ThoughtWorks®

Java e modularidade, um passeio cronológico

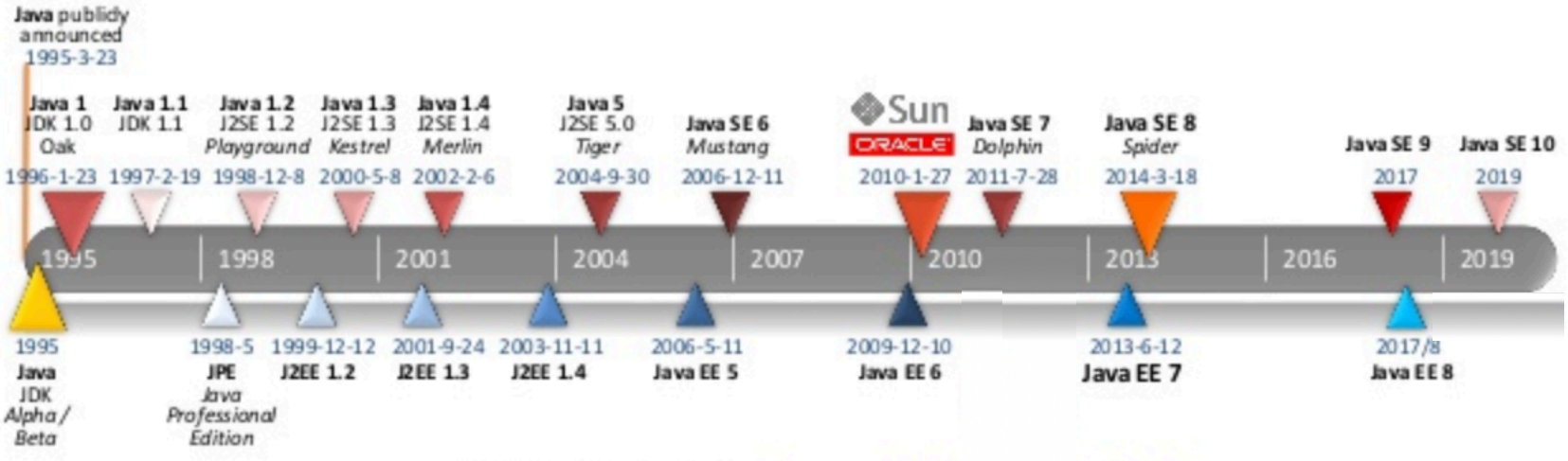
Nancy Lyra

Nancy Lyra

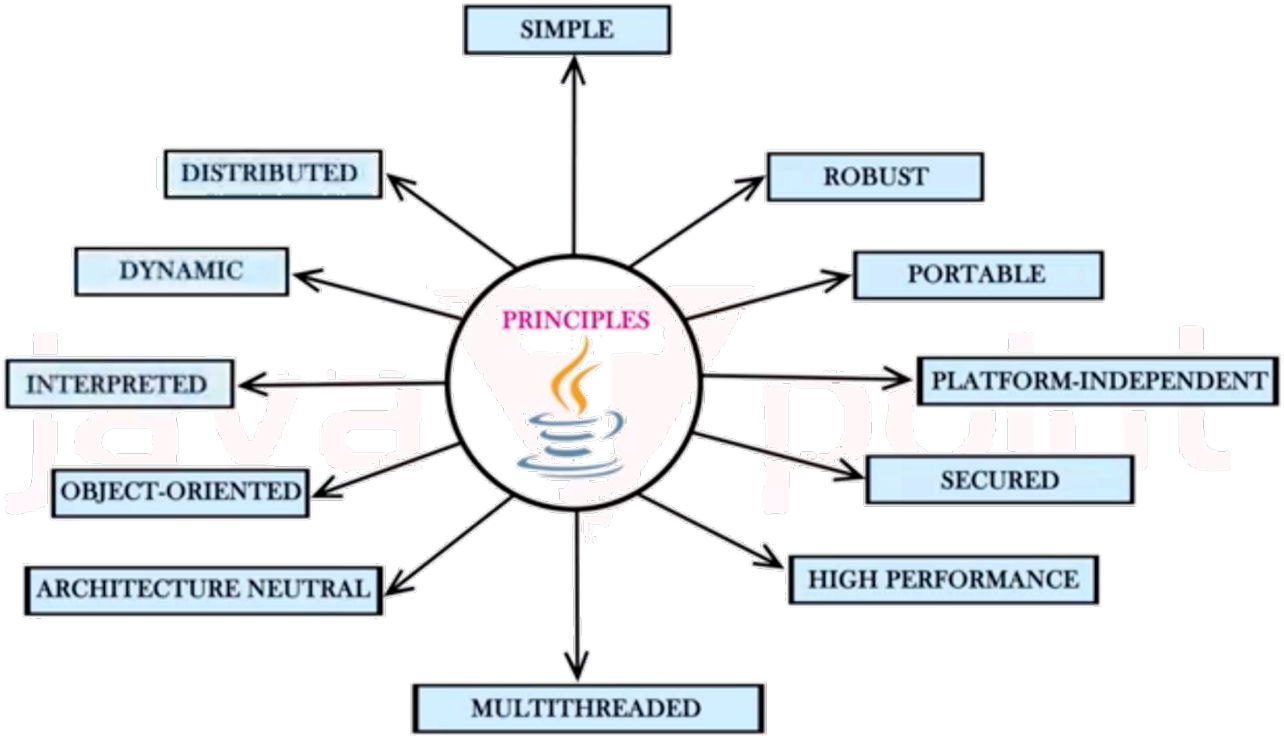
@nanalyra
nlino@thoughtworks.com

Pra chegar até aqui…

…Caminhada de erros e acertos



Príncipios


SIMPLE
DISTRIBUTED
ROBUST
DYNAMIC
PORTABLE
PRINCIPLES
INTERPRETED
PLATFORM-INDEPENDENT
OBJECT-ORIENTED
SECURED
ARCHITECTURE NEUTRAL
HIGH PERFORMANCE
MULTITHREADED

Deu errado

- *Foi originalmente desenhada para Televisão Digital*
- *Mas era uma tecnologia avançada paras as tvs digitais da época (jun/1991)*
- *Green Team (James Gosling, Patrick Naughton, Mike Sheridan)*

