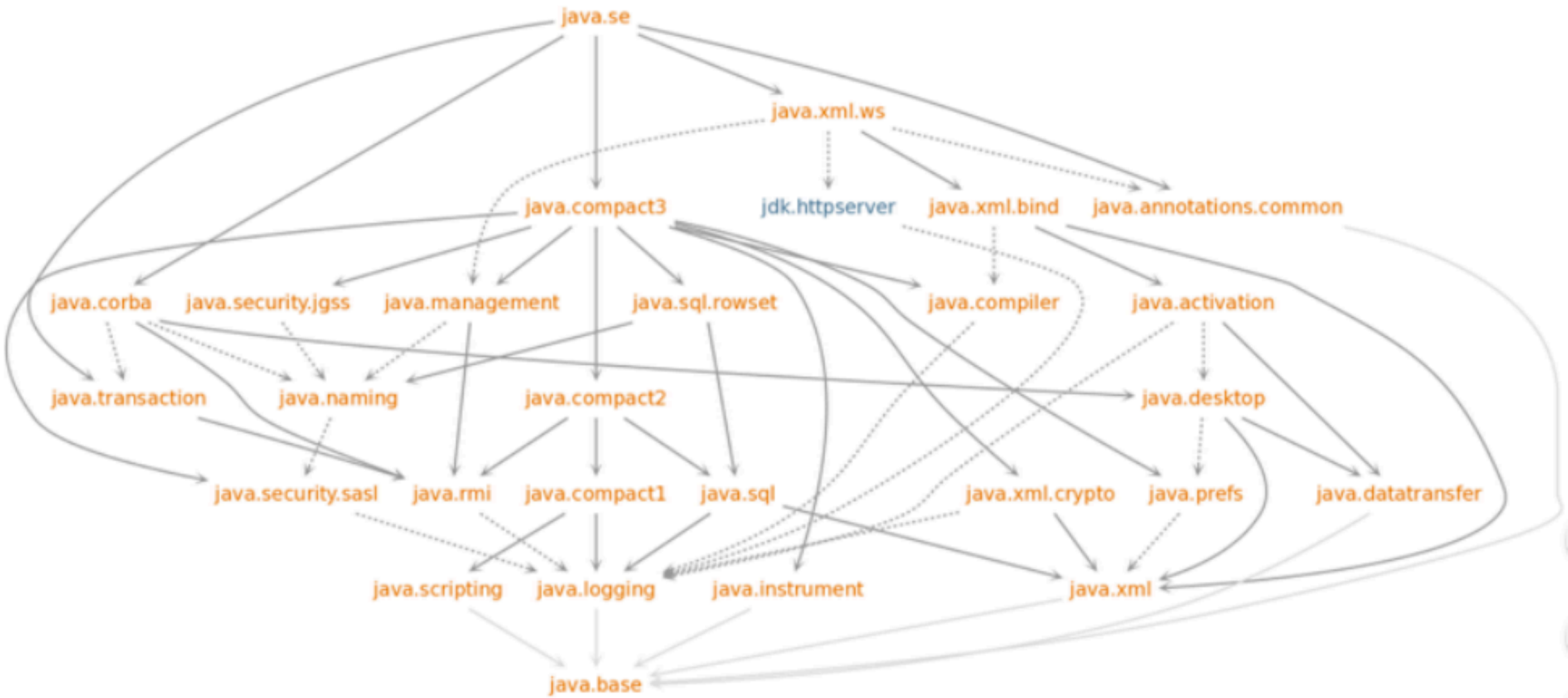
java --list-modules

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

Hierarquia de módulos java.*



- Pode haver implementações da plataforma que não contém todos os módulos

jdk.*
javafx.*

Fonte: Oracle (2016)

java.base

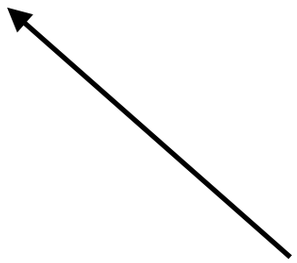
- Raiz de todos os módulos
- Todo módulo implicitamente requer java.base
- Define e exporta todos os pacotes fundamentais da plataforma Java

```
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;
    ...
}
```

Linking

```
jlink --modulepath $JDKMODS:$MYMODS
      --addmods teste.mod
      --output myimage
```

```
$ myimage/bin/java -listmods
java.base@9.0-ea
java.logging@9.0-ea
java.sql@9.0-ea
java.xml@9.0-ea
teste.mod
teste.mod.d1
```



Não é versionamento!

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!

