

Concepts des langages de programmation

Introduction à Clojure

Plan

- ▶ C'est quoi clojure ?
 - ▶ Types et structures de données
 - ▶ Fonctions et expressions dans Clojure
 - ▶ Commentaires en Clojure
 - ▶ Définition de variables et affectation
 - ▶ Instructions conditionnelles et itérations
 - ▶ Listes et fonctions
 - ▶ Quelques aspects avancés
 - ▶ Interaction avec Java
 - ▶ Exemples
 - ▶ Clojure et Netbeans
-

Plan

- ▶ **C'est quoi clojure ?**
 - ▶ Types et structures de données
 - ▶ Fonctions et expressions dans Clojure
 - ▶ Commentaires en Clojure
 - ▶ Définition de variables et affectation
 - ▶ Instructions conditionnelles et itérations
 - ▶ Listes et fonctions
 - ▶ Quelques aspects avancés
 - ▶ Interaction avec Java
 - ▶ Exemples
 - ▶ Clojure et Netbeans
-

C'est quoi Clojure ?

- ▶ Un langage de programmation **fonctionnel**
 - ▶ Met l'accent sur l'immutabilité
- ▶ Pas orienté objet
- ▶ Évaluation paresseuse
- ▶ Fonctions d'ordre supérieures
 - ▶ Une fonction peut prendre d'autres fonctions en tant que arguments
 - ▶ Une fonction peut retourner des fonctions en résultat

C'est quoi Clojure ? (suite)

- ▶ Typage dynamique
- ▶ Compilé : en byte-code de la JVM
 - ▶ Java 5 et plus
- ▶ Il existe aussi une version pour la plateforme .NET
 - ▶ ClojureCLR (version beta)
- ▶ C'est le lisp de la JVM et du .NET

C'est quoi Clojure ? (suite)

- ▶ Les blocks d'instructions sont délimités par des **parenthèses '(' ')'** au lieu des accolades de Java et Scala
- ▶ Notation **prefixe**
 - ▶ `(+ 1 4) ; 5`
 - ▶ `(+ 1 2 3) ; 6`

Plan

- ▶ C'est quoi clojure ?
 - ▶ **Types et structures de données**
 - ▶ Fonctions et expressions dans Clojure
 - ▶ Commentaires en Clojure
 - ▶ Définition de variables et affectation
 - ▶ Instructions conditionnelles et itérations
 - ▶ Listes et fonctions
 - ▶ Quelques aspects avancés
 - ▶ Interaction avec Java
 - ▶ Exemples
 - ▶ Clojure et Netbeans
-

Types de données atomiques

- ▶ Entiers : 1, 23, 4466677, ...
- ▶ Chaînes de caractères : "Ceci est une chaîne"
- ▶ Caractères : \a \b \c \d \z
- ▶ Symboles : :mon_symbol_1, :mon_symbol_2
- ▶ Booléens : true, false
- ▶ Null : nil
- ▶ Expressions régulières : #{a*b}
- ▶

Structures de données

► Listes

- (samedi , dimanche, lundi, mardi, mercredi, jeudi, vendredi)
- (1, 2, 3)
- (list 1, 2, 3)

► Vecteurs

- [samedi , dimanche, lundi, mardi, mercredi, jeudi, vendredi]
- [1, 2, 3]
- (vector 1 2 3)

Structures de données (suite)

- ▶ **Tables de hachage**

- ▶ `{:a 1, :b 2, :t 3}`

- ▶ **Ensembles**

- ▶ `#{1 2 3}`

Structures de données (suite)

- ▶ **Quelques fonctions utiles sur les listes :**
 - ▶ `(first '("one" "two" "three"))` ; retournera le premier élément (`one`)
 - ▶ `(rest '("one" "two" "three"))` ; retournera tous les éléments excepté le premier
 - ▶ `(cons 1 '(2 3))` ; retourna une nouvelle liste : `(1 2 3)`

Structures de données (suite)

- ▶ Récupérer l'élément correspondant à une clé d'une table de hachage
 - ▶ `(get {"a" 1, "b" 2, "c" 3} "a") ; retournera : 1`
 - ▶ `({"a" 1, "b" 2, "c" 3} "a") ; retournera : 1`
 - ▶ `(get {"a" 1, "b" 2, "c" 3} "d") ; retournera : nil`
 - ▶ `({"a" 1, "b" 2, "c" 3} "d") ; retournera : nil`
 - ▶ `(:a {:a 1, :b 2, :c 3}) ; retournera 1`

Plan

- ▶ C'est quoi clojure ?
 - ▶ Types et structures de données
 - ▶ **Fonctions et expressions dans Clojure**
 - ▶ Commentaires en Clojure
 - ▶ Définition de variables et affectation
 - ▶ Instructions conditionnelles et itérations
 - ▶ Listes et fonctions
 - ▶ Quelques aspects avancés
 - ▶ Interaction avec Java
 - ▶ Exemples
 - ▶ Clojure et Netbeans
-

Fonctions dans Clojure

► Définition de fonction en **Clojure** :

- `defn(fonction [x]`
 `(+ x 10)`
 `)`

► Définition de fonction en **Java** :

- `int fonction(int x) {`
 `return x + 10;`
 `}`

► Appel de fonction en **Clojure** :

- `(nom-fonction arg1 arg2 arg3)`
- `(fonction 5)`

► Appel de fonction en **Java** :

- `nom-fonction(arg1, arg2, arg3);`
- `fonction(5);`

Fonctions dans Clojure (suite)

► Affichage de chaînes de caractères :

► Clojure :

```
► (defn hello [name]
  (
    println "Hello,"    name
  )
)
```

► Java :

```
► public void hello(String name) {
  System.out.println("Hello, " +
    name);
}
```

(println "Hello," name) == (println (str("Hello," name)))
str : concatine des chaînes de caractères

Expressions Clojure (suite)

► Expressions en Clojure :

- `(+ 3 5)` ; résultat : 8
- `(< 2 4)` ; résultat : true
- `(+ 3 5 56 7 89)` ; résultat : 160
- `(+)` ; résultat : 0
- `(*)` ; résultat : 1
 - Si aucun argument n'est fourni à une fonction qui nécessite un, l'élément identité est retourné

► Expressions en Java :