Mas deu certo

- *Era mais adequada para programação de internet*
- *Foi incorporada ao Netscape*

Deu errado

- *“GreenTalk”, .gt*
- *Oak, como parte do Green Project*
- *Árvore que simbolizava força*

Mas deu certo

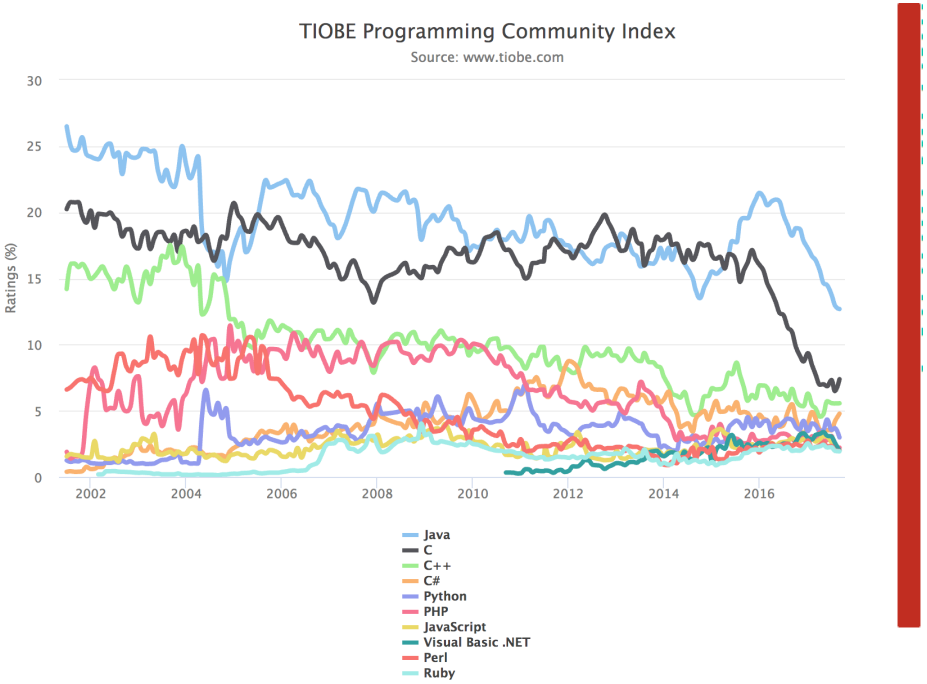
- *1995, por direitos autorais, mudança do nome pra Java*
- *Possível motivação: Café produzido na Ilha de Java (Indonésia)*
- *Só em jan/1996, saiu o JDK 1.0*

Outros Releases

JDK Beta	1995
JDK 1.0	January 1996
JDK 1.1	February 1997
J2SE 1.2	December 1998
J2SE 1.3	May 2000
J2SE 1.4	February 2002
J2SE 5.0	September 2004
Java SE 6	December 2006
Java SE 7	July 2011
Java SE 8 (LTS)	March 2014
Java SE 9	September 2017
Java SE 10	March 2018
Java SE 11 (LTS)	September 2018
Java SE 12	March 2019

Mas o que é JAVA?

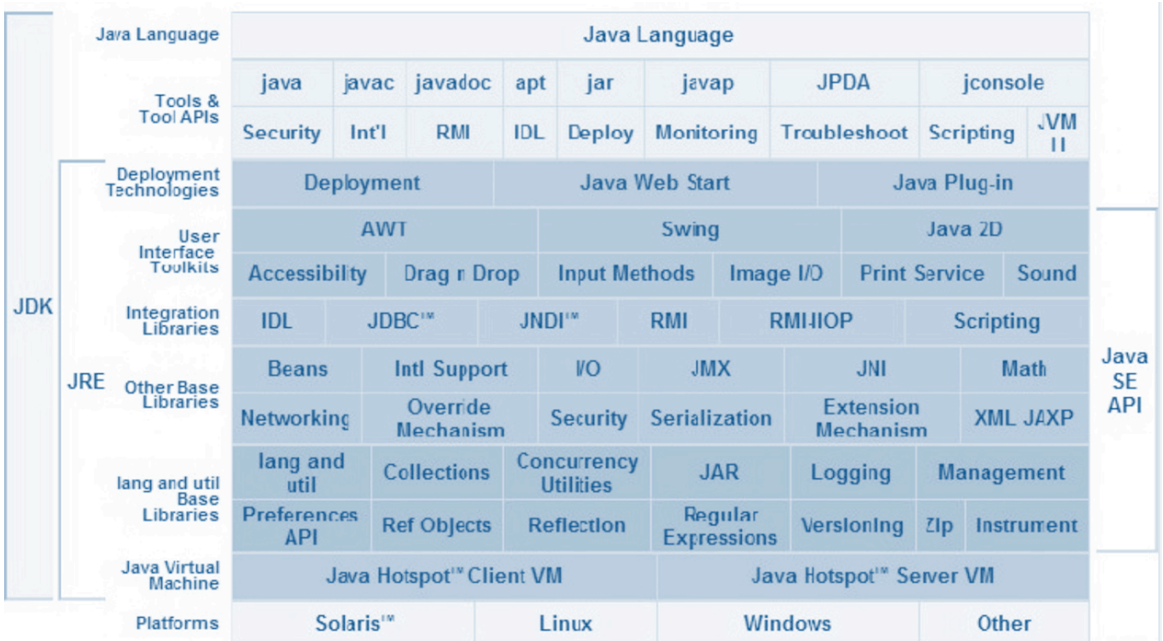
- Linguagem de Programação + Plataforma de Desenvolvimento



