

Programação Funcional

Unidade 9 – Ambiente e Interoperabilidade com Java

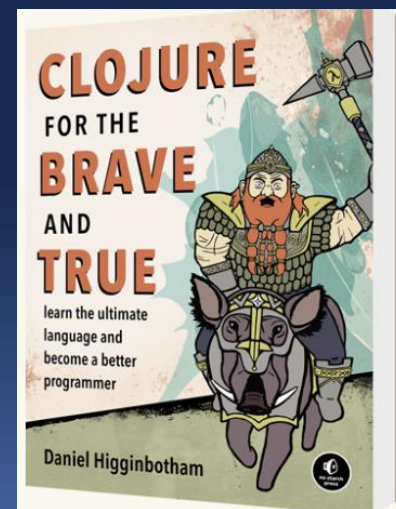
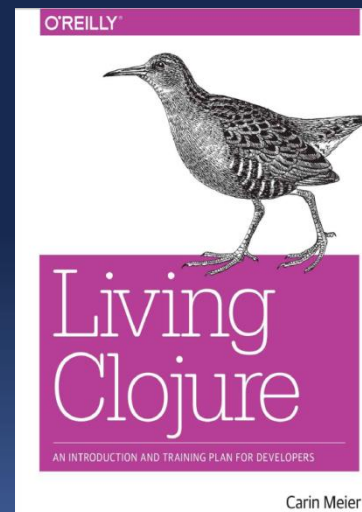
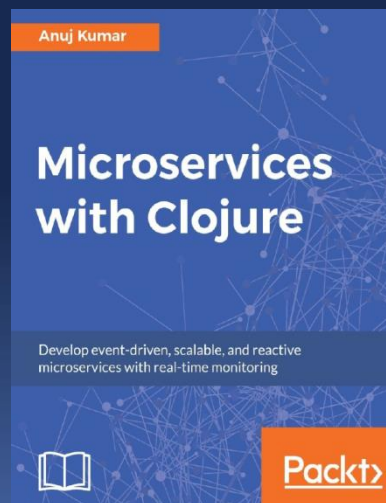
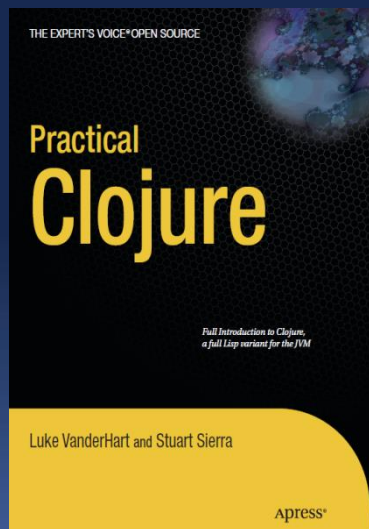
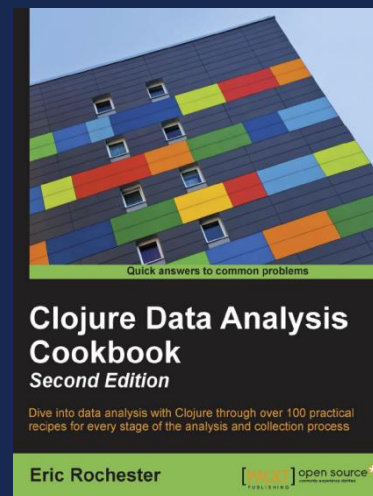
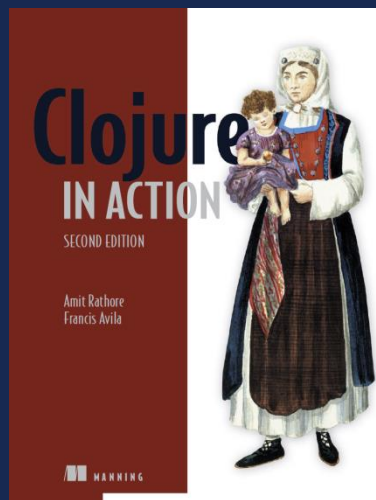
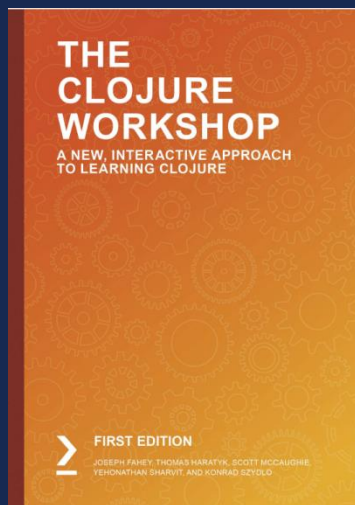


Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecido.freitas@prof.uscs.edu.br
aparecidovfreitas@gmail.com



Revisão Técnica: Mauricio Szabo
mauricio.szabo@gmail.com

Bibliografia



Namespaces

- ✓ Correspondem à forma pela qual o código **Clojure** é organizado;
- ✓ Pode-se pensar num **namespace** como se fosse um diretório que armazena um grupo de funções;
- ✓ Esses diretórios são independentes e essa independência nos ajuda a fornecer uma clara visão da estrutura do código;
- ✓ Considere duas funções com o **mesmo nome**. Elas apesar do mesmo nome, podem ter funcionalidades completamente diferentes. Apesar dos nomes iguais, essas funções podem existir em **namespaces** diferentes.



Namespaces

- ✓ Ao startar REPL, estamos posicionados no namespace **user**;
- ✓ Pode-se obter o namespace corrente por meio de `(ns-name *ns*)`;
- ✓ Programas clojure sempre estão em um namespace.

```
(ns-name *ns*)
```

```
user
```

Setando um novo namespace

- ✓ A melhor forma de se configurar um novo **namespace** é com o uso da macro `ns`;
- ✓ Por padrão, `ns` criará um novo **namespace** que contém mapeamentos para classes em **java.lang** e para as funções em `clojure.core`;

```
(ns namespace-mauricio)
| nil

(inc 3)

(ns-name *ns*)
| namespace-mauricio

(defn soma [x y] (+ x y))
| #'namespace-mauricio/soma ...

(namespace-mauricio/soma 4 5 )
| 9

(soma 4 6 )
| 10
```

Acessando funções de outro ns

- ✓ A função **require** permite que se acesse funções de bibliotecas externas;

unidade_08_3.cj

```
(split "A USCS está em São Caetano do Sul" #" ")  
  
java.lang.RuntimeException: "Unable to resolve symbol: split in this context"  
  in clojure.lang.Util.runtimeException (Util.java:221)  
  in clojure.lang.Compiler.resolveIn (Compiler.java:7414)  
  in clojure.lang.Compiler.resolve (Compiler.java:7358)  
  in clojure.lang.Compiler.analyzeSymbol (Compiler.java:7319)  
  in clojure.lang.Compiler.analyze (Compiler.java:6768)  
  in clojure.lang.Compiler.analyze (Compiler.java:6745)  
  in clojure.lang.Compiler$InvokeExpr.parse (Compiler.java:3820)  
  in ...  
  ...
```