- `3 + 5;`
- `2 < 4 ;`
- `3 + 5 + 56 + 7 + 89 ;`
- Pas d'équivalent

Plan

- ▶ C'est quoi clojure ?
 - ▶ Types et structures de données
 - ▶ Fonctions et expressions dans Clojure
 - ▶ **Commentaires en Clojure**
 - ▶ Définition de variables et affectation
 - ▶ Instructions conditionnelles et itérations
 - ▶ Listes et fonctions
 - ▶ Quelques aspects avancés
 - ▶ Interaction avec Java
 - ▶ Exemples
 - ▶ Clojure et Netbeans
-

Commentaires en Clojure

- ▶ ; ceci est un commentaire sur une seule ligne
- ▶ (comment
Ceci est un commentaire
sur plusieurs lignes
)

Plan

- ▶ C'est quoi clojure ?
 - ▶ Types et structures de données
 - ▶ Fonctions et expressions dans Clojure
 - ▶ Commentaires en Clojure
 - ▶ **Définition de variables et affectation**
 - ▶ Instructions conditionnelles et itérations
 - ▶ Listes et fonctions
 - ▶ Quelques aspects avancés
 - ▶ Interaction avec Java
 - ▶ Exemples
 - ▶ Clojure et Netbeans
-

Définition de variables et affectation

▶ Définition de variables en Clojure :

- ▶ `(def i 10)`
- ▶ `(def chaine "Salut")`

▶ Définition de variables en Java :

- ▶ `int i 10;`
- ▶ `String chaine = "Salut";`

Définition de variables et affectation (suite)

► Comment effectuer une affectation ?

► Deux possibilités :

1. Créer une nouvelle variable

□ Exemple :

- `(def a 5)`
- `(def a (+ 1 a))`
- `(println a)`

2. Utiliser la macro **let** pour créer une variable locale

□ Exemple :

- `(def a 5)`
- `(let [a (+ 1 a)] (println a)) ; valeur de a est : 6`
- `(println a) ; valeur de a est : 5`

Plan

- ▶ C'est quoi clojure ?
 - ▶ Types et structures de données
 - ▶ Fonctions et expressions dans Clojure
 - ▶ Commentaires en Clojure
 - ▶ Définition de variables et affectation
 - ▶ **Instructions conditionnelles et itérations**
 - ▶ Listes et fonctions
 - ▶ Quelques aspects avancés
 - ▶ Interaction avec Java
 - ▶ Exemples
 - ▶ Clojure et Netbeans
-

Instructions conditionnelles

- ▶ If else :
 - ▶ (**if** condition then-expression else-expression)
 - ▶ (if (< 4 2) (println "4 inférieur à 2") (println "2 inférieur à 4"))
- ▶ When (If sans else) :
 - ▶ (**when** condition expression)
 - ▶ Équivalent à : (**if** condition then-expression)
 - ▶ (when (< 2 4) (println "2 inférieur à 4"))
- ▶ When not :
 - ▶ (**when-not** condition expression)
 - ▶ (when-not (< 4 2) (println "4 n'est pas inférieur à 2"))

Exemple d'utilisation de l'instruction conditionnelle if

► Calcul du factoriel :

► Clojure :

```
► (defn fac [n]
  (if (== n 1)
      1
      (* n (fac (- n 1)))))
)
```

► Java :

```
► public int fac(int n) {
  if (n == 1)
    return 1;
  else
    return n * fac(n-1);
}
```


Instructions conditionnelles (suite)

- ▶ Plusieurs conditions et expressions :
 - ▶ (**cond** condition1 expression1 condition2 expression2 ... true expressionN)
 - ▶ Seulement l'expression associée à la première condition évaluée à true est exécutée
 - ▶ (**condp** predicat condition1 expression1 condition2 expression2 ... true expressionN)
 - ▶ Seulement l'expression associée à la première condition évaluée avec predicat à true est exécutée

Exemple d'utilisation de **cond**

► Fibonacci :

```
► (defn fib [n]
  (cond
    (== 0 n) 0
    (== 1 n) 1
    (< 1 n) (+ (fib (- n 1)) (fib (- n 2)))
  )
)
```

Exemple d'utilisation de **condp**

```
▶ (
  let [a 5]
  (condp == a
    1 (println "Salut" )
    5 (println "Bonjour")
  )
)
```

Instructions itératives

▶ while :

- ▶ (**while** condition expression modification_condition)

- ▶ Exemple

- ▶ (def i 5) (while (> i 0) (println "i")(- i 1))

▶ dotimes :

- ▶ (**dotimes** [x nombre-fois] expression) ; x : indice de l'itération courante

- ▶ Exemple

- ▶ (dotimes [i 5] (println "i"))

Instructions itératives (suite)

► loop :

- (**loop** [variable valeur_initiale] code **recur** (**inc|dec** variable))

► Exemple

- (loop [i 0]
 (when (< i 5)
 (println i)
 (recur (inc i))
)
)

Instructions itératives (suite)

- ▶ loop avec un pas d'incrémentation différent de 1:

- ▶ (**loop** [variable valeur_initiale] code **recur** (**op** variable nombre_incr_ou_dec))

- ▶ Exemple

- ▶ (loop [i 0]
 (when (< i 5)
 (println i)
 (recur (+ i 2))
)
)

Plan

- ▶ C'est quoi clojure ?
 - ▶ Types et structures de données
 - ▶ Fonctions et expressions dans Clojure
 - ▶ Commentaires en Clojure
 - ▶ Définition de variables et affectation
 - ▶ Instructions conditionnelles et itérations
 - ▶ **Listes et fonctions**
 - ▶ Quelques aspects avancés
 - ▶ Interaction avec Java
 - ▶ Exemples
 - ▶ Clojure et Netbeans
-

Listes et fonctions

- ▶ Les listes se déclarent avec la syntaxe :
 - ▶ `(1 2 4)`
- ▶ Lorsque Clojure rencontre cette syntaxe, il essaie de l'évaluer (en tant que fonction) :
 - ▶ Essaie d'appeler la fonction 1 pour laquelle on a associé les attributs 2 et 4
 - ▶ Ce qui génère l'erreur suivante :
 - ▶ `user=> (1 2 3)`
`#<CompilerException java.lang.ClassCastException: java.lang.Integer cannot be cast to clojure.lang.IFn (NO_SOURCE_FILE:0)>`
- ▶ Pour dire à Clojure que ce n'est pas une fonction, on précède la déclaration du caractère " `'` "
 - ▶ `'(1 2 4)`