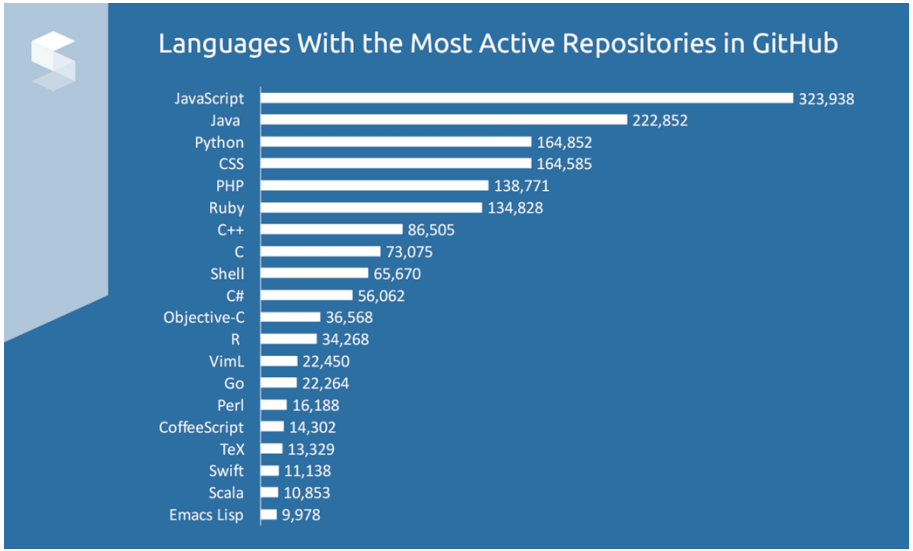
Programming Language	2017	2012	2007	2002	1997	1992	1987
Java	1	2	1	1	15	–	–
C	2	1	2	2	1	1	1
C++	3	3	3	3	2	2	5
C#	4	4	7	13	–	–	–
Python	5	7	6	11	27	–	–
Visual Basic .NET	6	17	–	–	–	–	–
PHP	7	6	4	5	–	–	–
JavaScript	8	9	8	7	23	–	–
Perl	9	8	5	4	4	10	–
Assembly language	10	–	–	–	–	–	–
COBOL	25	28	17	9	3	9	9
Lisp	31	12	15	12	9	4	2
Prolog	32	31	26	16	20	11	3
Pascal	114	15	21	97	8	3	4

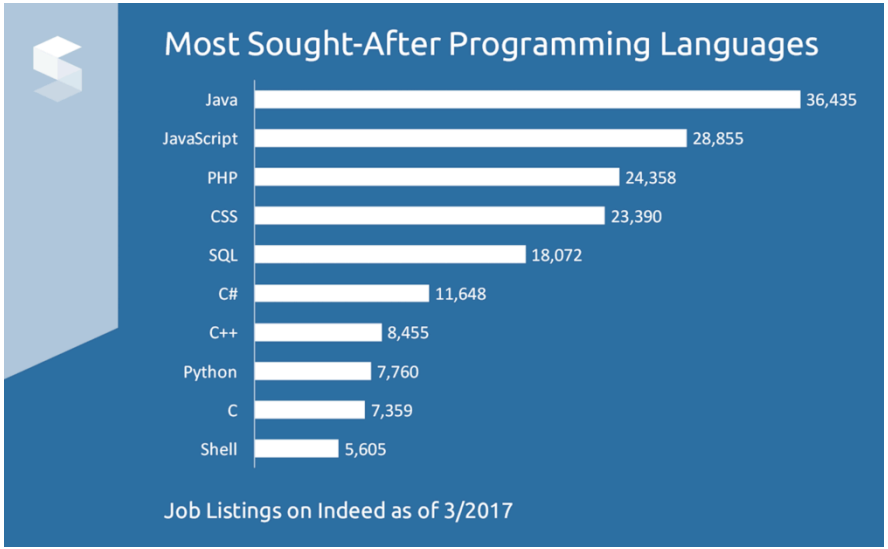
Mas o que é JAVA?

- Linguagem de Programação + Plataforma de Desenvolvimento

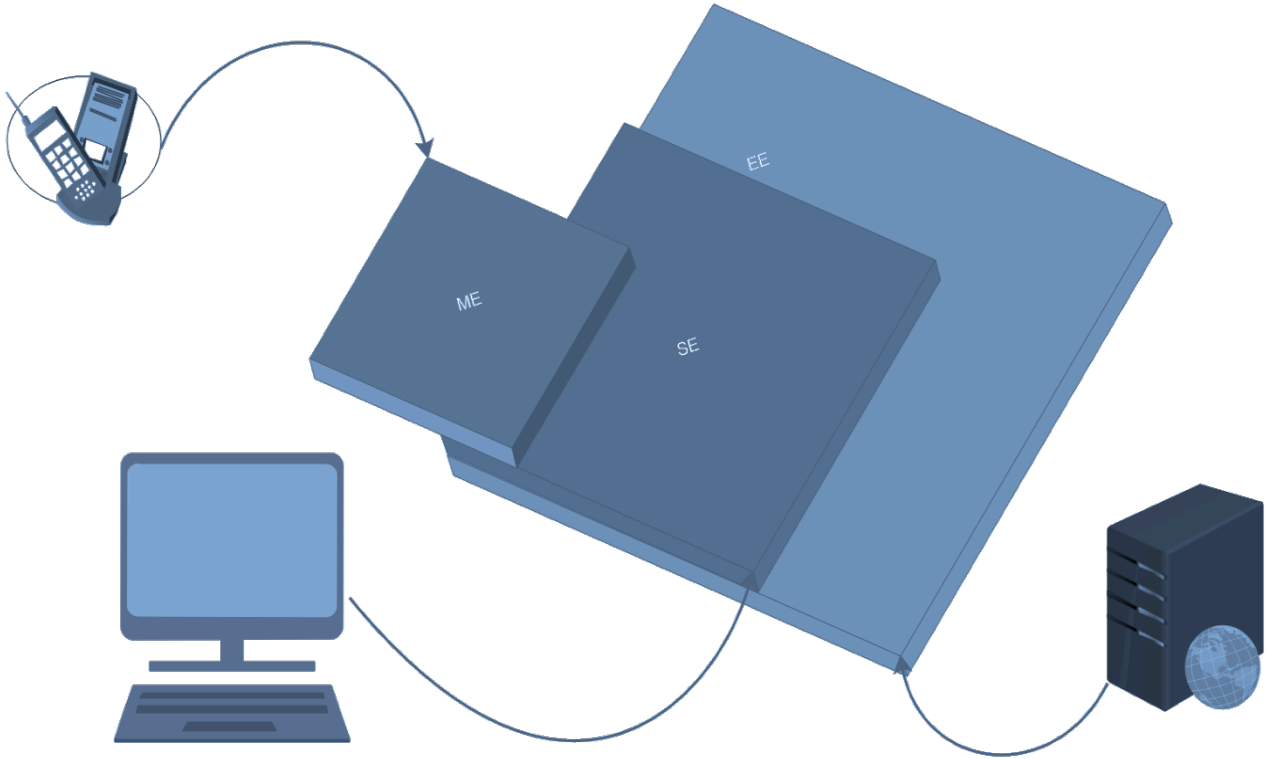


Mas o povo usa?