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)

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**

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

- 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

requires public

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

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

Módulos sem nome

- Tentativa de carregar uma classe que não se encontra em nenhum módulo conhecido

- 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)

- Pode-se declarar dependências de módulos automáticos

Suporte SPI

- Service Provider Interface
- 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

- Frameworks que dependem de reflection (a maioria) têm problemas para acessar dados privativos em módulos
- Antes de Java 9 Reflection tudo era possível
- Solução
  - Pacotes e módulos abertos
  - Opções da JVM

Pacotes abertos

- Libera acesso total (permite reflection) 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 pacotes

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

Opções da app java

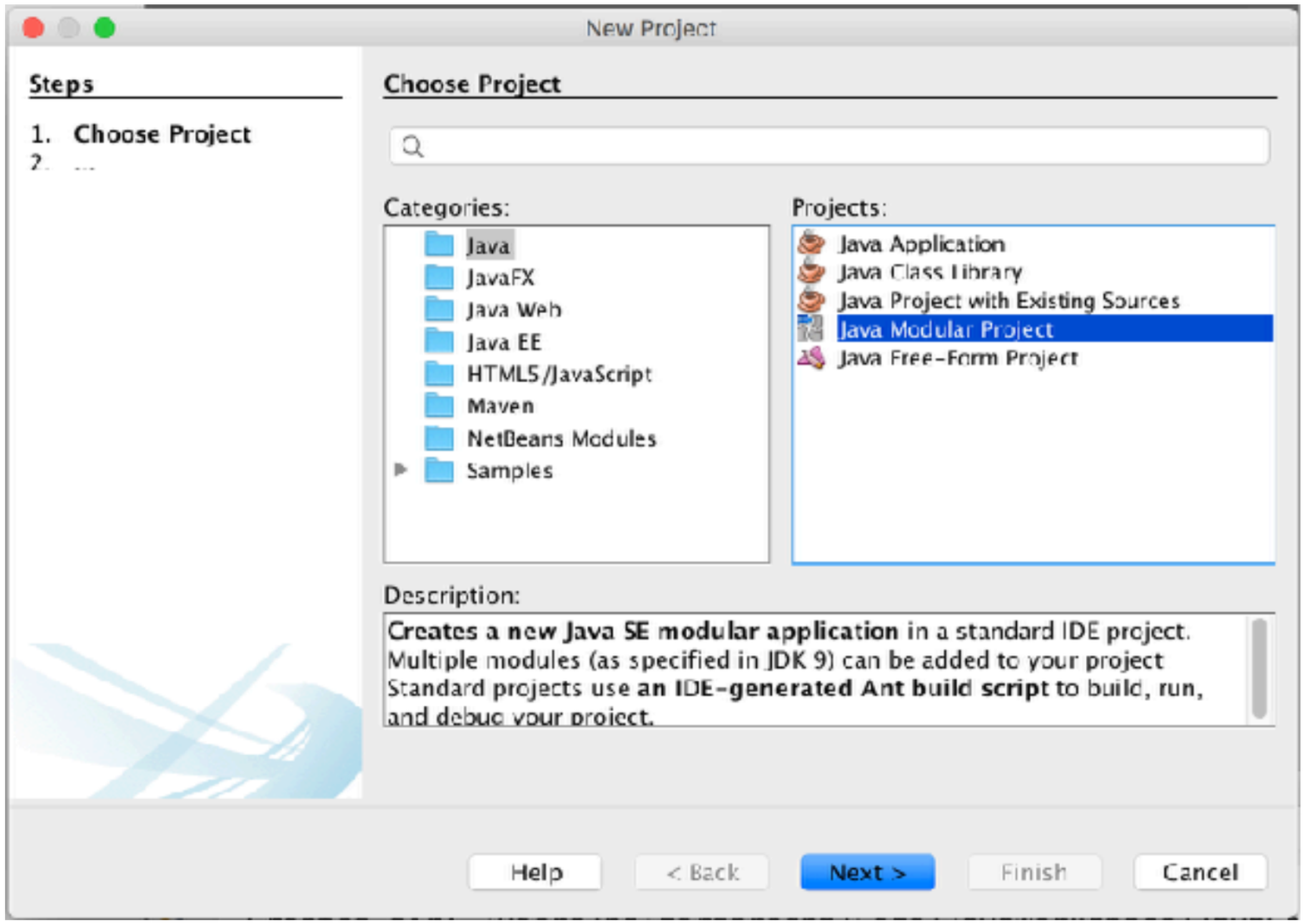
```
java

   --module-path caminho

   --add-modules

   --add-opens

   -- module

   --permit-illegal-access
```