Plan

- ▶ C'est quoi clojure ?
 - ▶ Types et structures de données
 - ▶ Fonctions et expressions dans Clojure
 - ▶ Commentaires en Clojure
 - ▶ Définition de variables et affectation
 - ▶ Instructions conditionnelles et itérations
 - ▶ Listes et fonctions
 - ▶ **Quelques aspects avancés**
 - ▶ Interaction avec Java
 - ▶ Exemples
 - ▶ Clojure et Netbeans
-

Fonctions avec un nombre variables d'arguments

- ▶ Clojure supporte le polymorphisme :
 - ▶ Fonctions portant le même nom mais avec :
 - ▶ Un nombre de paramètres variable
 - ▶ Un corps différent

- ▶ Deux façons différentes

- ▶ Première possibilité :

```
(defn nom_fonction  
  ([params1] corps1 )  
  ([params2] corps2 )  
  ....  
)
```

Fonctions avec un nombre variables d'arguments (suite)

► Exemple

```
(defn say_hello  
  ([nom]  
   (println "salut " nom "!"))  
  )  
  
  ([]  
   (println "Salut anonyme !"))  
  )  
)
```

Fonctions avec un nombre variable d'arguments (suite)

- ▶ Même exemple écrit autrement (plus compact)

```
(defn say_hello  
  
  ([nom]  
    (println "salut " nom "!"))  
  )  
  
  ([]  
    (say_hello ""))  
  )  
)
```

Fonctions avec un nombre variables d'arguments (suite)

- ▶ Deuxième possibilité :

```
(defn nom_fonction  
  [& params]  
  corps  
)
```

- ▶ Exemple :

```
(defn compter_nombre_params  
  [& params]  
  (count params)  
)
```

Fonctions anonymes

- ▶ Une fonction anonyme est une fonction qui n'a pas de nom
- ▶ Dans Clojure, une fonction anonyme se déclare avec le mot clé **fn**
- ▶ Pourquoi les fonctions anonymes ?
 - ▶ La fonction est seulement utilisée à l'intérieur d'une autre fonction
- ▶ Les fonctions anonymes sont optionnelles et ne sont donc pas indispensables
 - ▶ Les utiliser si elle rendent votre code plus lisible

Fonctions anonymes (suite)

- ▶ Syntaxe d'une fonction anonyme :
 - ▶ (fn [parametres] corps_de_la_fonction)
- ▶ Exemple :
 - ▶ (
 - ▶ (fn [x] (* 10 x))
 - ▶ 20
 - ▶)
 - ▶ ; retournera : 200

Map

- ▶ Utilisation de map :
 - ▶ `((if true dec) 3)` ; si true alors appeler la fonction dec sur 3
 - ▶ Appeler la fonction dec sur plusieurs éléments (d'un vecteur ou d'une liste) :
 - ▶ `(map (if true dec) [1 2 3])` ; retournera : `(0 1 2)`
 - ▶ `(map (if true dec) '(1 2 3))` ; même résultat : `(0 1 2)`

Comment exécuter plusieurs instructions de manière consécutive ?

- ▶ Utiliser **do**
- ▶ Do permet d'enchaîner plusieurs instructions et évaluer la dernière expression comme étant la valeur de retour
- ▶ Exemples
 - ▶ `((println "salut 1")(println "salut 2"))`
 - Retournera :
 - salut 1
 - salut 2
 - `#<CompilerException java.lang.NullPointerException (NO_SOURCE_FILE:0)>`

Comment exécuter plusieurs instructions de manière consécutive ? (suite)

► Exemples (suite)

► (do (println "salut 1")(println "salut 2"))

□ Retournera :

□ salut 1

□ salut 2

□ nil

► (do (println "salut 1")(println "salut 2") (+ 4 1))

□ Retournera :

□ salut 1

□ salut 2

□ 5

Remarque : **when** fait la même chose que **do** et retourne la valeur évaluée seulement si la condition est vérifiée

Plan

- ▶ C'est quoi clojure ?
 - ▶ Types et structures de données
 - ▶ Fonctions et expressions dans Clojure
 - ▶ Commentaires en Clojure
 - ▶ Définition de variables et affectation
 - ▶ Instructions conditionnelles et itérations
 - ▶ Listes et fonctions
 - ▶ Quelques aspects avancés
 - ▶ **Interaction avec Java**
 - ▶ Exemples
 - ▶ Clojure et Netbeans
-

Appel de méthodes de la classe String de Java à partir de Clojure

- ▶ Les méthodes de la classes String de Java peuvent être appelées directement à partir de Clojure (à l'exception de toString)
- ▶ Exemples :
 - ▶ `(.toUpperCase "salut")` ; retournera : SALUT
 - ▶ `(.replace (.toUpperCase "abc") "B" "-")` ; retournera : A-C
 - ▶ Le point (.) indique à Clojure qu'on appelle une méthode Java

Plan

- ▶ C'est quoi clojure ?
 - ▶ Types et structures de données
 - ▶ Fonctions et expressions dans Clojure
 - ▶ Commentaires en Clojure
 - ▶ Définition de variables et affectation
 - ▶ Instructions conditionnelles et itérations
 - ▶ Listes et fonctions
 - ▶ Quelques aspects avancés
 - ▶ Interaction avec Java
 - ▶ **Exemples**
 - ▶ Clojure et Netbeans
-

Quelques exemples

- ▶ Calcul de la somme
- ▶ Calcul de la moyenne

Calcul de la somme

```
► (defn sum
  ([]
   0
  )
  ([x]
   x
  )
  ([x & more]
   (+ x (apply sum more) )
  )
)
```

Calcul de la moyenne

```
► (defn average
  ([] 0)
  ([x] x)
  ([x & more]
    (/ (+ x (apply sum more)) (+ 1 (count more))))
  )
)
```