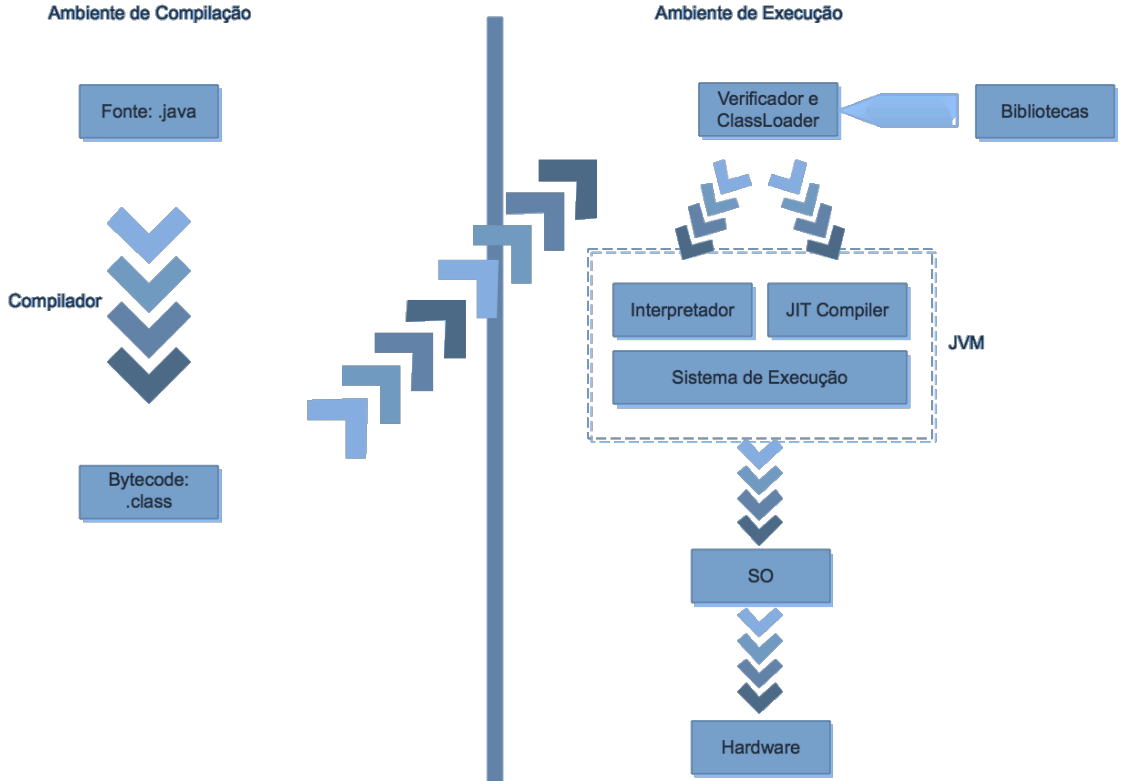




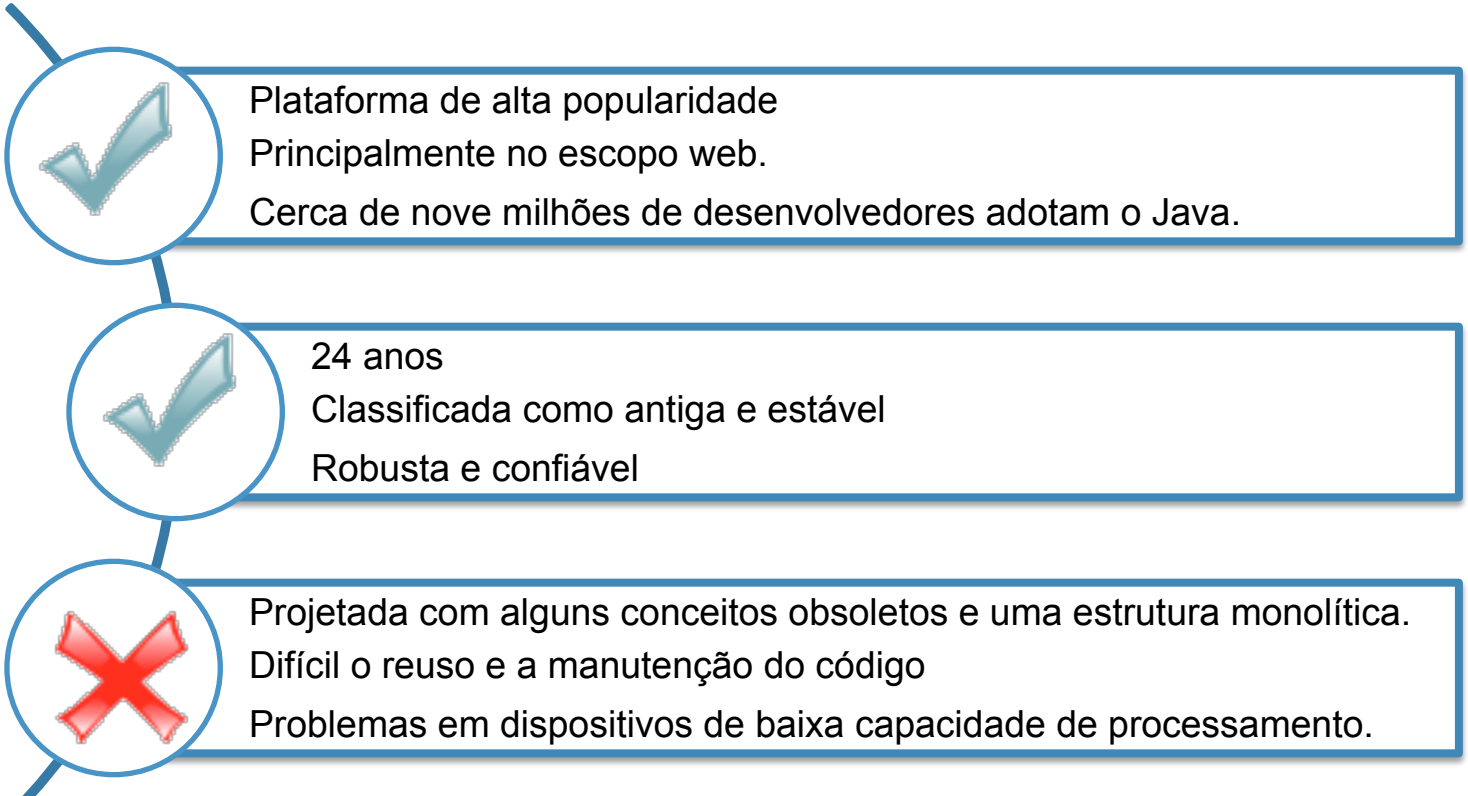
Opera, principalmente, em 3 áreas



Como isso é possível?