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/

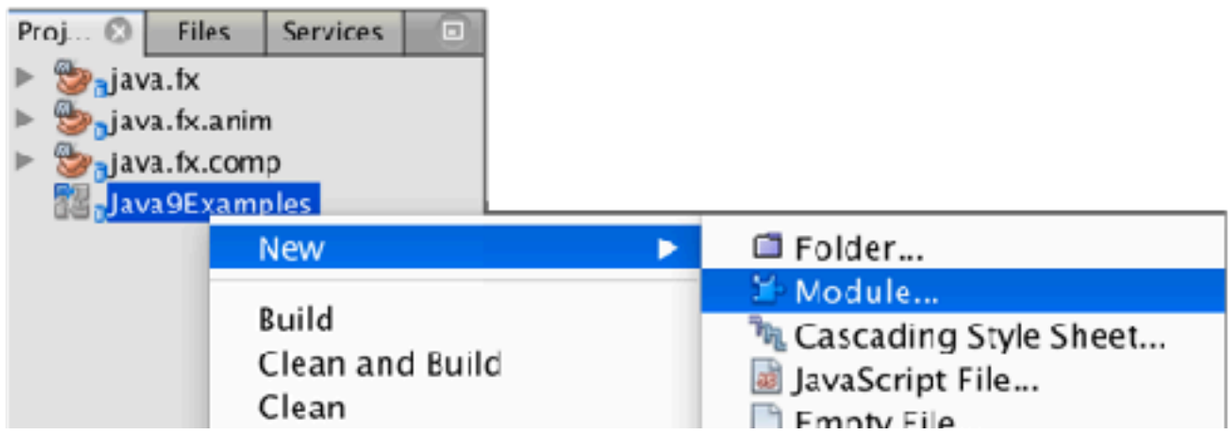
Projeto modular no NetBeans

- Depois crie um novo projeto modular

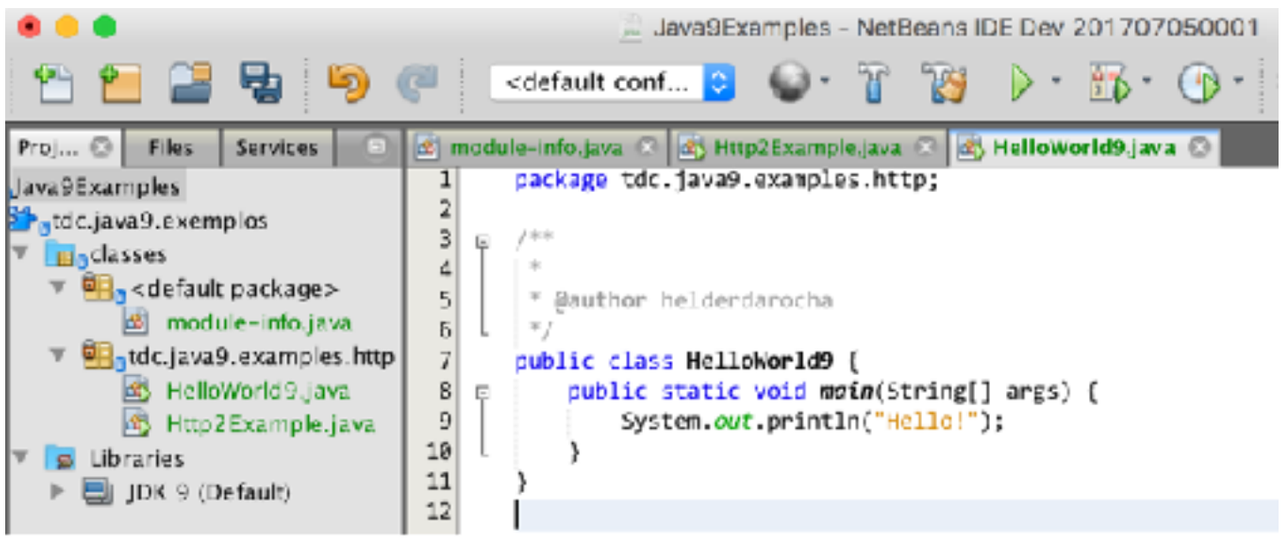


Projeto modular no NetBeans

- 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



Exemplo

- Para usar uma outra novidade do Java 9 (que não contamos entre as 9 :-)

- Suporte HTTP/2

```
module tdc.java9.exemplos {
    requires jdk.incubator.httpclient;
}
```

HTTP/2

- HTTP/2 suporta conexões persistentes, cabeçalhos mais eficientes, e outros melhoramentos

- Java 9 tem suporte nativo a HTTP/2

```
HttpClient client = HttpClient.newHttpClient();
HttpRequest request =
      HttpRequest.newBuilder(
        URI.create("https://www.thedevelopersconference.com.br"))
      .GET()
      .build();
HttpResponse.BodyHandler responseBodyHandler =
      HttpResponse.BodyHandler.asString();
HttpResponse response = client.send(request, responseBodyHandler);
String body = response.body().toString();
System.out.println(body);
```

2

Coleções

2 Coleções

- **Factory methods** para criar coleções **imutáveis** (não é possível alterar elementos, nem adicionar, nem remover)
- Não suportam valores nulos
- Se elementos forem serializáveis, coleções também serão

Java 8

```
List<String> lista1 = new ArrayList<>();
lista1.add("Um");
lista1.add("Dois");
lista1.add("Tres");
lista1 = Collections.unmodifiableList(lista1);
```

Java 8

```
List<String> lista1 = new ArrayList<>();
lista1.add("Um");
lista1.add("Dois");
lista1.add("Tres");
lista1 = Collections.unmodifiableList(lista1);
```