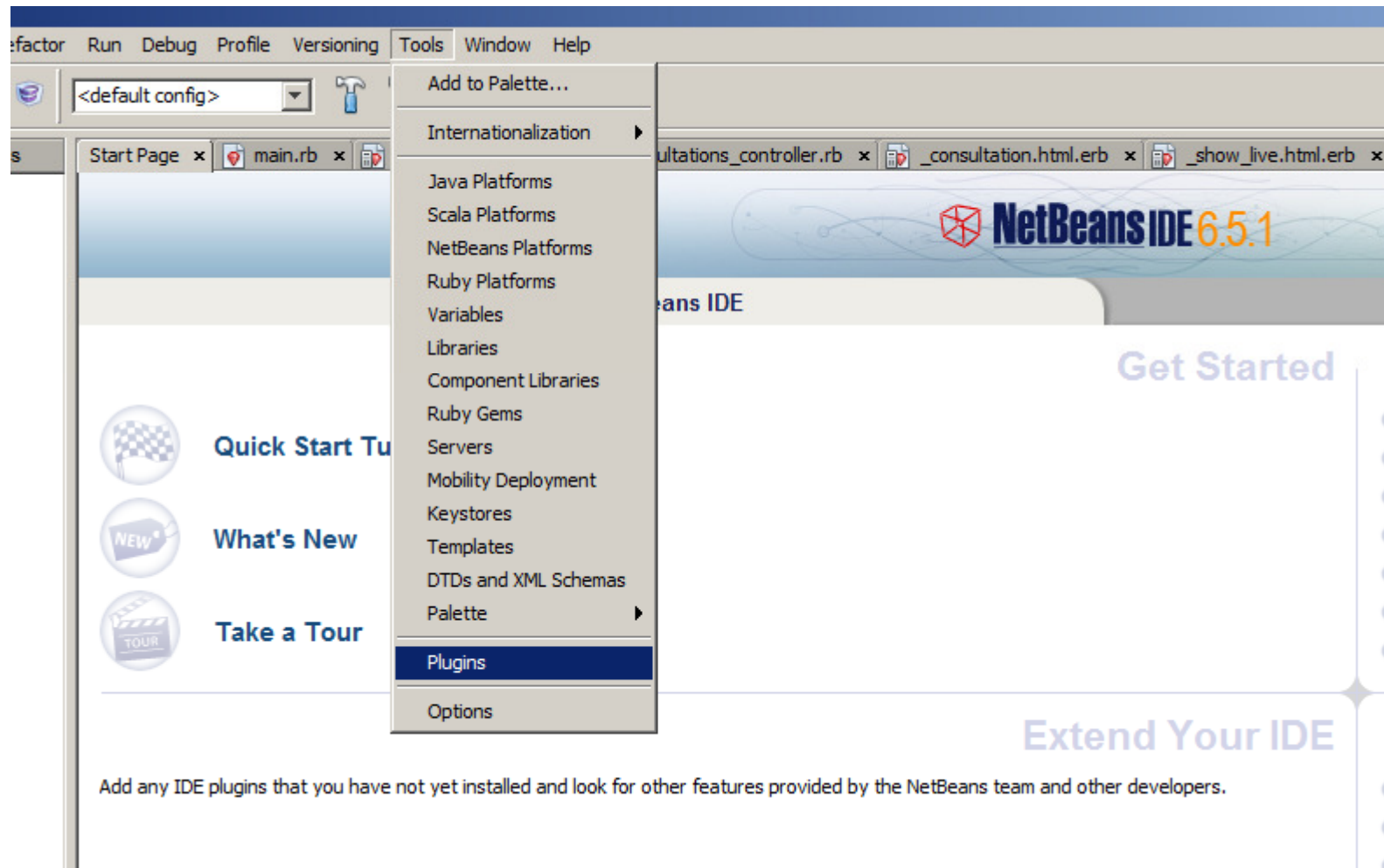

Plan

- ▶ C'est quoi clojure ?
 - ▶ Types et structures de données
 - ▶ Fonctions et expressions dans Clojure
 - ▶ Commentaires en Clojure
 - ▶ Définition de variables et affectation
 - ▶ Instructions conditionnelles et itérations
 - ▶ Listes et fonctions
 - ▶ Quelques aspects avancés
 - ▶ Interaction avec Java
 - ▶ Exemples
 - ▶ **Clojure et Netbeans**
-

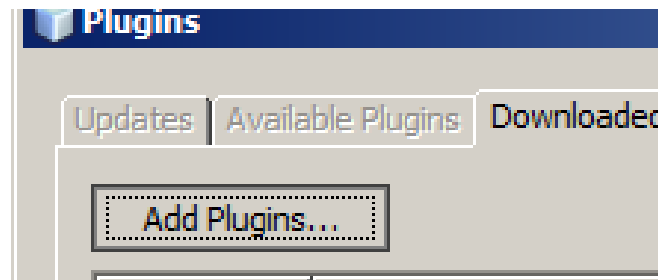
Clojure et Netbeans

- ▶ Un plugin Netbeans pour programmer avec Clojure est disponible
- ▶ Installation du plugin :
 - ▶ Télécharger le plugin Netbeans pour Clojure de la page Web du cours
 - ▶ Décompresser l'archive
 - ▶ Suivre les étapes indiquées par les captures d'écran des diapositifs qui suivent