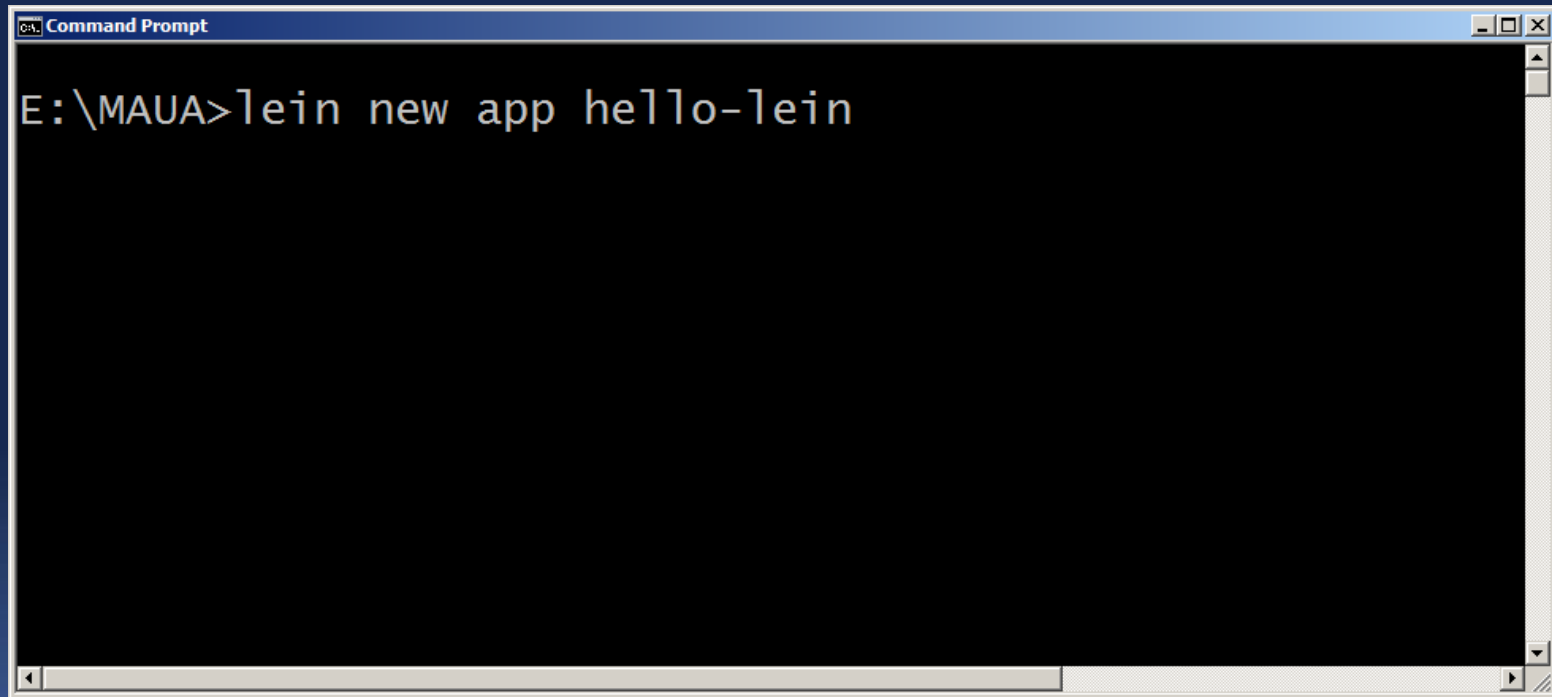
```
(require '[clojure.string :as str])
```

```
nil
```

```
(str/split "A USCS fica em São Caetano do Sul" #" ")  
=>["A" "USCS" "fica" "em" "São" "Caetano" "do" "Sul"]
```

Leiningen

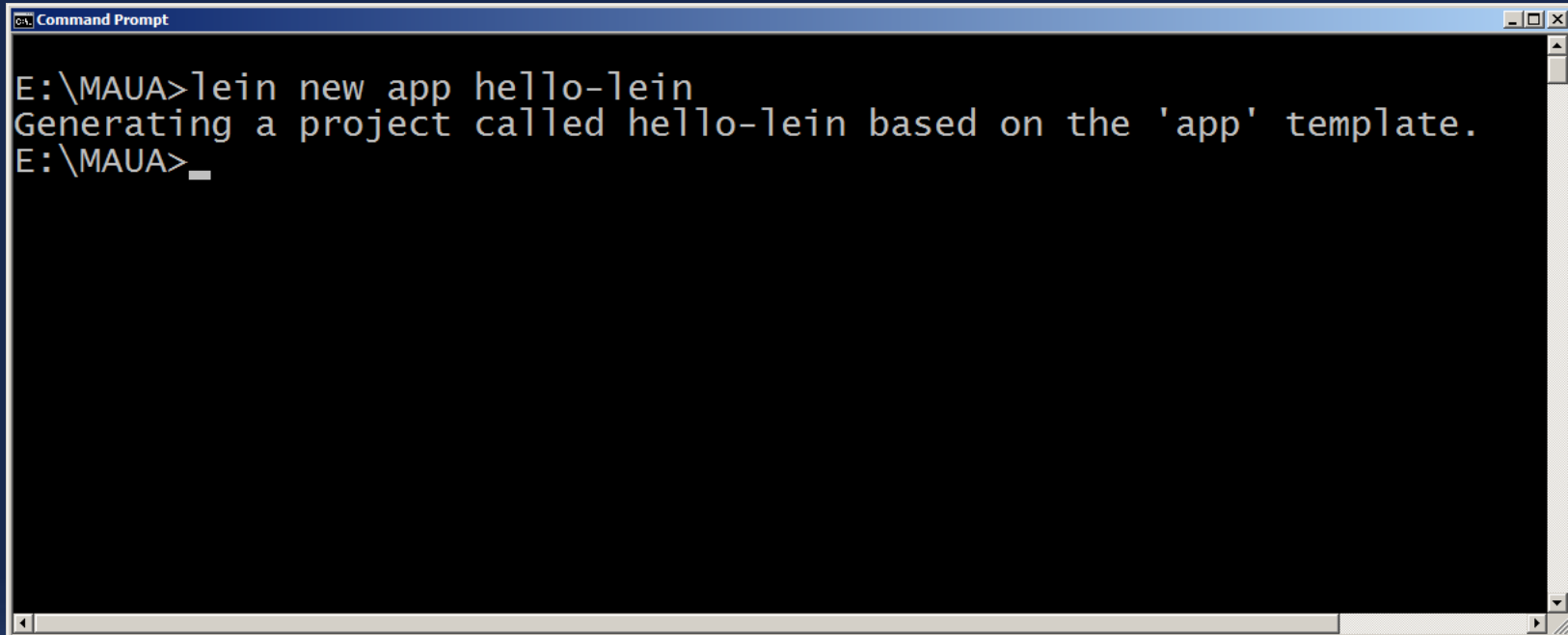
- ✓ Ferramenta de **build** muito popular na Comunidade **Clojure**;
- ✓ Provê diversos templates que facilitam a criação de projetos **Clojure**;
- ✓ Um desses templates é **app**.



```
Command Prompt
E:\MAUA>lein new app hello-lein
```


Leiningen

- ✓ O projeto foi criado pelo **Leiningen** por meio do template **app**;



```
Command Prompt
E:\MAUA>lein new app hello-lein
Generating a project called hello-lein based on the 'app' template.
E:\MAUA>
```