Java 9

```
List<String> lista2 = List.of("Um", "Dois", "Três");

Set<String> conj = Set.of("A", "B", "C");

Map<Integer, String> mapa1 = Map.of(1, "Um", 2, "Dois", 3, "Três");

Map.Entry<String, String> e1 = Map.entry("A", "Um");
        Map.Entry<String, String> e2 = Map.entry("B", "Dois");
        Map<String, String> mapa2 = Map.ofEntries(e1, e2);
```

3

Concorrência

3 Concorrência

- Novidades no pacote **java.concurrency** (Java 5)
- Reactive Streams
- CompletableFuture

Reactive Streams

- Implementação do padrão (reactivestreams.org)
- Um stream reativo interage com a **fonte** para otimizar a transferência e evitar sobrecarregar o destino
- API oferece interface **Flow** com arquitetura publish-subscribe (componentes **Publisher**, **Subscriber**, **Subscription**)

Exemplo (subscriber)

```
import java.util.concurrent.Flow;
import java.util.concurrent.Flow.Subscription;

public class Assinante<T> implements Flow.Subscriber<T> {
    private Subscription assinatura;

    @Override
    public void onSubscribe(Flow.Subscription subscription) {
        ...}

    @Override
    public void onNext(T item) {
        ...}

    @Override
    public void onError(Throwable throwable) {
        ...}

    @Override
    public void onComplete() {
        ...}
}
```

Exemplo (teste)

```
SubmissionPublisher<String> publisher =
     new SubmissionPublisher<>();

Assinante<String> assinante = new Assinante<>();

publisher.subscribe(assinante);
```

```
System.out.println("Publicando dados...");
String[] dados = { "Um", "dois", "três", "quatro"};

Arrays.asList(dados).stream().forEach(i -> publisher.submit(i));

publisher.close();
```

4

Interfaces

4 Interfaces

- Interfaces em Java 8 ganharam métodos **default**
- Interfaces em Java 9 ganharam métodos **private**

```
public interface Java9Interface {
    default String tagText(String nome, String tag) {
        return initag(tag) + nome + endtag(tag);
    }

    private String initag(String tag) {
        return "<"+tag+">";
    }

    private String endtag(String tag) {
        return "</"+tag+">";
    }
}
```

5

Process API

Process API

- JEP 102: Process API Updates

- Mais controle sobre **processos do sistema operacional,** sem precisar recorrer a métodos nativos; notificações assíncrona sobre eventos do processo

- Adiciona recursos em **java.lang.Process** e cria a interface **java.lang.ProcessHandle.Info**

- Facilita a obtenção do **PID** (ID do processo nativo)

Métodos novos de Process

- `Stream<ProcessHandle> children()`
- `Stream<ProcessHandle> descendants()`
- `long getPid()`
- `ProcessHandle.Info info()`
- `CompletableFuture<Process> onExit()`
- `boolean supportsNormalTermination()`
- `ProcessHandle toHandle()`

ProcessHandle

- `static Stream<ProcessHandle> allProcesses()`
- `Stream<ProcessHandle> children()`
- `int compareTo(ProcessHandle other)`
- `static ProcessHandle current()`
- `Stream<ProcessHandle> descendants()`
- `boolean destroy()`
- `boolean destroyForcibly()`
- `long getPid()`
- `ProcessHandle.Info info()`
- `boolean isAlive()`
- `static Optional<ProcessHandle> of(long pid)`
- `CompletableFuture<ProcessHandle> onExit()`
- `Optional<ProcessHandle> parent()`
- `boolean supportsNormalTermination()`

ProcessHandle.Info

- `Optional<String[]> arguments()`
- `Optional<String> command()`
- `Optional<String> commandLine()`
- `Optional<Instant> startInstant()`
- `Optional<Duration> totalCpuDuration()`
- `Optional<String> user()`

* **Optional** (Java 8) retorna uma referência ou null - evita null pointer

Exemplo

- Lendo processo de uma aplicação (abre a aplicação e obtém o PID) e o processo da aplicação Java atual

```
Process p =
      new ProcessBuilder
       ("/Applications/Firefox.app/Contents/MacOS/firefox")
       .start();

System.out.println("Firefox pid: " + p.getPid());

long pid = ProcessHandle.current().getPid();
System.out.println("Java pid: " + pid);
```

6

Exceções

Try com resources

- Melhora o uso do try-com-resources, criado no Java 7
- Variáveis que forem **efetivamente finais** (ou declaradas finais) não precisam ser declaradas dentro do try.

```
public static void read(InputStream is) throws IOException {
        BufferedReader reader = null;
        final OutputStream os = null;
        //try (InputStream a = is) {
        try (is; os) {
            reader = new BufferedReader(new InputStreamReader(is));
            String line = reader.readLine();
            while(line != null) {
                System.out.println(line);
                line = reader.readLine();
            }
        }
    }
```