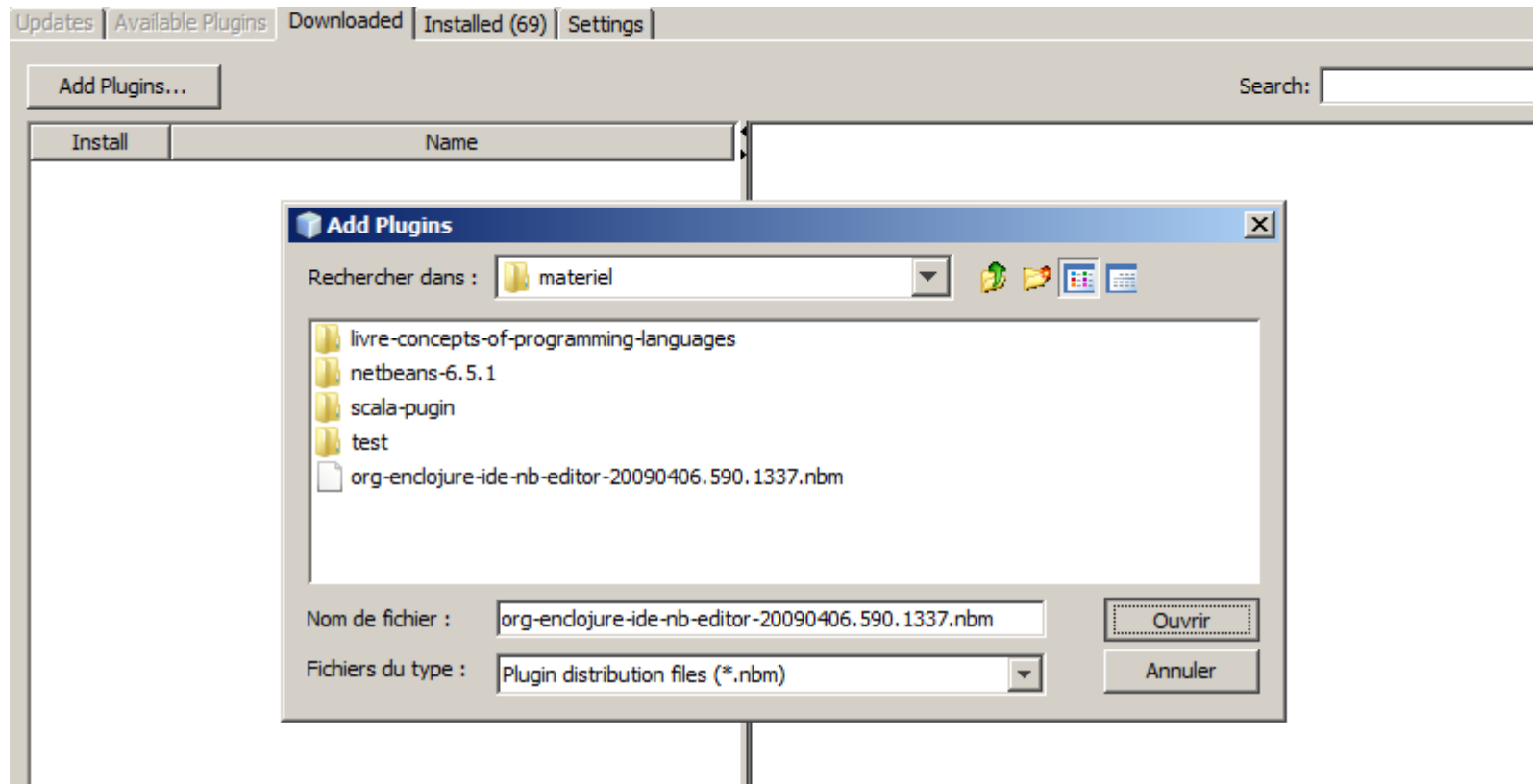
Installation du plugin



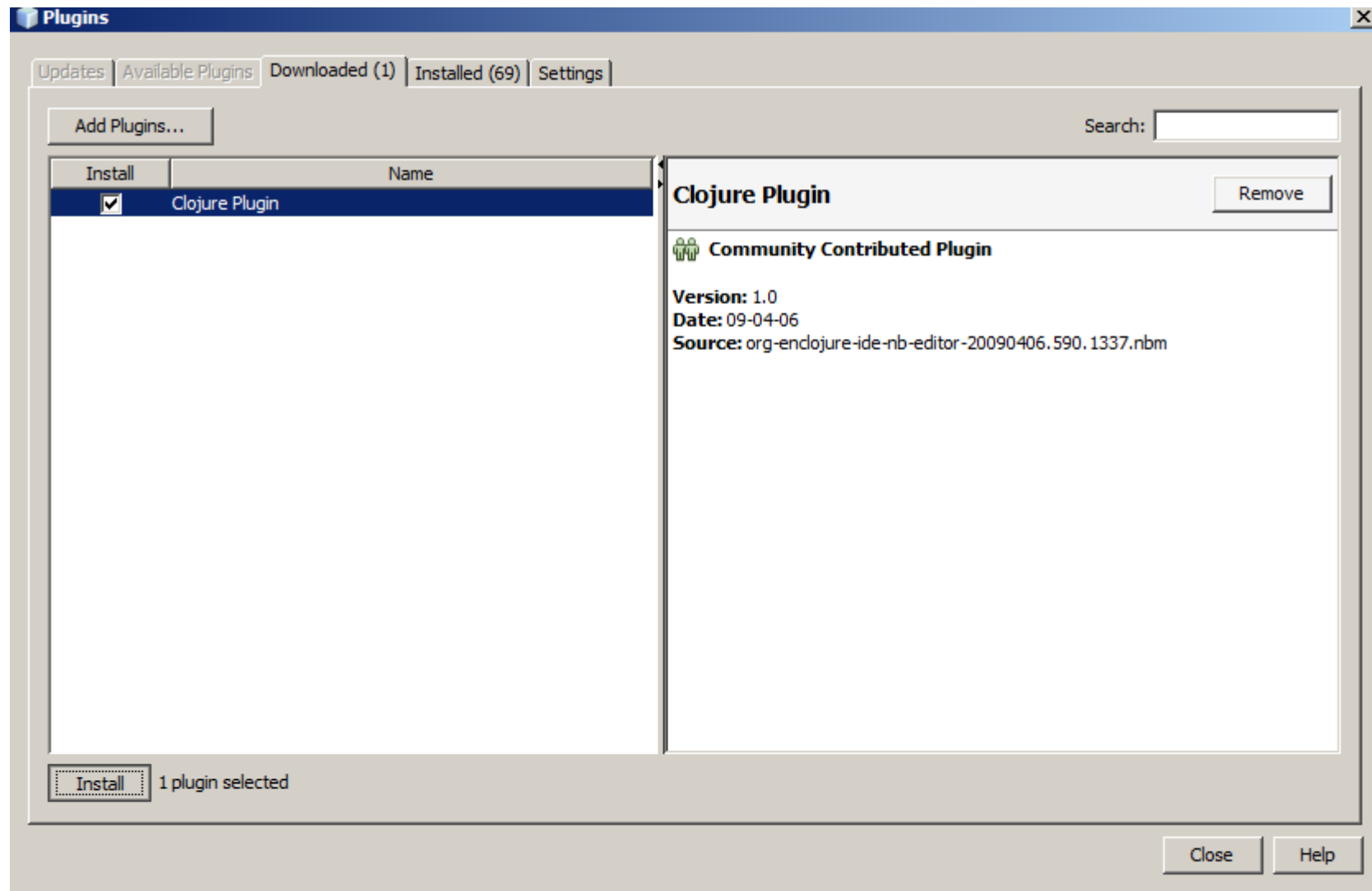
Installation du plugin (suite)