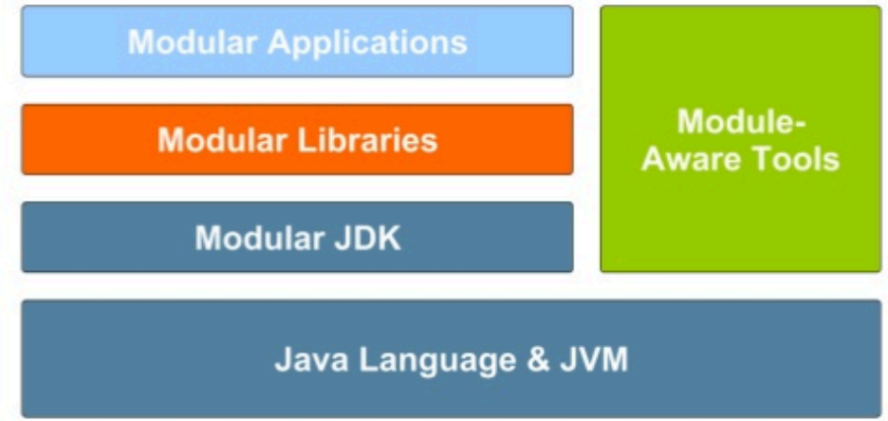


Sim, ok. Mas…


Plataforma de alta popularidade
Principalmente no escopo web.
Cerca de nove milhões de desenvolvedores adotam o Java.
24 anos
Classificada como antiga e estável
Robusta e confiável
Projetada com alguns conceitos obsoletos e uma estrutura monolítica.
Difícil o reuso e a manutenção do código
Problemas em dispositivos de baixa capacidade de processamento.

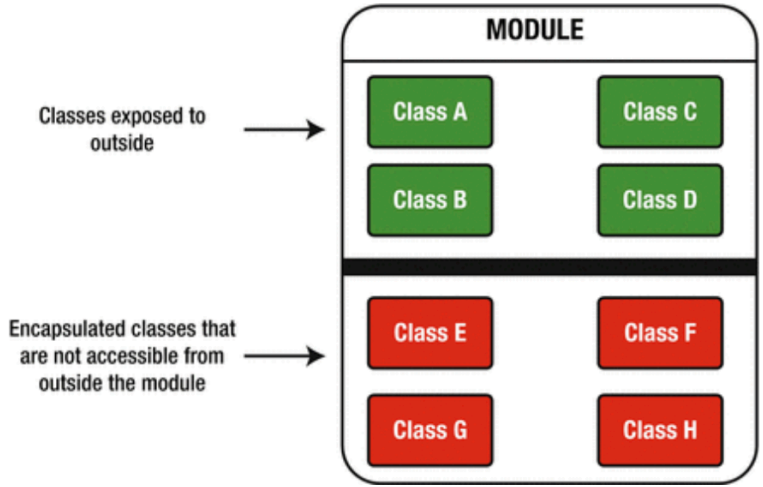
Consequências imediatas (Sistema Monolítico)

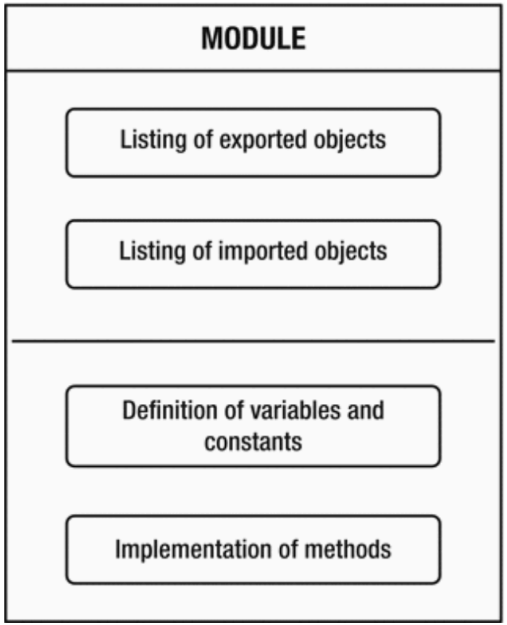
- Aumento da complexidade a cada nova funcionalidade
- Erros inesperados pode acontecer após mínimas alterações
- Baixo reuso
- Forte acoplamento
- Dependência de boa documentação, processo e ferramentas



- MANUTENÇÃO????
  - Tempo, esforço e custo
- Possível Solução: MODULARIDADE

Aplicação Modular

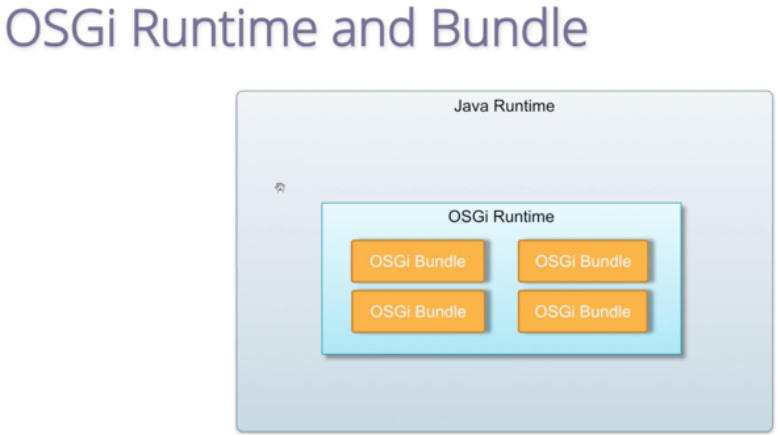




Basicamente o que queremos é:

- Forte encapsulamento
- Interfaces explícitas
- Alta coesão
- Baixo acoplamento

- Já temos isso tudo em OO?
- E OSGi?