Navegando no projeto criado

```
Command Prompt
E:\MAUA\hello-lein>dir
Volume in drive E is Dados
Volume Serial Number is 14D9-25F0

Directory of E:\MAUA\hello-lein

23-Jul-20   11:59 PM      <DIR>          .
23-Jul-20   11:59 PM      <DIR>          ..
23-Jul-20   11:59 PM                124 .gitignore
23-Jul-20   11:59 PM                164 .hgignore
23-Jul-20   11:59 PM                798 CHANGELOG.md
23-Jul-20   11:59 PM      <DIR>          doc
23-Jul-20   11:59 PM           14,652 LICENSE
23-Jul-20   11:59 PM                408 project.clj
23-Jul-20   11:59 PM           1,027 README.md
23-Jul-20   11:59 PM      <DIR>          resources
23-Jul-20   11:59 PM      <DIR>          src
23-Jul-20   11:59 PM      <DIR>          test
                6 File(s)                17,173 bytes
                6 Dir(s)  350,552,141,824 bytes free

E:\MAUA\hello-lein>
```

Estrutura do Projeto criado

- ✓ O diretório **src** conterá o código fonte Clojure;
- ✓ O arquivo **project.clj** contém uma descrição do projeto;
- ✓ **README.md** é um entry point com informações da aplicação;
- ✓ Os testes serão feitos a partir do diretório **test**.

```
Command Prompt
E:\MAUA\hello-lein>dir
Volume in drive E is Dados
Volume Serial Number is 14D9-25F0

Directory of E:\MAUA\hello-lein

23-Jul-20  11:59 PM    <DIR>          .
23-Jul-20  11:59 PM    <DIR>          ..
23-Jul-20  11:59 PM                124 .gitignore
23-Jul-20  11:59 PM                164 .hgignore
23-Jul-20  11:59 PM                798 CHANGELOG.md
23-Jul-20  11:59 PM    <DIR>          doc
23-Jul-20  11:59 PM            14,652 LICENSE
23-Jul-20  11:59 PM                408 project.clj
23-Jul-20  11:59 PM            1,027 README.md
23-Jul-20  11:59 PM    <DIR>          resources
23-Jul-20  11:59 PM    <DIR>          src
23-Jul-20  11:59 PM    <DIR>          test
                6 File(s)            17,173 bytes
                6 Dir(s)   350,552,141,824 bytes free

E:\MAUA\hello-lein>
```

○ arquivo `project.clj`

```
(defproject hello-lein "0.1.0-SNAPSHOT"  
  :description "FIXME: write description"  
  :url "http://example.com/FIXME"  
  :license {:name "EPL-2.0 OR GPL-2.0-or-later WITH Classpath-exception-2.0"  
            :url "https://www.eclipse.org/legal/epl-2.0/" }  
  :dependencies [[org.clojure/clojure "1.10.1"]]  
  :main ^:skip-aot hello-lein.core  
  :target-path "target/%s"  
  :profiles {:uberjar {:aot :all}})
```

○ arquivo `project.clj`

- ✓ **Project Version:** versão do projeto
- ✓ **Description:** descrição do projeto
- ✓ **URL:** página com informações do projeto
- ✓ **License:** informações de licenciamento
- ✓ **Dependencies:** informações de dependências do projeto
- ✓ **Main namespace:** Define o namespace que é o entry point da aplicação
- ✓ **Ahead of time (AOT):** Permite que se compile o código antes de executá-lo
- ✓ **Profiles:** Permite a customização do projeto de acordo com as necessidades

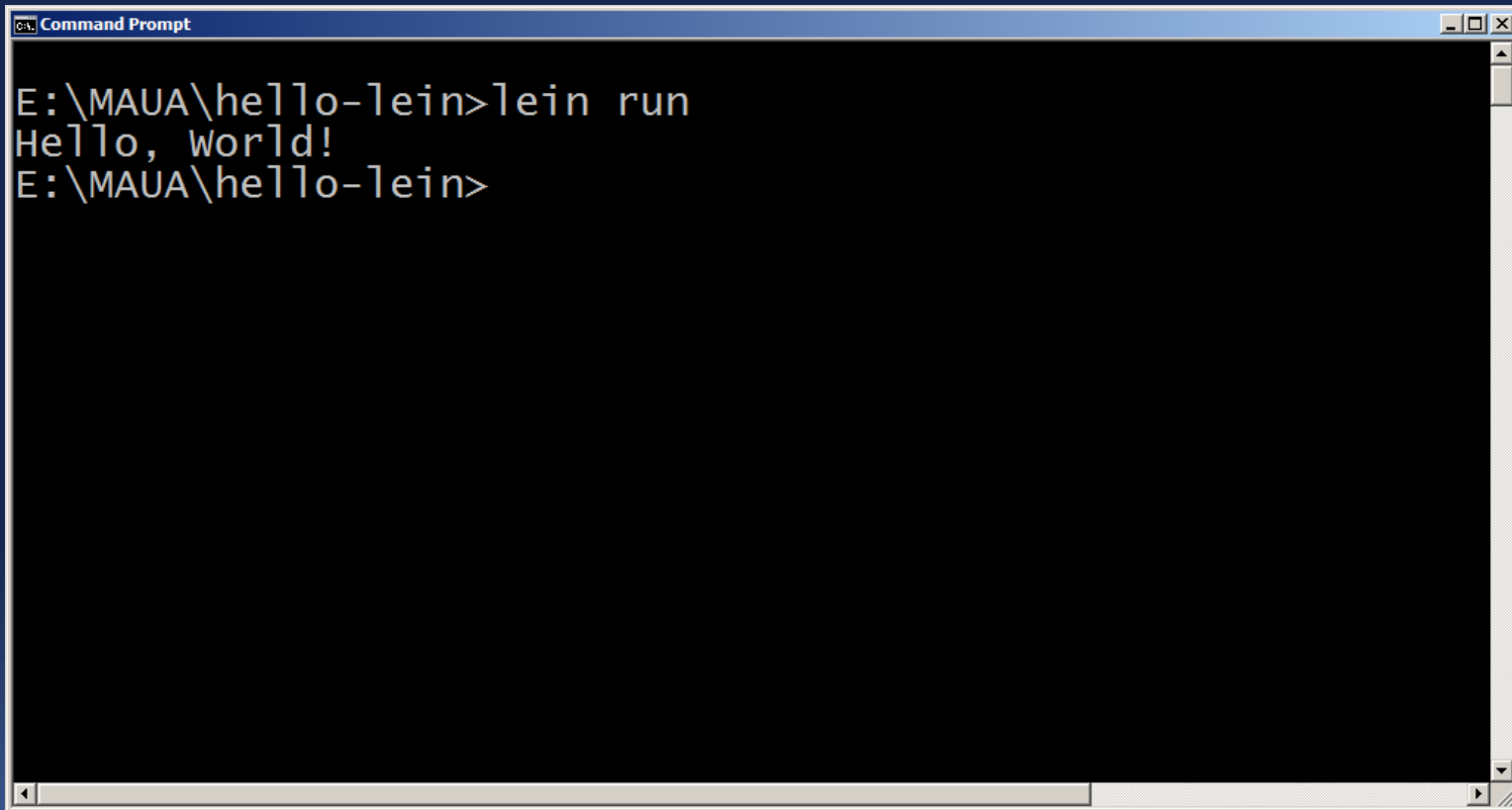
Execução da Aplicação

- ✓ Para executar a aplicação devemos chamar a task `run` ;
- ✓ A task `run` irá pesquisar no arquivo `project.clj` a keyword `:main` e o correspondente `namespace`;
- ✓ Por default, `Leiningen` pesquisará o `namespace` especificado em `:main`;
- ✓ No nosso caso, a pesquisa ocorrerá no `namespace` `hello-lein.core` ;
- ✓ Nesse `namespace`, temos a função `-main` que será chamada.

```
:main ^:skip-aot hello-lein.core
```