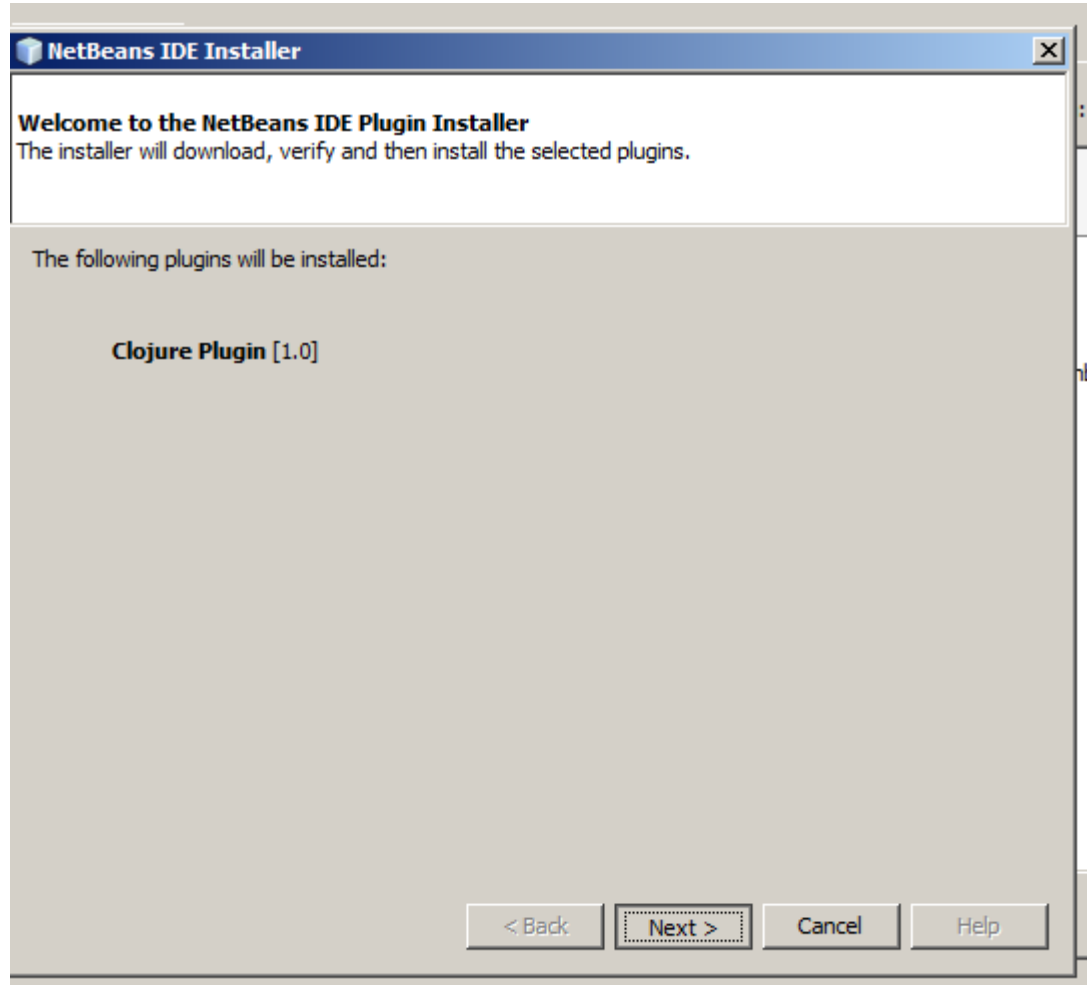
Installation du plugin (suite)



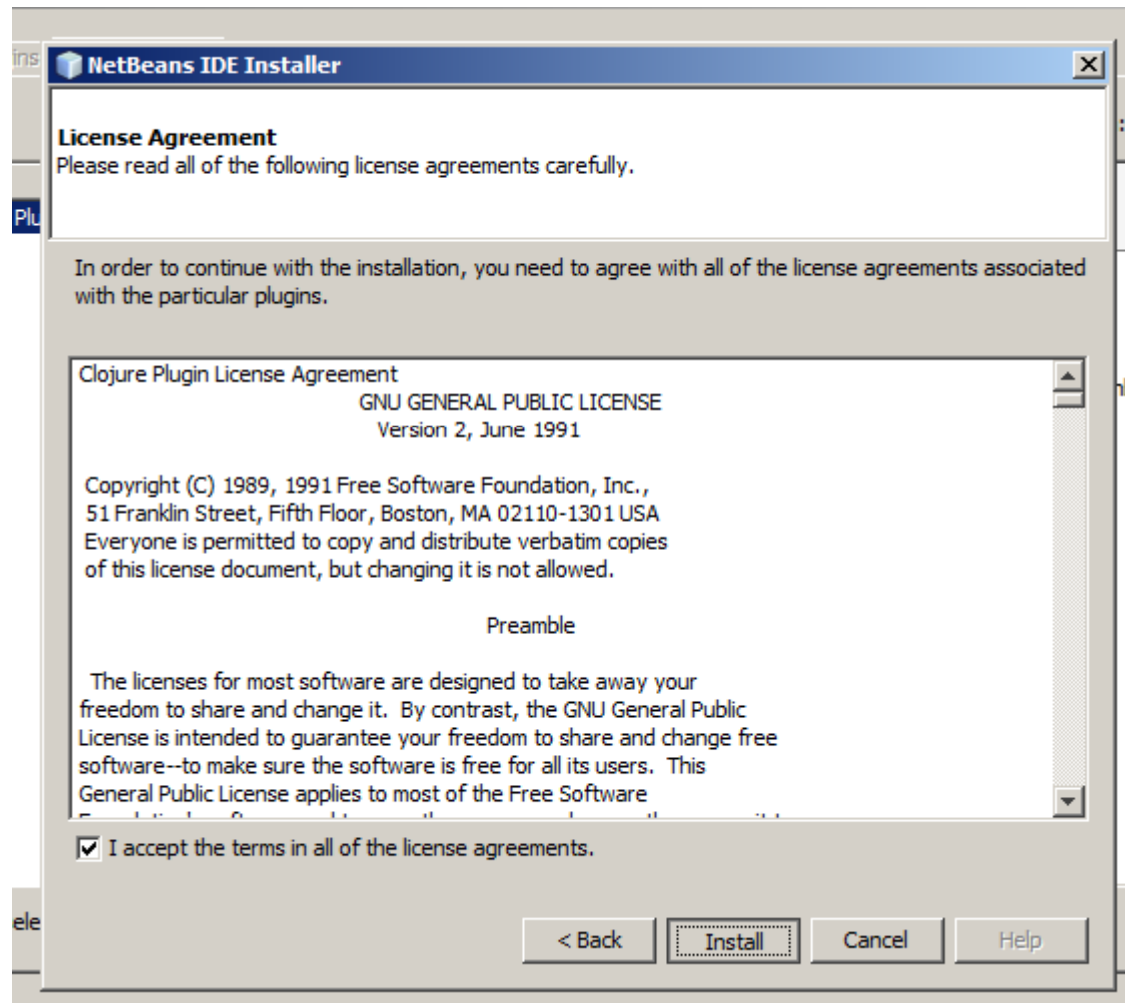
Installation du plugin (suite)