7

Performance

Imagens de múltiplas resoluções

- JEP-251: **java.awt.image.MultiResolutionImage**

- Objeto que encapsula URLs de variantes de uma imagem em resoluções diferentes

G1GC é default

- Coletor "Garbage-First" G1 (existe desde Java 7/8) é o coletor padrão (no lugar de ParallelGC)

- Multithreaded; pausas mais curtas e previsíveis (geralmente); particiona o heap em **várias áreas pequenas** (em vez de três áreas - eden, survivor, e old)

- Eden, survivor e old são **papéis** exercidas pelas áreas

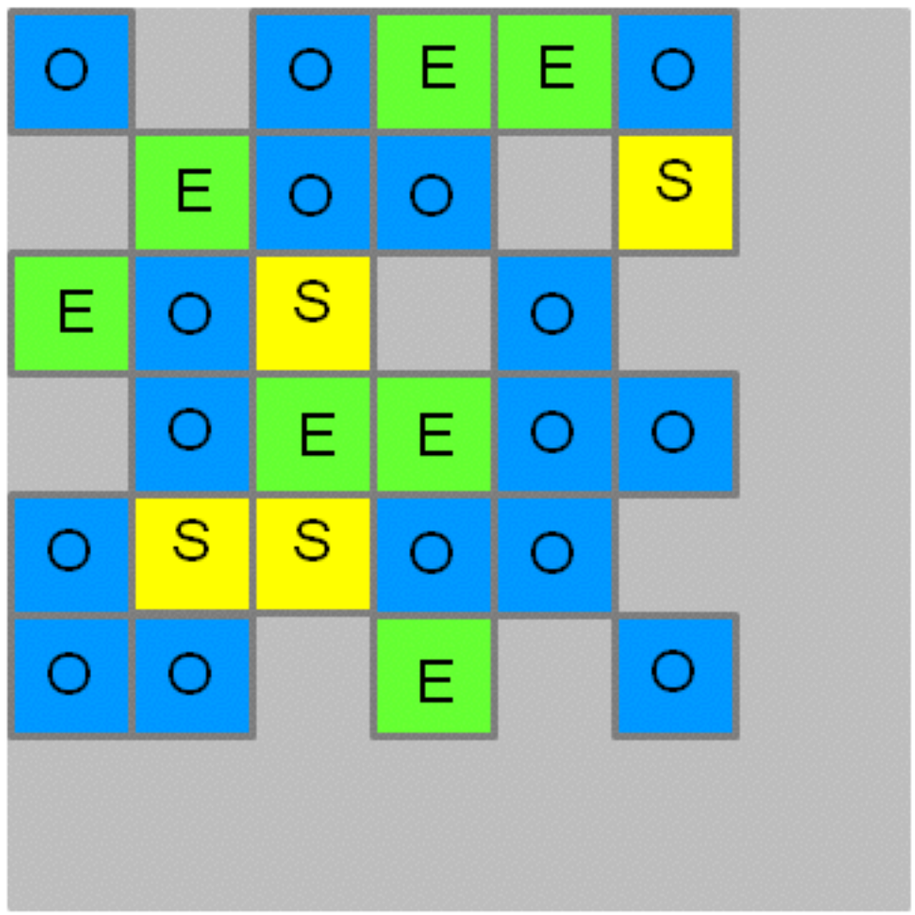
- Prioriza a coleta nas **áreas com mais lixo**