Execução da Aplicação

lein run



```
Command Prompt
E:\MAUA\hello-lein>lein run
Hello, world!
E:\MAUA\hello-lein>
```

Trabalhando com bibliotecas externas

- ✓ Bibliotecas são programas (**packaged**) que estão aptos para serem usados em outros projetos;
- ✓ Exemplos: **Ring**, uma biblioteca HTTP
clojure.java-time, para manipulação de datas;

Usando bibliotecas externas – Leiningen

- ✓ Vamos adicionar, como exemplo, uma biblioteca num projeto Leiningen e utilizá-la;
- ✓ Usaremos nesse exemplo, a biblioteca **clojure.java-time** para manipulação de data e hora;
- ✓ A primeira coisa a fazer é adicionar a dependência no arquivo `project.clj`;

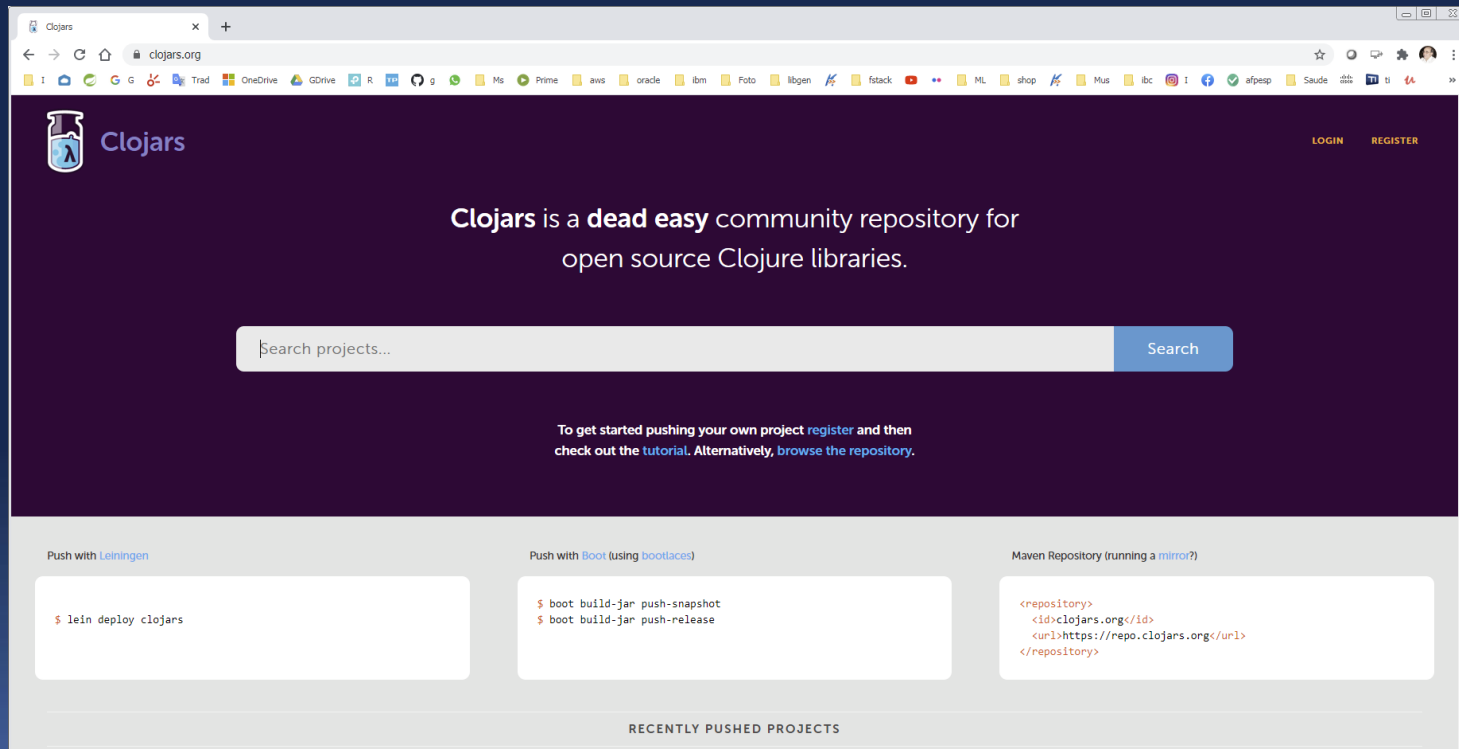
```
project.clj

(defproject hello-lein "0.1.0-SNAPSHOT"
  :description "FIXME: write description"
  :url "http://example.com/FIXME"
  :license {:name "EPL-2.0 OR GPL-2.0-or-later WITH Classpath-exception"
            :url "https://www.eclipse.org/legal/epl-2.0/"}
  :dependencies [[org.clojure/clojure "1.10.1"]
                 → [clojure.java-time "0.3.2"] ]
  :main ^:skip-aot hello-lein.core
  :target-path "target/%s"
  :profiles {:uberjar {:aot :all}}))
```