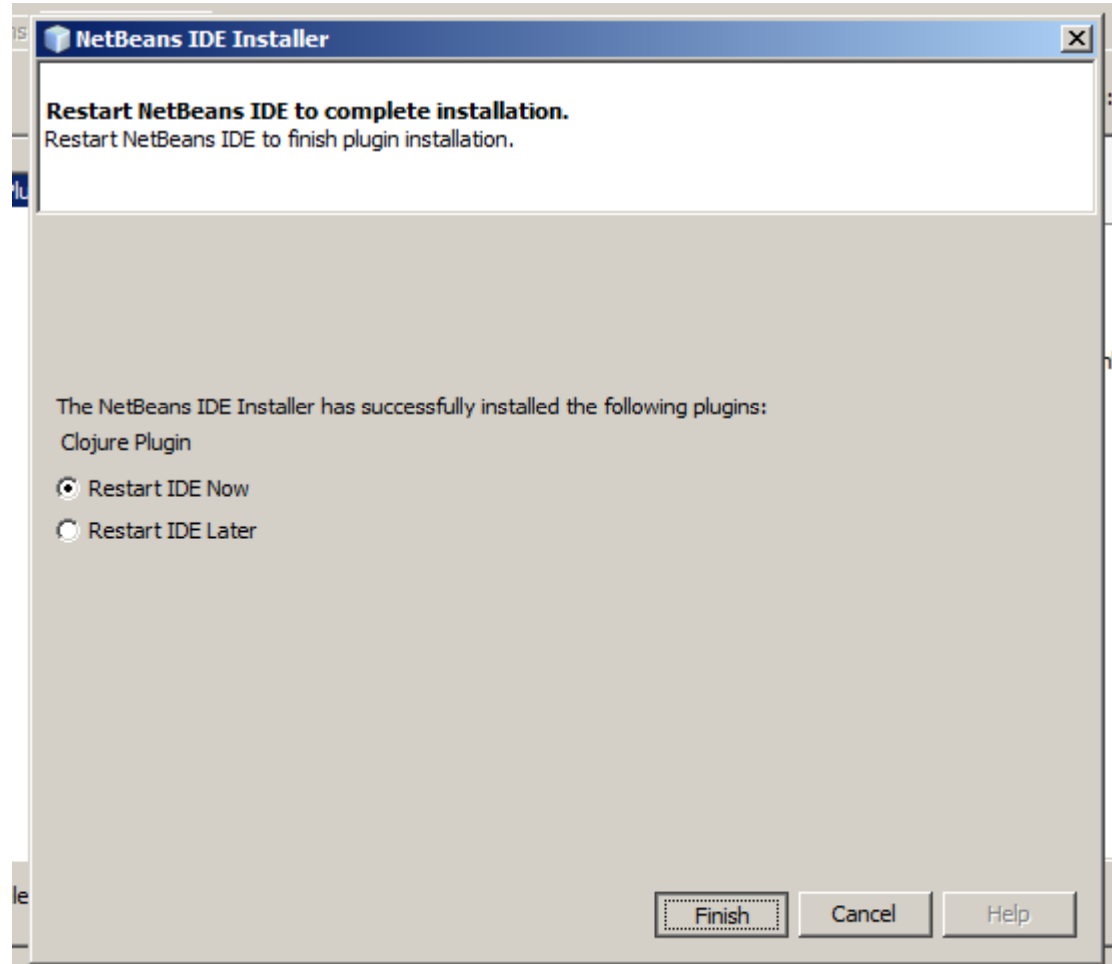
Installation du plugin (suite)