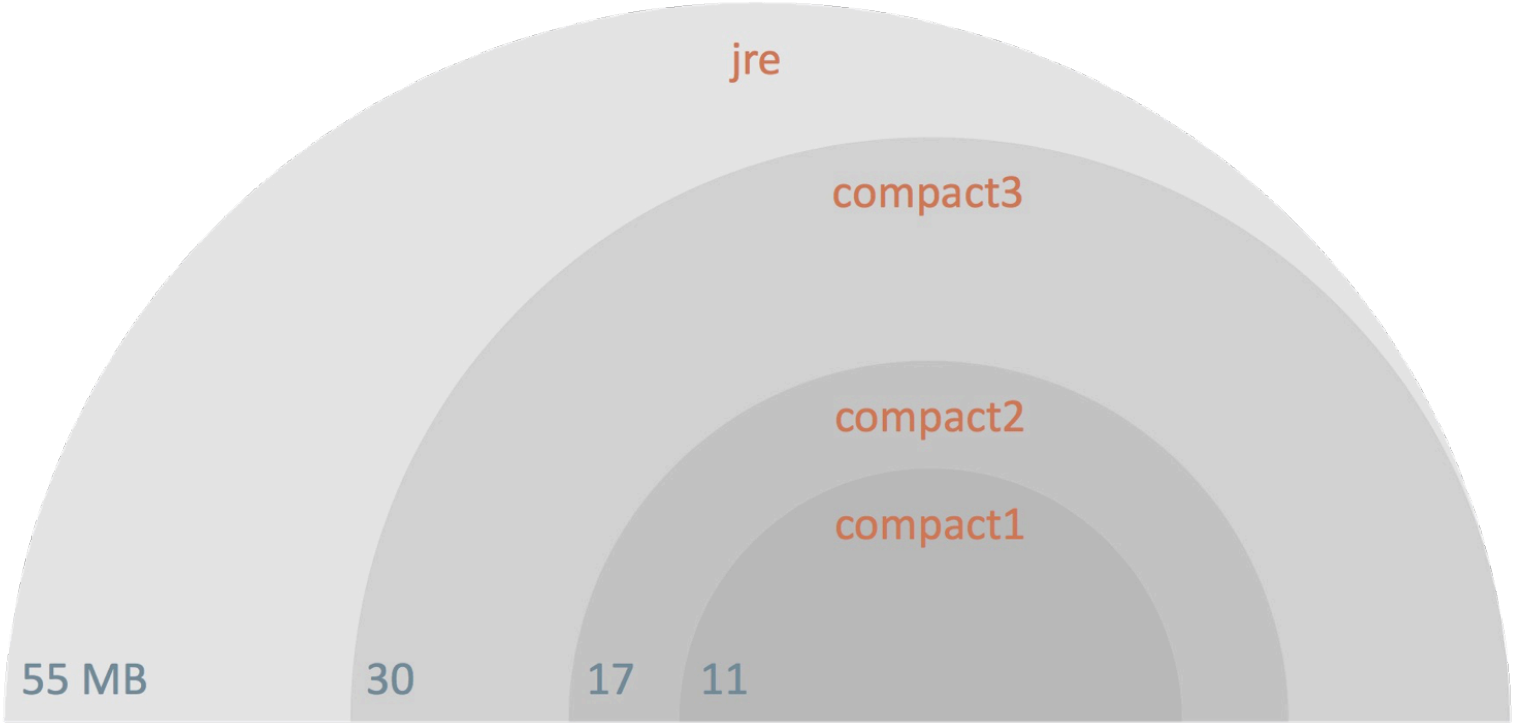


JDK Modular

- JDK 8 já temos Compact Profiles

Full SE API	Beans	Input Methods	IDL
	Preferences	Accessibility	Print Service
	RMI-IIOP	CORBA	Java 2D
	Sound	Swing	
	AWT	Drag and Drop	
	Image I/O	JAX-WS	
compact3	Security[1]	JMX	JNDI
	XML JAXP[2]	Management	Instrumentation
compact2	JDBC	RMI	XML JAXP
compact1	Core (`java.lang.*`)	Security	Serialization
	Networking	Ref Objects	Regular Expressions
	Date and Time	Input/Output	Collections
	Logging	Concurrency	Reflection
	JAR	ZIP	Versioning
	Internationalization	JNI	Override Mechanism
	Extension Mechanism	Scripting	

Tamanho do arquivo



Um pequeno detalhe:

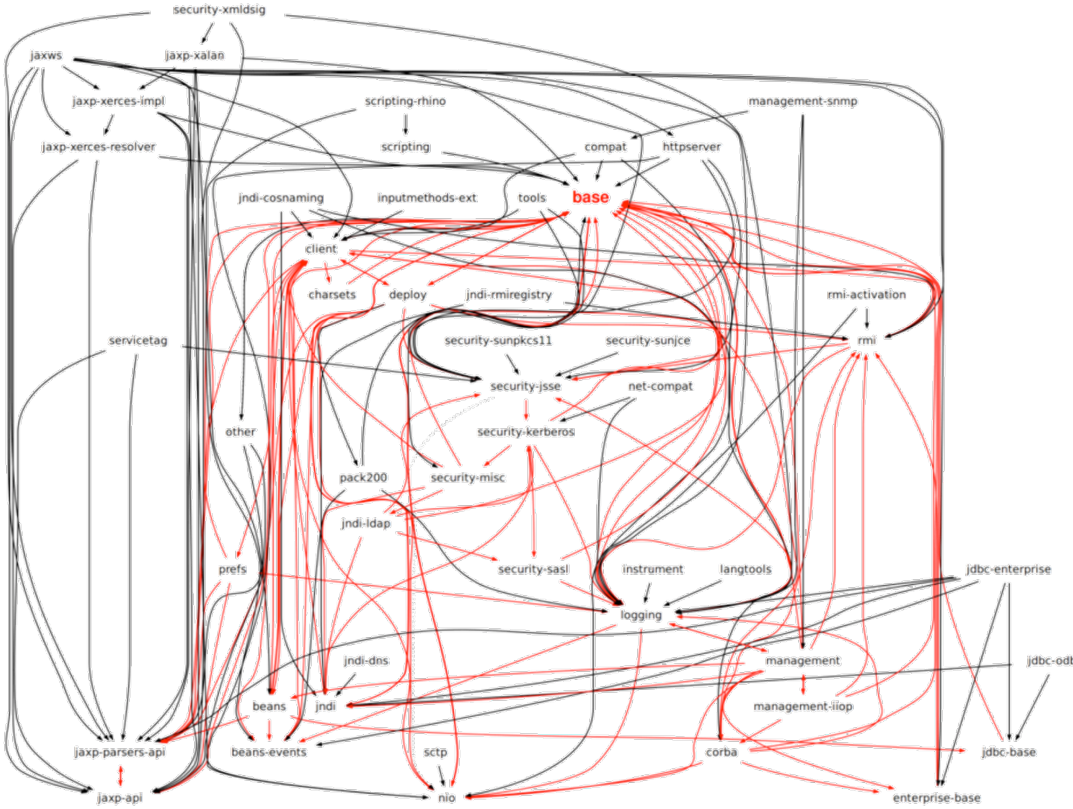
- A plataforma evoluiu:
  - Sistema (dispositivos embarcados) para uma rica coleção de bibliotecas