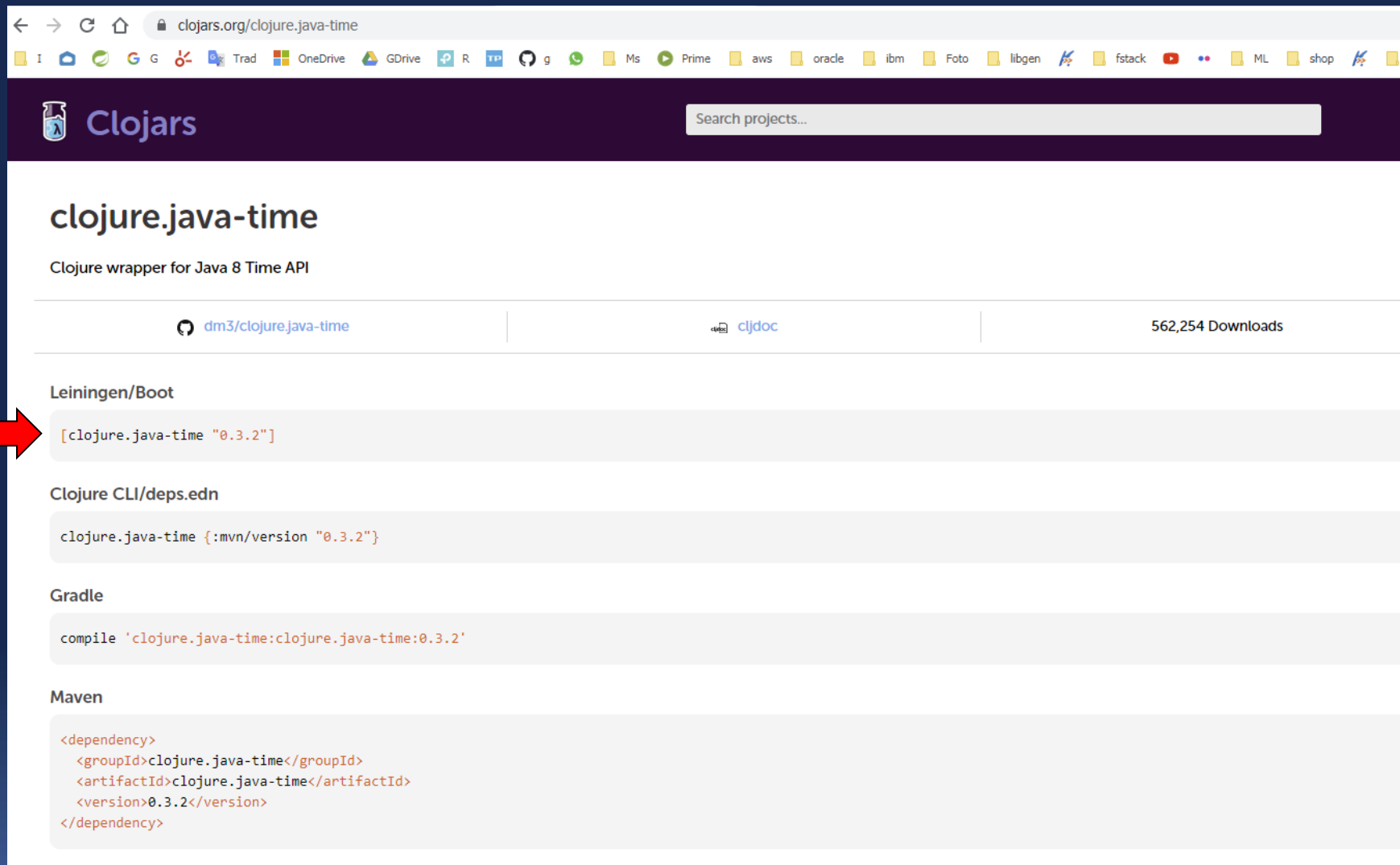
Usando bibliotecas externas – Leiningen

- ✓ O ecossistema Clojure tem um número muito grande de bibliotecas;
- ✓ Clojure provê um local central para pesquisa das bibliotecas disponíveis:

➔ <https://clojars.org>



Usando bibliotecas externas – Leiningen



The screenshot shows the Clojars website for the `clojure.java-time` project. The page includes a search bar, project details, and installation instructions for various build tools. A red arrow points to the Leiningen/Boot section.

clojure.java-time
Clojure wrapper for Java 8 Time API

dm3/clojure.java-time | cljdoc | 562,254 Downloads

Leiningen/Boot

```
[clojure.java-time "0.3.2"]
```

Clojure CLI/deps.edn

```
clojure.java-time {:mvn/version "0.3.2"}
```

Gradle

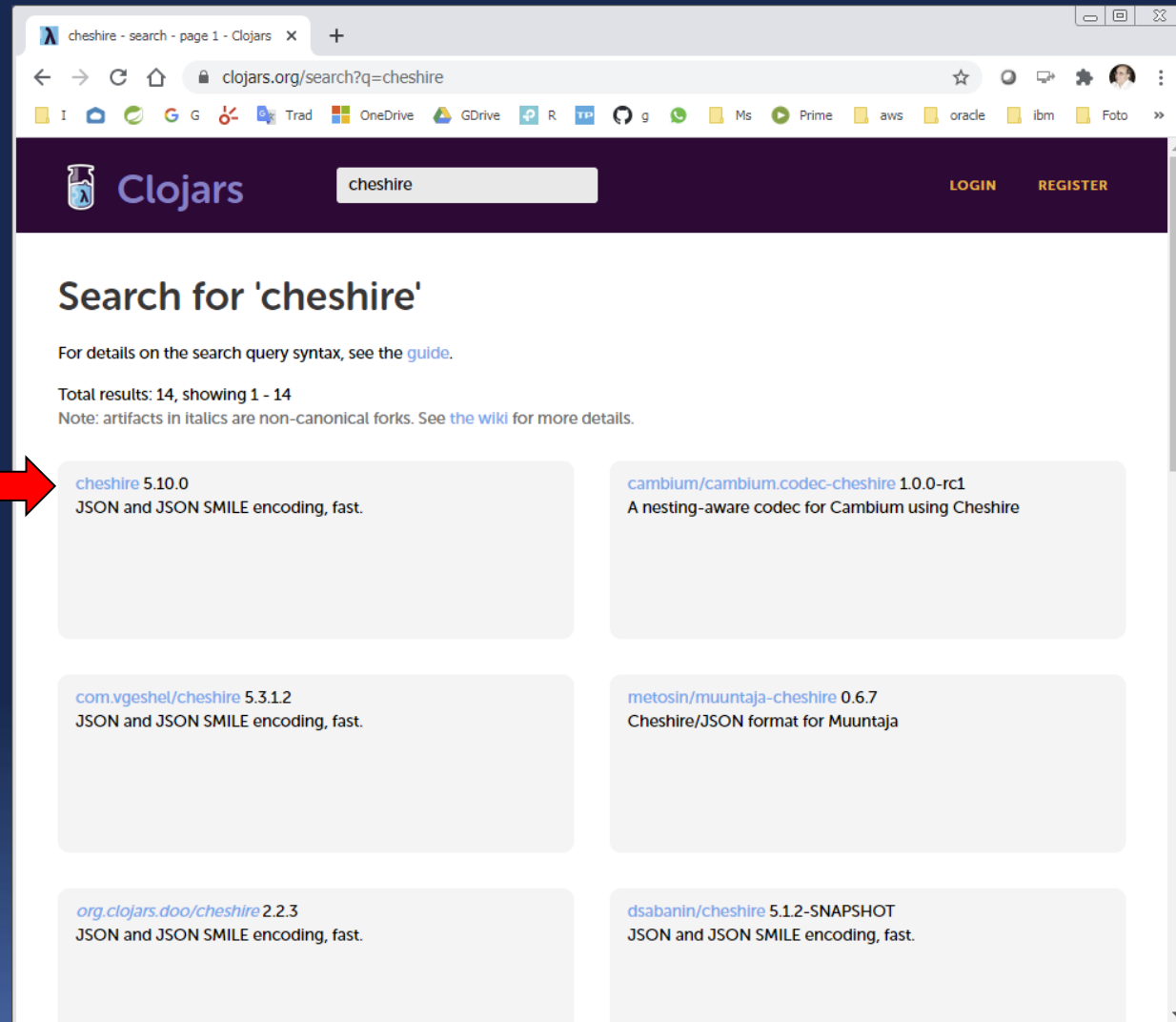
```
compile 'clojure.java-time:clojure.java-time:0.3.2'
```

Maven

```
<dependency>
  <groupId>clojure.java-time</groupId>
  <artifactId>clojure.java-time</artifactId>
  <version>0.3.2</version>
</dependency>
```

Exemplo bibliotecas externas – Leiningen

- ✓ A biblioteca cheshire provê funcionalidades para ler e gravar arquivos JSON.



Usando bibliotecas externas – Leiningen

- ✓ O próximo passo é **importar** a biblioteca para o nosso **core namespace**;
- ✓ Vamos alterar o arquivo **hello-lein.core**:

```
(ns hello-lein.core
  →( :require [java-time :as time]
      :gen-class))

(defn -main
  "I don't do a whole lot ... yet."
  [& args]
  (println "Hello, World!"))
```

Usando bibliotecas externas – Leiningen

- ✓ Finalmente, modificaremos a função `-main` para imprimir a hora local usando a função a partir da biblioteca `clojure.java-time`.