- **Compacta** o heap, reduzindo a fragmentação

Estrutura do G1GC

–XX:+UseG1GC

–XX:-UseG1GC



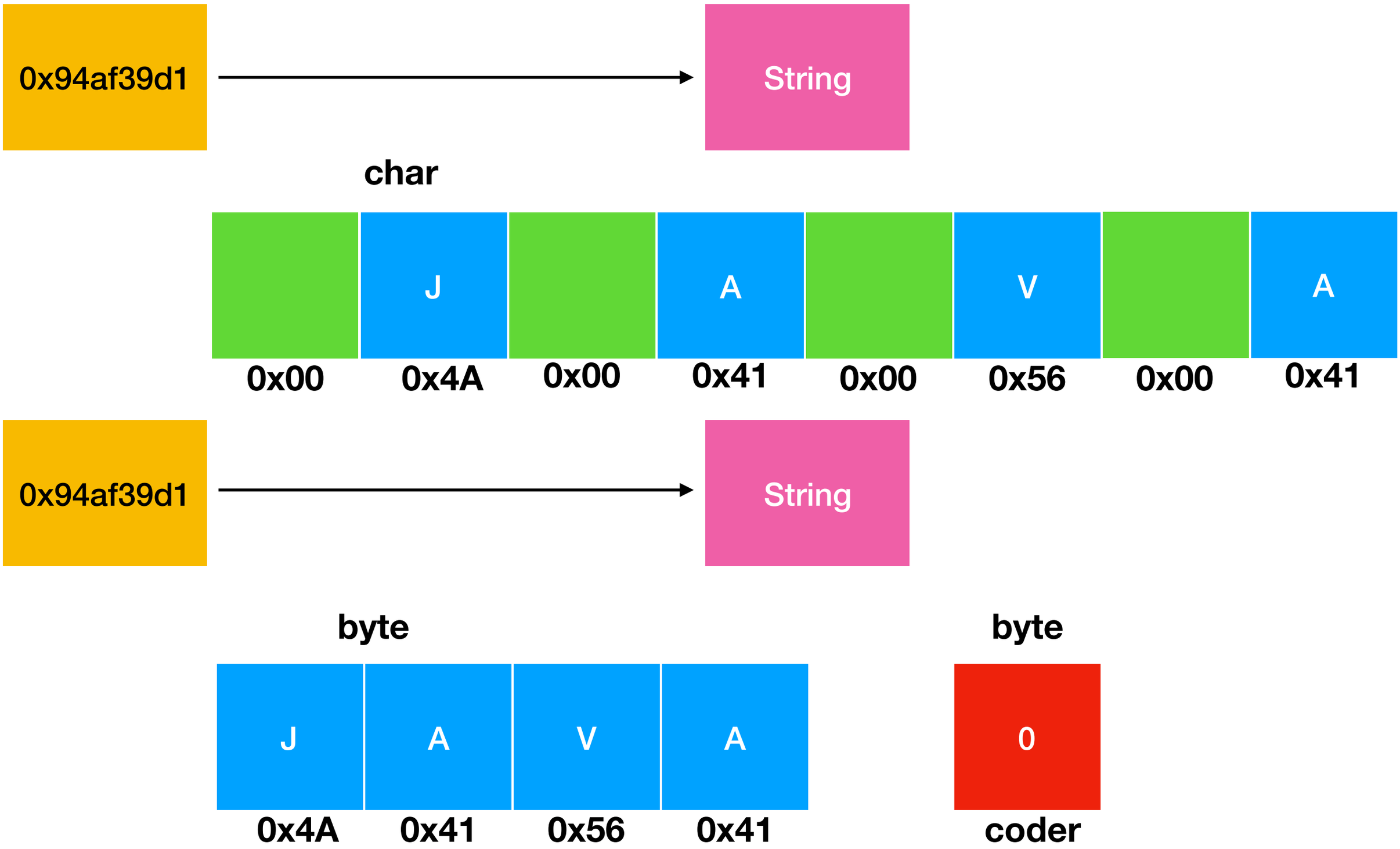
http://www.oracle.com/technetwork/tutorials/tutorials-1876574.html

Strings compactos

- **Reduz o tamanho** da representação interna de strings, usando menos bytes (eficiente para textos que não precisam de 16 bits)
  - Java 6 tinha função similar que precisava ser habilitada na JVM, mas não era habilitada por default
    - Habilitado por default em Java 9
- Quase **dobra a performance** da criação de strings e gera quase **50% menos lixo**, para aplicações típicas
- Se a maior parte dos strings usados precisarem de 16 bits, pode não ser eficiente (**+XX:-CompactStrings** para desabilitar)

```
+XX:-CompactStrings
```