- Trouxe problemas
  - **Tamanho:** O JDK sempre foi disponibilizado como um grande e indivisível artefato de software. Ao longo dos anos só aumentou
  - **Complexidade:** O JDK é profundamente interconectado, ou seja, as bibliotecas são muito dependentes umas das outras, compondo assim uma estrutura monolítica.

O tamanho do desafio:

- A estrutura apresenta também conexões inesperadas entre APIs e suas respectivas implementações
- Consequência imediata:
  - Alto tempo de inicialização
  - Alto consumo de memória
  - Degradação do desempenho das aplicações

- Para se ter uma ideia, o ”Alô mundo”
  - Carrega e inicializa mais de 300 classes
  - Gasta, em média, 100ms em uma máquina desktop.



Alguns números

- JDK 1.0 - (1996)
  - standard packages: java.lang, java.io, java.applet, java.awt, java.net, and java.util.
  - 8 packages e 212 classes/interfaces.
- JDK 1.1
  - 504 classes/interfaces.
- JDK 1.2 – (1998)
  - 1.520 classes/interfaces.
- JDK 1.4
  - 2,991 classes/interfaces.
- JSE 8.0
  - 4,240 classes/interfaces.

Aí está o GRANDE trabalho:

Aprimorar o tempo de inicialização e o consumo de memória.

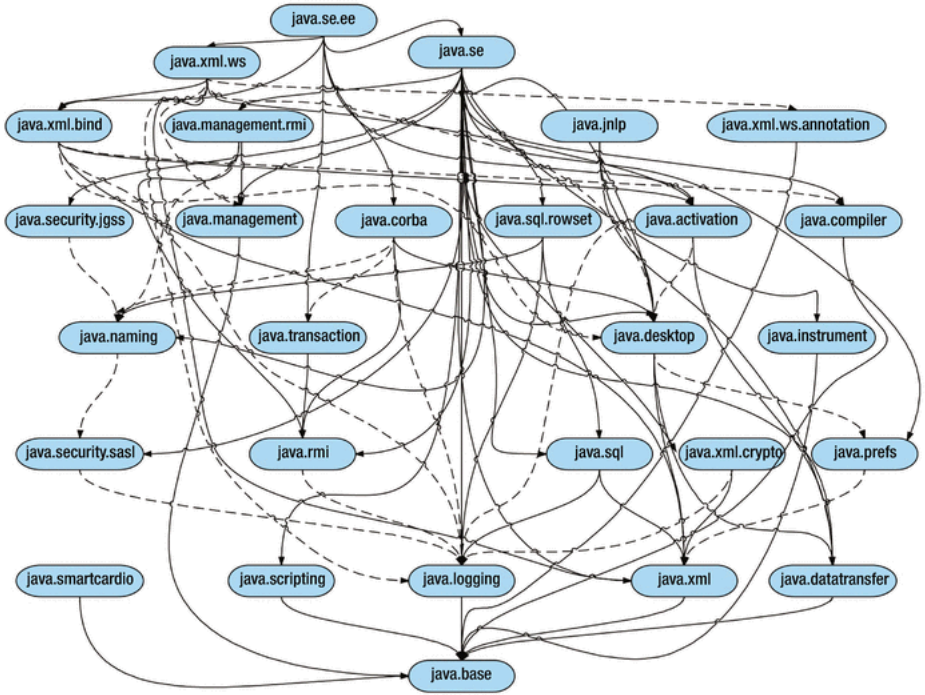
Com esse objetivo, a Oracle optou por dividir o JDK

- Conjunto de módulos separados e independentes.
- Identificar interconexões entre bibliotecas
- Eliminar dependências quando possível.

Consequências do JDK modular

- Apenas os módulos necessários para inicializar a aplicação serão carregados.
- Utilização de módulos poderá ser empregada não apenas pelo JDK, mas também por bibliotecas e aplicações.

Como ficou o JDK 9

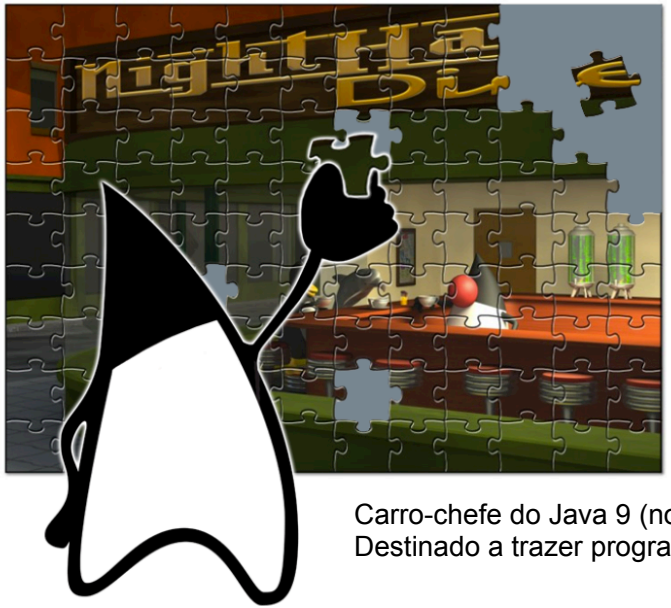


Módulos do JDK

- Um aplicação pode conter apenas alguns módulos (benefício para dispostivos que requerem executáveis menores)
- Único módulo obrigatório (é o core): java.base
  - Não há cláusula require no module-info.java

```
// module-info.java (module java.base)
module java.base {
        exports java.io;
        exports java.lang;
        exports java.lang.module;
        ...
        ...
        exports java.text;
        exports java.time;
        exports java.util;
        exports javax.net;
}
```