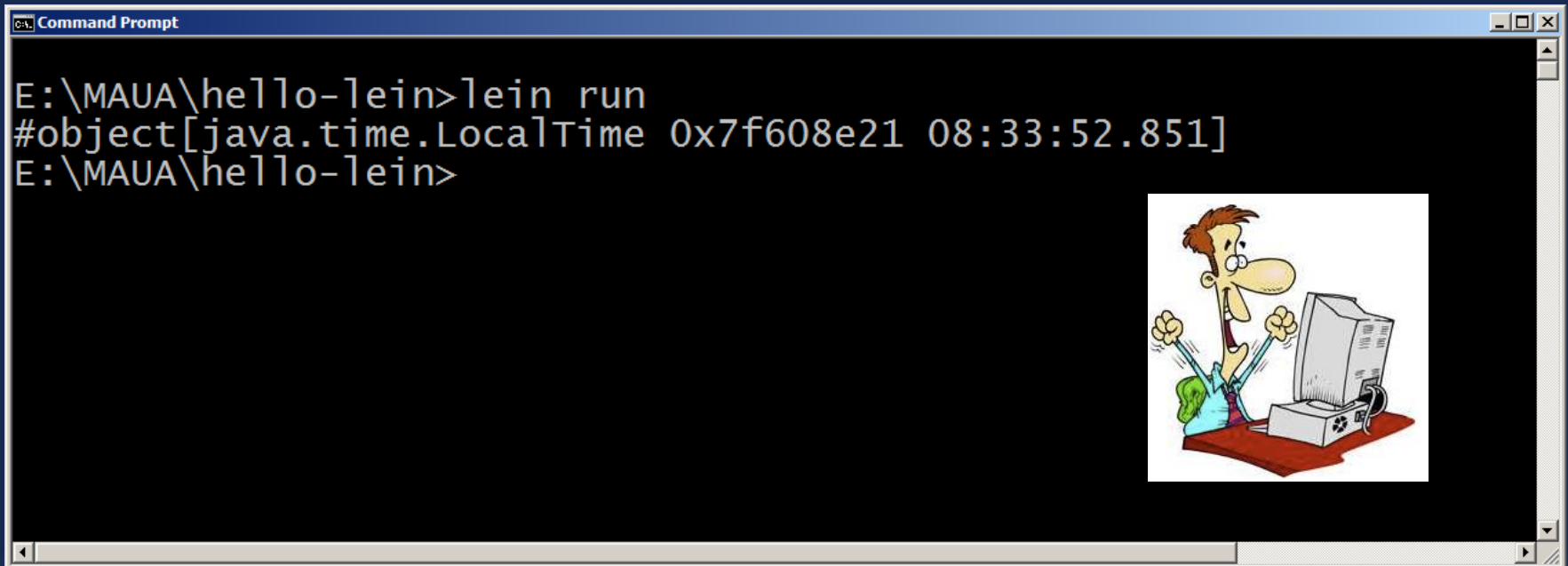
```
core.clj

(ns hello-lein.core
  (:require [java-time :as time])
  (:gen-class))

→ (defn -main
   "Exibe hora local corrente!"
   [& args]
   → (println (time/local-time)))
```

Executando a aplicação

lein run , no diretório da aplicação



```
Command Prompt
E:\MAUA\hello-lein>lein run
#object[java.time.LocalDateTime 0x7f608e21 08:33:52.851]
E:\MAUA\hello-lein>
```



Criando e executando um jar com Leiningen

- ✓ Os projetos em **Leiningen** são empacotados (packed) como **jar** files;
- ✓ **Leiningen** provê duas tasks para a criação de um jar: **uberjar** e **jar** ;
- ✓ Ambas as tasks criam um arquivo **zipped** com nosso código;
- ✓ A diferença é que uma **jar task** empacotará somente nosso código enquanto que **uberjar task** **também** empacotará as **dependências**;


:gen-class

- ✓ Observe no arquivo `hello-lein.core` na declaração do namespace a diretiva **:gen-class**;
- ✓ A diretiva **:gen-class** é um conceito importante em Clojure;
- ✓ Esta diretiva gerará as classes Java correspondentes ao namespace target;
- ✓ O resultado será a geração de uma classe Java em um arquivo **.class**;

Chamando a task uberjar na linha de comando

lein uberjar

```
Command Prompt
E:\MAUA\hello-lein>lein uberjar
Compiling hello-lein.core
Created E:\MAUA\hello-lein\target\uberjar\hello-lein-0.1.0-SNAPSHOT.jar
Created E:\MAUA\hello-lein\target\uberjar\hello-lein-0.1.0-SNAPSHOT-standalone.jar
E:\MAUA\hello-lein>_
```



Comparando o tamanho dos arquivos


```
Command Prompt
E:\MAUA\hello-lein\target\uberjar>dir
Volume in drive E is Dados
Volume Serial Number is 14D9-25F0

Directory of E:\MAUA\hello-lein\target\uberjar

24-Jul-20  08:56 AM    <DIR>          .
24-Jul-20  08:56 AM    <DIR>          ..
24-Jul-20  08:56 AM    <DIR>          classes
24-Jul-20  08:56 AM           5,692,562 hello-lein-0.1.0-SNAPSHOT-standalone.jar
24-Jul-20  08:56 AM           987,839 hello-lein-0.1.0-SNAPSHOT.jar
24-Jul-20  08:56 AM    <DIR>          stale
                2 File(s)          6,680,401 bytes
                4 Dir(s)   350,539,952,128 bytes free

E:\MAUA\hello-lein\target\uberjar>
```

Executando o jar file



```
Command Prompt
E:\MAUA\hello-lein>java -jar target/uberjar/hello-lein-0.1.0-SNAPSHOT-standalone.jar
#object[java.time.LocalDateTime 0x4de025bf 09:04:52.723]
E:\MAUA\hello-lein>
```





Interoperabilidade com Java



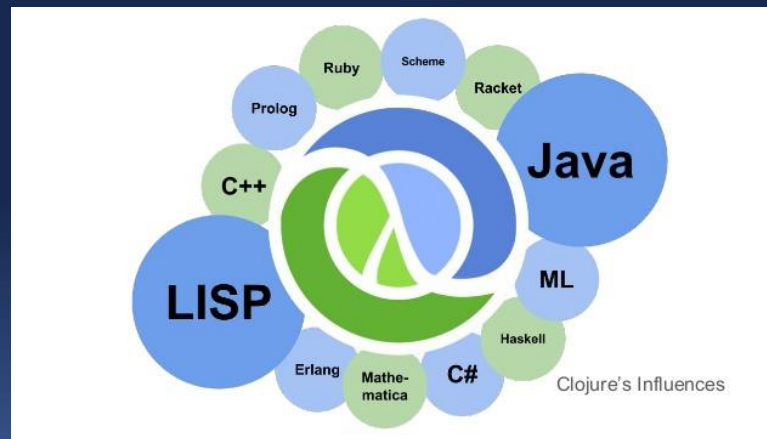
Interoperabilidade com Java

- ✓ Nos slides anteriores, aprendemos a criar um projeto **Leiningen**;
- ✓ Um projeto nos ajuda a organizar o nosso código;
- ✓ O nosso projeto está estruturado em **namespaces**;
- ✓ Criamos novos **namespaces** e importamos **bibliotecas externas** a fim de usá-las em nosso código;
- ✓ Vamos agora trabalhar com projetos que usam **Java** e **Javascript**.



Interoperabilidade com Java

- ✓ A Linguagem **Clojure** é compilada para bytecodes e é processada na **JVM**;
- ✓ Assim, por causa disso **Clojure** é chamada linguagem hosted;
- ✓ A importação de classes **Java** é diferente da importação de bibliotecas **Clojure**;



Usando Java em Clojure

- ✓ **Clojure** é hosted language. Isso significa que **Clojure** usa JVM ao invés de criar um novo ambiente de runtime;
- ✓ Assim, **Clojure** reusa todas as facilidades providas pela JVM;
- ✓ Esta é uma poderosa característica de Clojure;
- ✓ Coisas tais como Garbage Collection, threading, concorrência, operações de I/O que são largamente já testadas e consolidadas, são incorporadas ao **Clojure**.