Compact strings



8

Java Shell

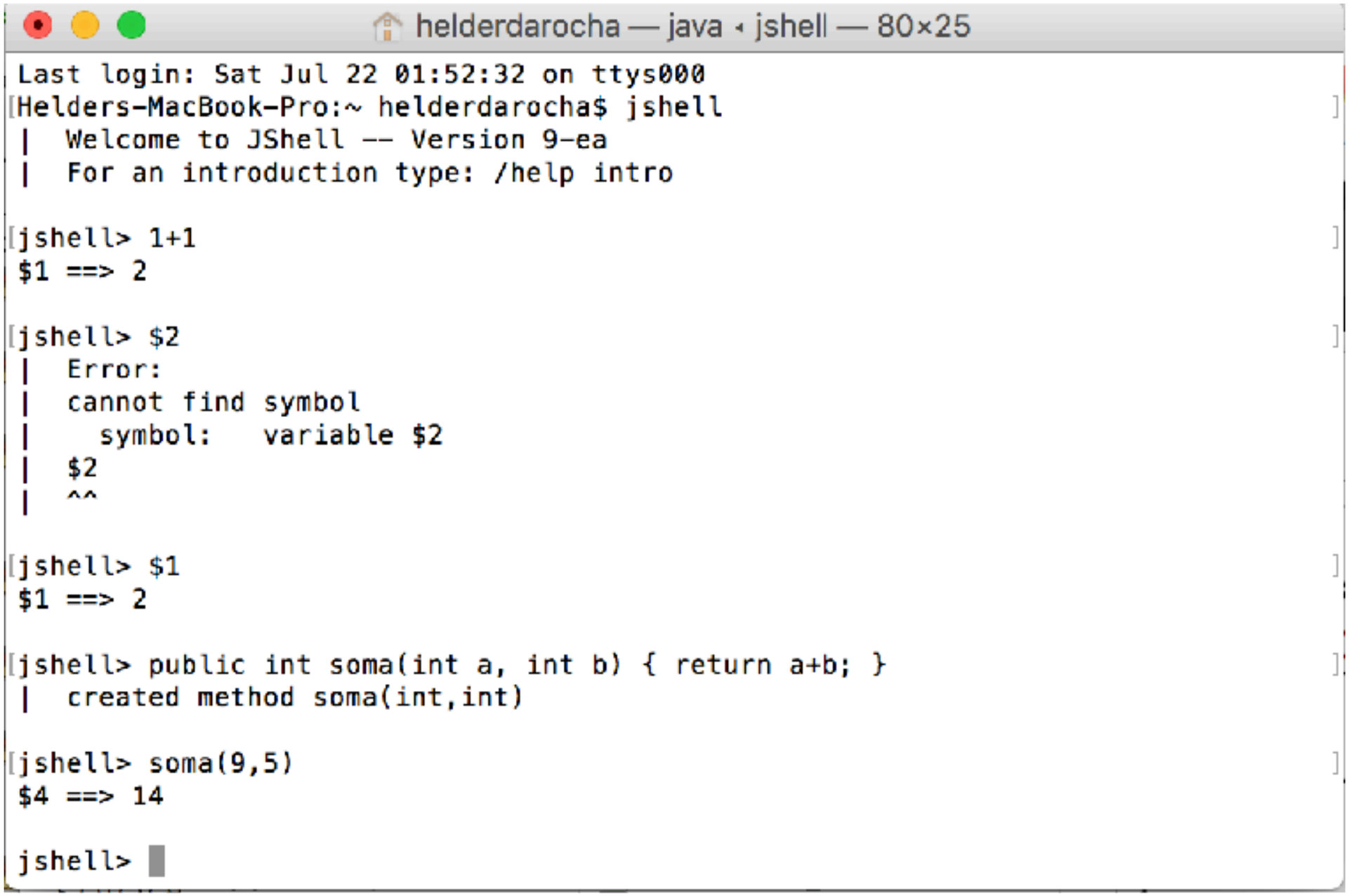
6 JShell

- REPL (Read-Eval-Print-Loop)

- Permite programar em Java de forma interativa

- Comando: **jshell**

Exemplo


helderdarocha — java ◂ jshell — 80×25
Last login: Sat Jul 22 01:52:32 on ttys000
Helders-MacBook-Pro:~ helderdarocha$ jshell
| Welcome to JShell -- Version 9-ea
| For an introduction type: /help intro
jshell> 1+1
$1 ==> 2
jshell> $2
| Error:
| cannot find symbol
| symbol: variable $2
| $2
| ^^
jshell> $1
$1 ==> 2
jshell> public int soma(int a, int b) { return a+b; }
| created method soma(int,int)
jshell> soma(9,5)
$4 ==> 14
jshell>

/help

helderdarocha — java ◂ jshell — 89×44

```
jshell> /help
|  Type a Java language expression, statement, or declaration.
|  Or type one of the following commands:
|  /list [<name or id>|-all|-start]
|       list the source you have typed
|  /edit <name or id>
|       edit a source entry referenced by name or id
|  /drop <name or id>
|       delete a source entry referenced by name or id
|  /save [-all|-history|-start] <file>
|       Save snippet source to a file.
|  /open <file>
|       open a file as source input
|  /vars [<name or id>|-all|-start]
|       list the declared variables and their values
|  /methods [<name or id>|-all|-start]
|       list the declared methods and their signatures
|  /types [<name or id>|-all|-start]
|       list the declared types
|  /imports
|       list the imported items
|  /exit
|       exit jshell
|  /env [-class-path <path>] [-module-path <path>] [-add-modules <modules>] ...
|       view or change the evaluation context
|  /reset [-class-path <path>] [-module-path <path>] [-add-modules <modules>]...
|       reset jshell
|  /reload [-restore] [-quiet] [-class-path <path>] [-module-path <path>]...
|       reset and replay relevant history -- current or previous (-restore)
|  /history
|       history of what you have typed
|  /help [<command>|<subject>]
|       get information about jshell
|  /set editor|start|feedback|mode|prompt|truncation|format ...
|       set jshell configuration information
|  /? [<command>|<subject>]
|       get information about jshell
|  /!
|       re-run last snippet
|  /<id>
|       re-run snippet by id
|  /-<n>
|       re-run n-th previous snippet
|
```

9

Outras novidades

Multi-release JARs

- Em um único JAR pode-se conter **versões diferentes** de classes e resources (JEP-238)

- Ideal para frameworks que suportam muitas versões (ex: Spring)

- Deve incentivar frameworks a usarem recursos mais novos, e consequentemente usuários a terem interesse em migrar para novas versões.