Principal feature no jdk 9: Jigsaw(JSR 376)



Carro-chefe do Java 9 (nosso objetivo aqui hoje):
Destinado a trazer programação modular

Forte Encapsulamento

- O módulo declara o que quer expor através do descritor do módulo

```
// src/java.sql/module-info.java
module java.sql {
  exports java.sql;
  exports javax.sql;
  exports javax.transaction.xa;
}
```

Estrutura de Pastas

```
src/com.apress.moduleA/
                              module-info.java
                              com/
                                    apress/
                                          moduleA/
                                                      Main.java
    com.apress.moduleB/
                              module-info.java
                              com/
                                    apress/
                                          moduleB/
                                                      ClassB1.java
                                                      ClassB2.java
    com.apress.moduleC/
                              module-info.java
                              com/
                                    apress/
                                          moduleC/
                                                      ClassC1.java
                                                      ClassC2.java
```

Mas o que são módulos?

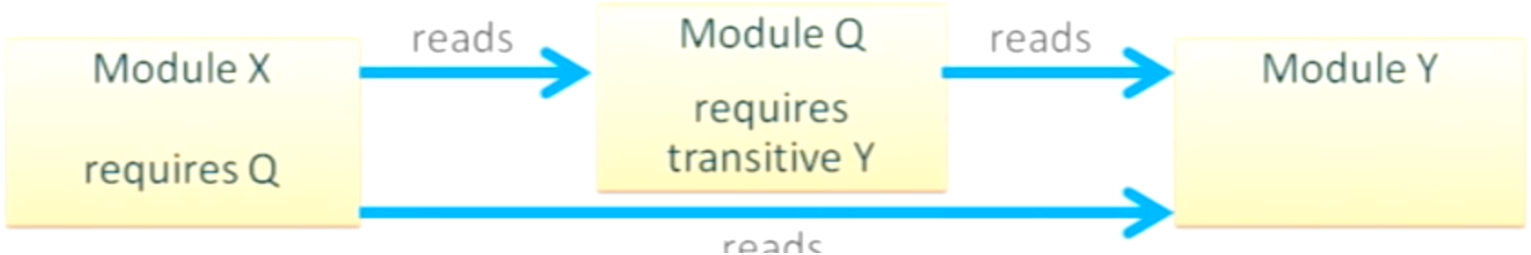
- Um novo tipo de componente java (Um nível de abstração maior, em cima de package)
  - Possui um nome e um código descritivo
- Deve informar se faz uso de tipos (classes, interfaces ou pacotes) de outros módulos
  - Como? Cláusula **requires**.
- Além disso, pode ser necessário informar ao compilador quais tipos desse módulo podem ser acessados por outros.
  - Como? Cláusula **exports**.


Cadê as ibagens?
Põe na tela as ibagens!

Mais IBAGENS (requires x requires transitive)

- X reads Y if:
  - X requires Y

  or
  - X reads Q, and Q requires transitive Y



```
// Main.java (module com.apress.moduleA)
package com.apress.moduleA;
import com.apress.moduleB.*;

public class Main {
        public static void main(String[] args) {
                Employee employee = new Employee("John", "Albert");
                System.out.println("First name is : " + employee.getFirstName());
                System.out.println("Last name is : " + employee.getLastName());
        }
}
```

```
// Employee.java (module com.apress.moduleB)
package com.apress.moduleB;
public class Employee {

        private String firstName;
        private String lastName;

        public Employee() {
        }

        public Employee(String firstName, String lastName) {
                this.firstName = firstName;
                this.lastName = lastName;
        }

        public String getFirstName() {
                return firstName;
        }

        public String getLastName() {
                return lastName;
        }
}
```

```
// module-info.java
module com.apress.moduleA {
        requires com.apress.moduleB;
}
```

```
// module-info.java
module com.apress.moduleB {

}
```

```
.\src\com.apress.moduleA\com\apress\moduleA\Main.java:3: error: package com.apress.moduleB does n
import com.apress.moduleB.*;
^
.\src\com.apress.moduleA\com\apress\moduleA\Main.java:8: error: cannot find symbol
                Employee employee = new Employee("John", "Albert");
                ^
  symbol:   class Employee
  location: class Main
.\src\com.apress.moduleA\com\apress\moduleA\Main.java:8: error: cannot find symbol
                Employee employee = new Employee("John", "Albert");
                                        ^
  symbol:   class Employee
  location: class Main
3 errors
```

Ou seja:

- A partir disso, o sistema de módulos localiza os módulos necessários e, ao contrário do sistema de classpath, garante que o código de um módulo acesse apenas os tipos dos módulos dos quais ele depende.

- Também evita forte dependência (API – IMPL)
    - Um módulo pode declarar que utiliza (**uses**) uma interface (tipo genérico) em vez de uma classe (tipo específico) de um determinado serviço cuja implementação é fornecida (**provided**) em tempo de execução por outro modulo.

Tem Muito MAIS

- Qualified names
- Módulos com nomes
- Outras cláusulas
  - Uses, open, provides
- Serviços

jShell

- Ferramenta de linha de comando para a linguagem Java
  - Possibilita a execução de declarações e expressões Java de forma interativa, sem classes ou métodos.
  - Ótima opção para iniciantes
  - Viabiliza experimentar algoritmos, criar protótipos ou até mesmo tentar utilizar uma nova API
- Faz parte do projeto Kulla

Exemplos práticos no jShell

```
-> int multiplicaPorDois(int x){
>>     return x*2;
>> }
|  Added method multiplicaPorDois(int)

-> multiplicaPorDois(2)
|  Expression value is: 4
|    assigned to temporary variable $6 of type int
```

```
->
int x = 45;
|  Added variable x of type int with initial value 45
```

```
-> 2+2
|  Expression value is: 4
|    assigned to temporary variable $3 of type int

-> x+$3
|  Expression value is: 49
|    assigned to temporary variable $4 of type int
```

NANCY LYRA

Consultora de Desenvolvimento | Professora de Programação

nlino@thoughtworks.com

@nanalyra

@nana.lyra