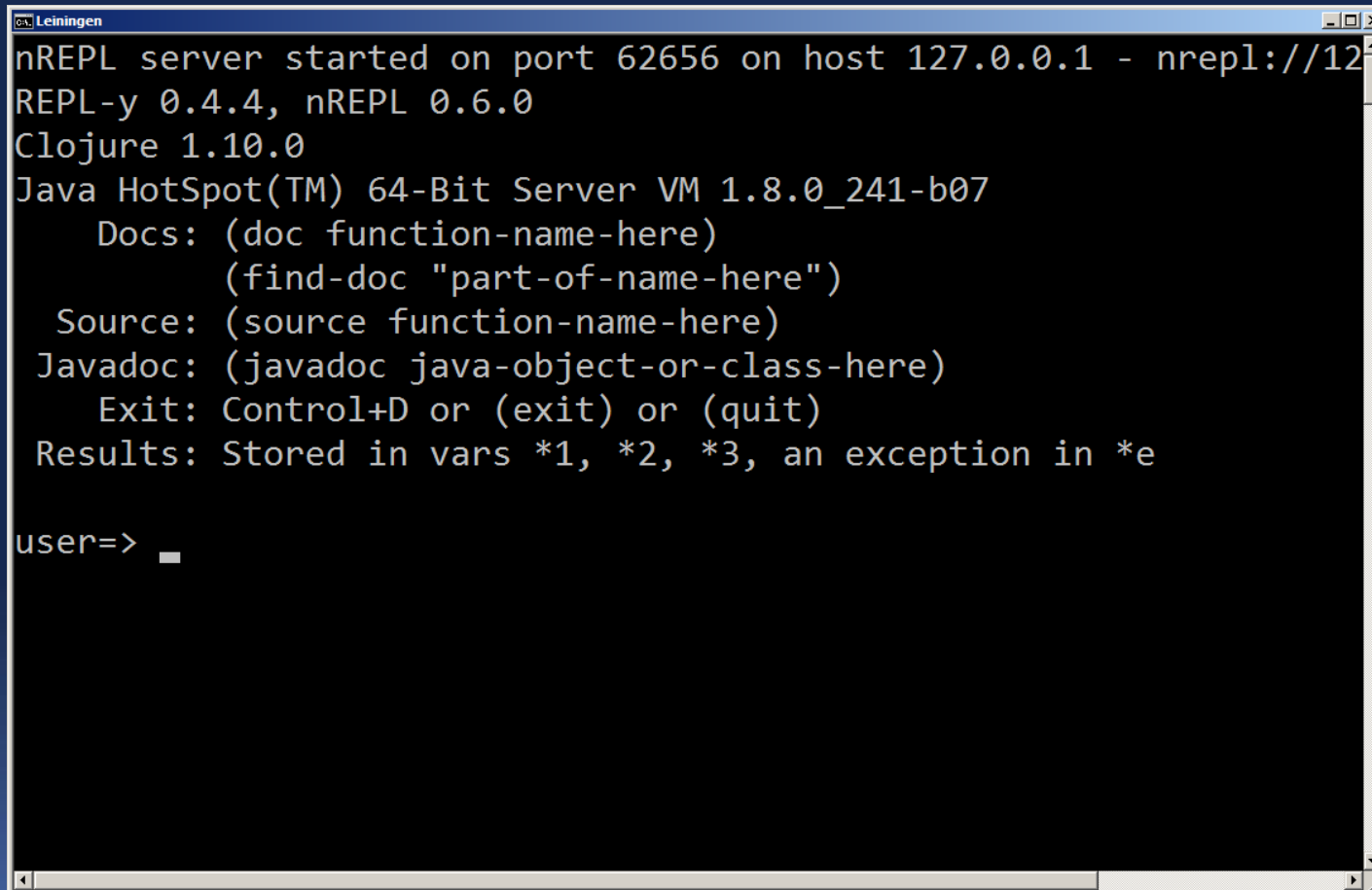


Usando Java em Clojure

- ✓ Assim, todo desenvolvedor **Clojure** tem acesso ao ecossistema de bibliotecas **JVM**;
- ✓ Além disso, considerando que **Java** é uma das mais populares linguagens de programação, há uma comunidade muito grande e atuante, trazendo para os desenvolvedores **Clojure** os benefícios de usar bibliotecas bem testadas e otimizadas;
- ✓ Veremos agora como importar a classe Java **BigDecimal**.

Importando a classe Java BigDecimal

lein repl



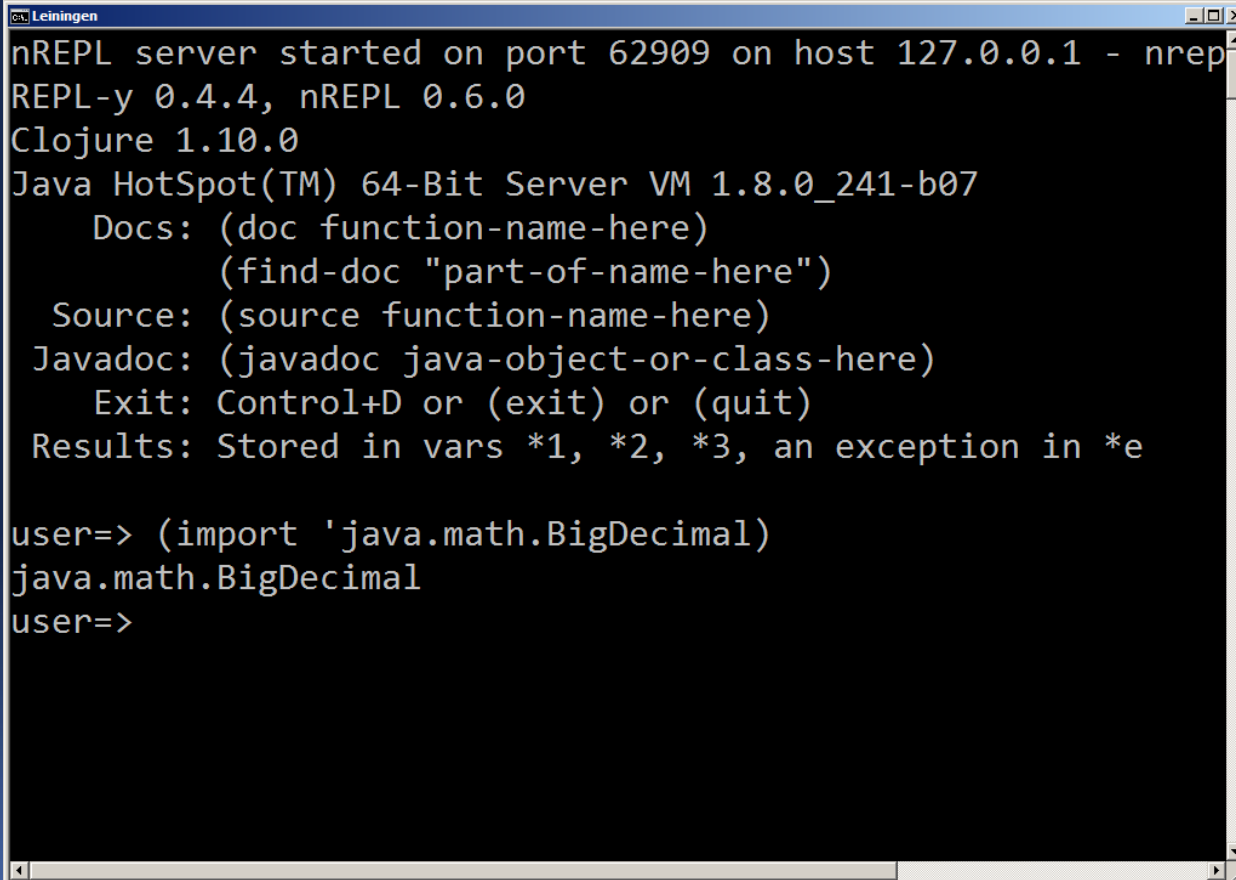
```
Leiningen
nREPL server started on port 62656 on host 127.0.0.1 - nrepl://127.0.0.1:62656
REPL-y 0.4.4, nREPL 0.6.0
Clojure 1.10.0
Java HotSpot(TM) 64-Bit Server VM 1.8.0_241-b07
  Docs: (doc function-name-here)
        (find-doc "part-of-name-here")
  Source: (source function-name-here)
  Javadoc: (javadoc java-object-or-class-here)
  Exit: Control+D or (exit) or (quit)
  Results: Stored in vars *1, *2, *3, an exception in *e

user=> _
```