Multi-release JARs

- Manifest.mf contém **Multi-Release: true**

- + Subdiretório **/META-INF/versions** que contém um diretório com número da versão

```
Um.class
Dois.class
Tres.class
META-INF
    MANIFEST.MF
    versions/
        9/
            Um.class
        10/
            Um.class
            Dois.class
```

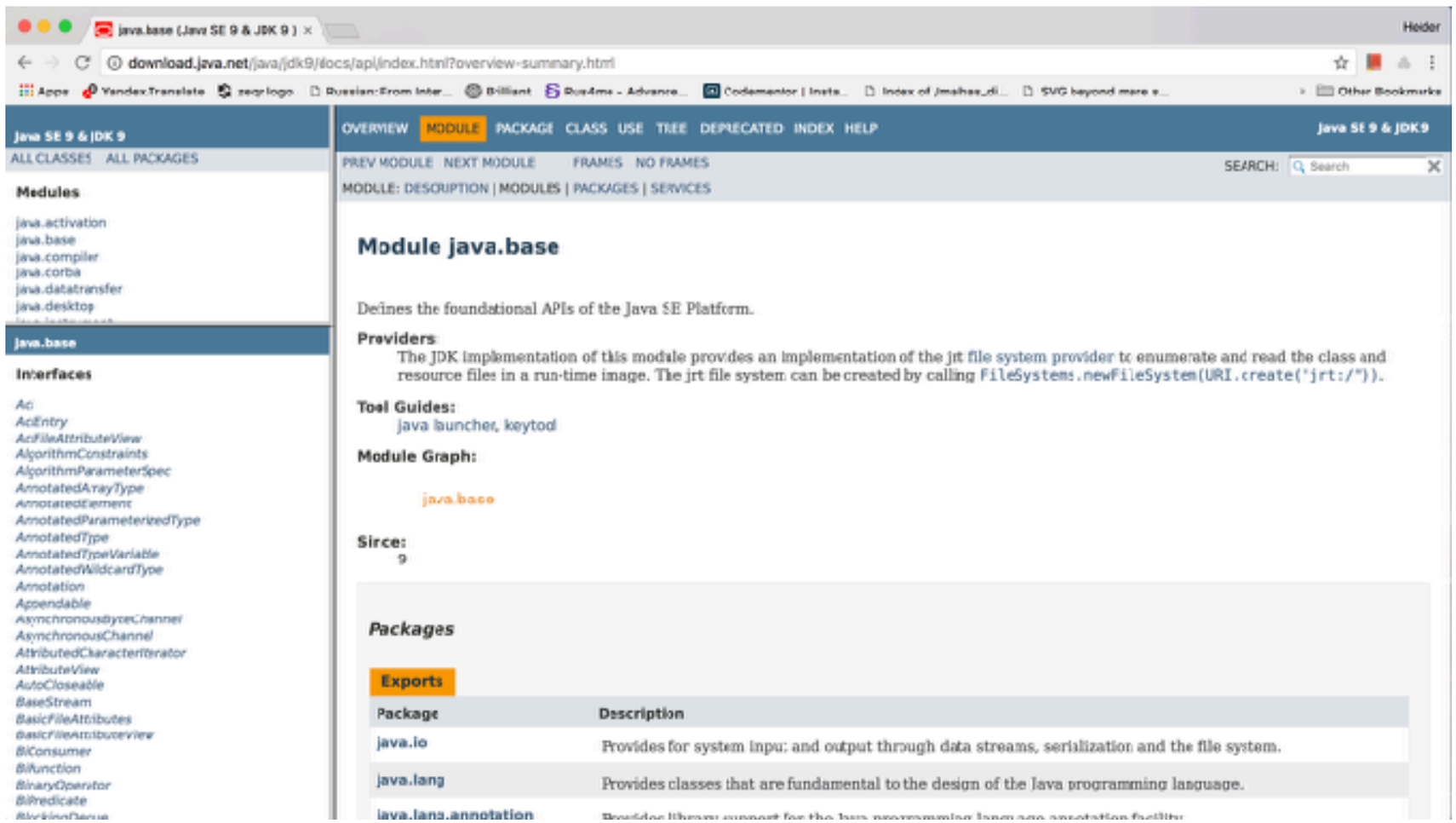
Multi-release: true

- Menor valor aceito é **9** (Java 8 e anterior **ignoram** pasta versions)

Documentação

- Nova API de Doclet permite nova documentação, para acomodar módulos, e inclui recursos como busca

- http://download.java.net/java/jdk9/docs/api/overview-summary.html



Quando adotar Java 9?

- Maior empecilho é a arquitetura modular.
- Vale a pena modularizar, mas migrar não será tão fácil quanto java 8
- Dificuldades com reflection. Vários frameworks não estão preparados para usar módulos
- Mecanismos que flexibilizam o encapsulamento (como --permit-illegal-access) permitem migração mais suave

nove novidades do java nove

Baixe esta palestra em

http://www.argonavis.com.br/download/tdc_2017_java9.html

+Links para outros recursos, código, referências

helder da rocha

helder@summa.com.br