Importando a classe Java BigDecimal

(import ...)

- ✓ A classe BigDecimal é chamada por meio da função import.



```
Leiningen
nREPL server started on port 62909 on host 127.0.0.1 - nrepl
REPL-y 0.4.4, nREPL 0.6.0
Clojure 1.10.0
Java HotSpot(TM) 64-Bit Server VM 1.8.0_241-b07
  Docs: (doc function-name-here)
        (find-doc "part-of-name-here")
  Source: (source function-name-here)
  Javadoc: (javadoc java-object-or-class-here)
  Exit: Control+D or (exit) or (quit)
  Results: Stored in vars *1, *2, *3, an exception in *e

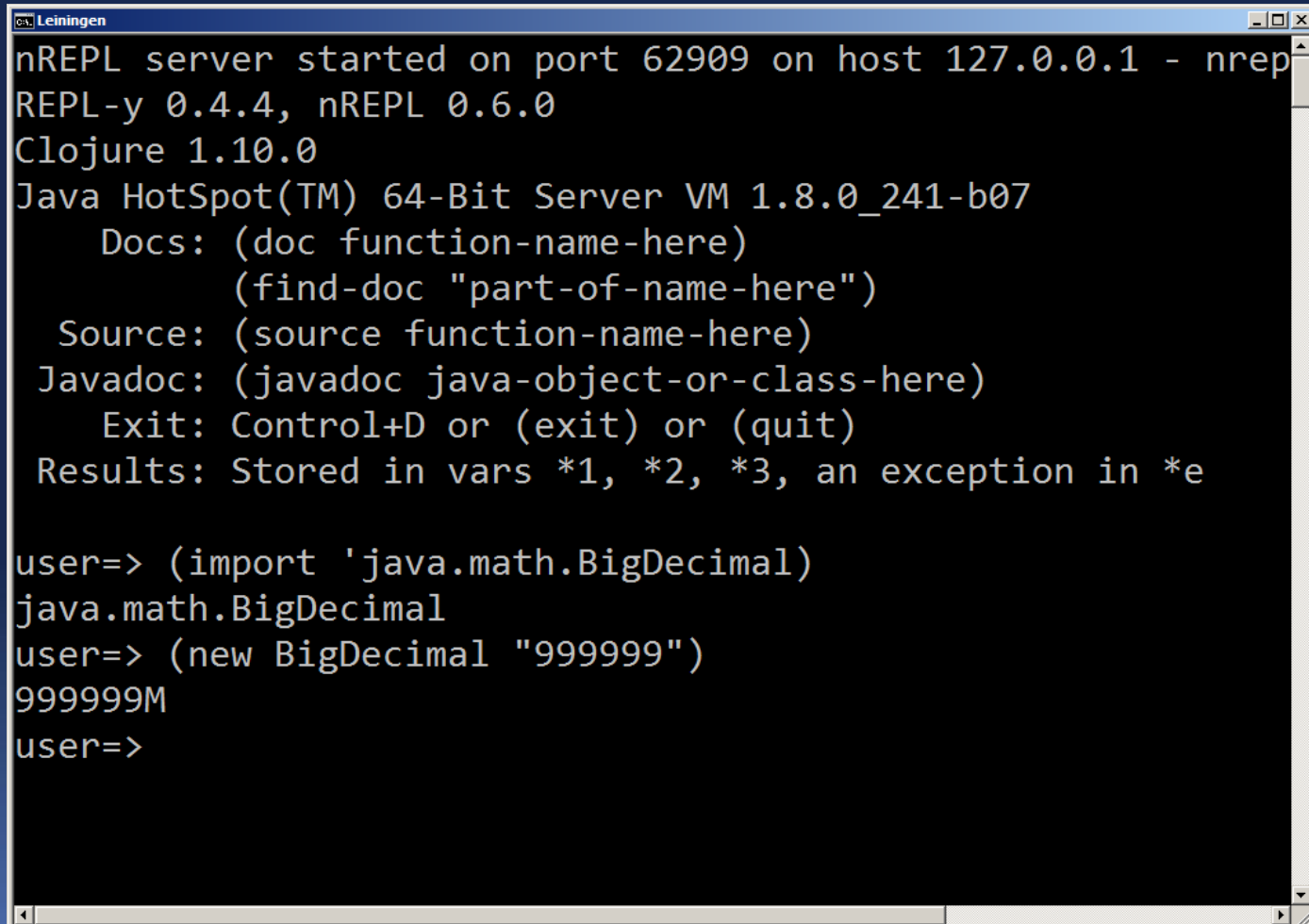
user=> (import 'java.math.BigDecimal)
java.math.BigDecimal
user=>
```

Importando a classe Java BigDecimal

- ✓ Em Java, criamos uma instância da classe BigDecimal por meio de:

`BigDecimal grandeNumero = new BigDecimal("999999");`

- ✓ Similarmente, em Clojure escrevemos: `(new BigDecimal "999999")`



```
Leiningen
nREPL server started on port 62909 on host 127.0.0.1 - nrepl
REPL-y 0.4.4, nREPL 0.6.0
Clojure 1.10.0
Java HotSpot(TM) 64-Bit Server VM 1.8.0_241-b07
  Docs: (doc function-name-here)
        (find-doc "part-of-name-here")
  Source: (source function-name-here)
  Javadoc: (javadoc java-object-or-class-here)
  Exit: Control+D or (exit) or (quit)
  Results: Stored in vars *1, *2, *3, an exception in *e

user=> (import 'java.math.BigDecimal)
java.math.BigDecimal
user=> (new BigDecimal "999999")
999999M
user=>
```



Importando a classe Java BigDecimal

- ✓ Se quisermos usar várias vezes, podemos fazer o binding em um símbolo

```
(def grande-numero (new BigDecimal "999999"))
```



```
Leiningen
user=> (import 'java.math.BigDecimal)
java.math.BigDecimal
user=> (new BigDecimal "999999")
999999M
user=> (def grande-numero (new BigDecimal "999999"))
#'user/grande-numero
user=> grande-numero
999999M
user=>
user=>
user=>
```