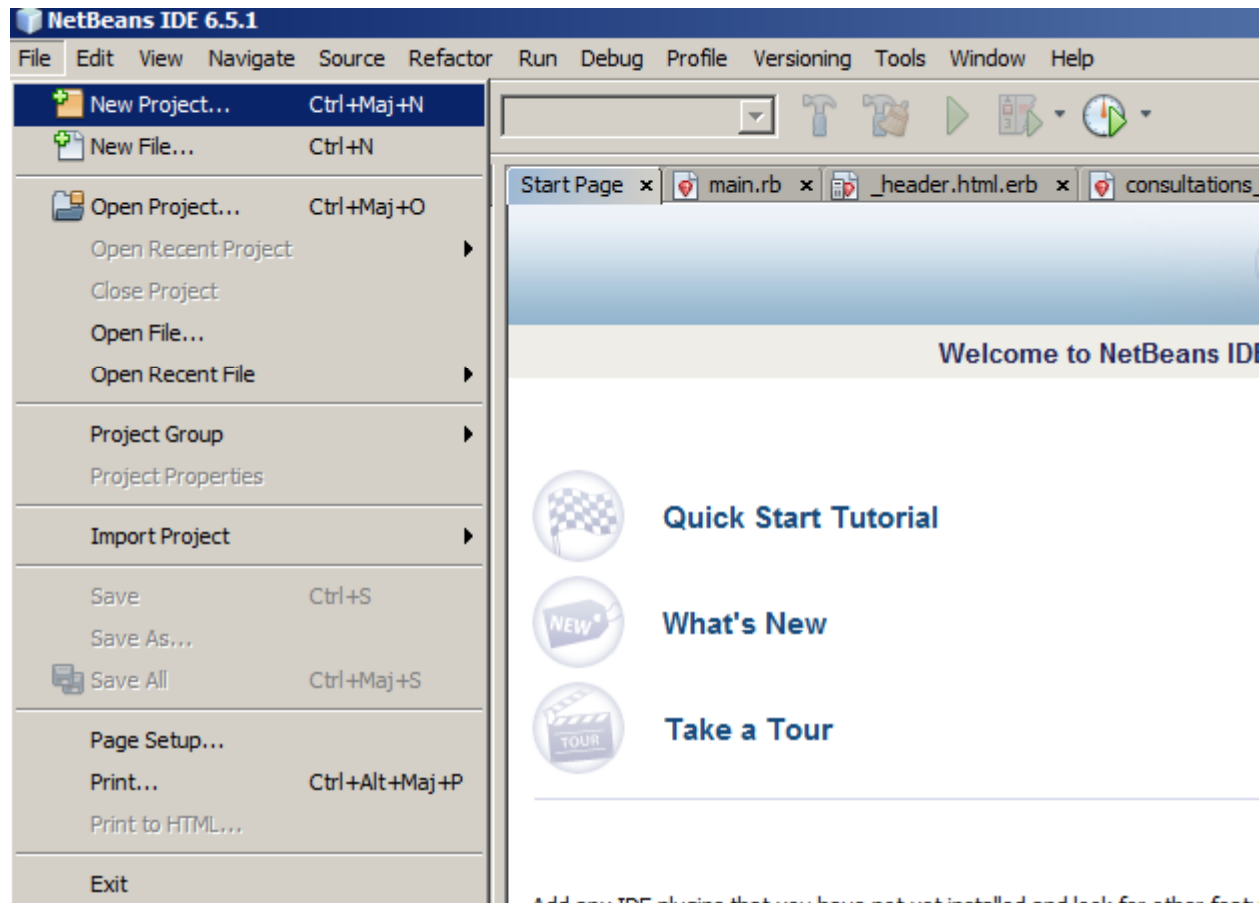
Installation du plugin (suite)



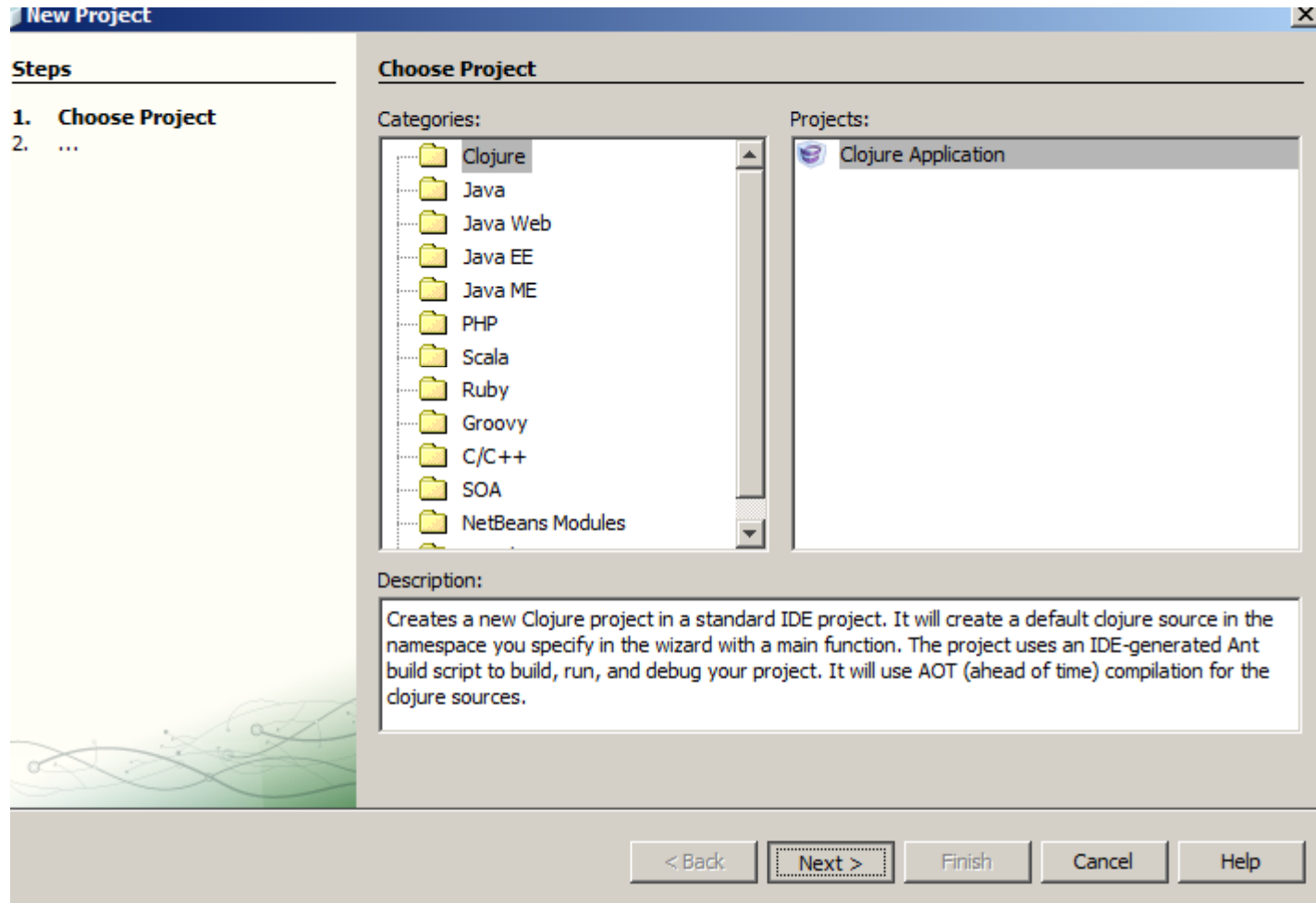
Installation du plugin (suite)



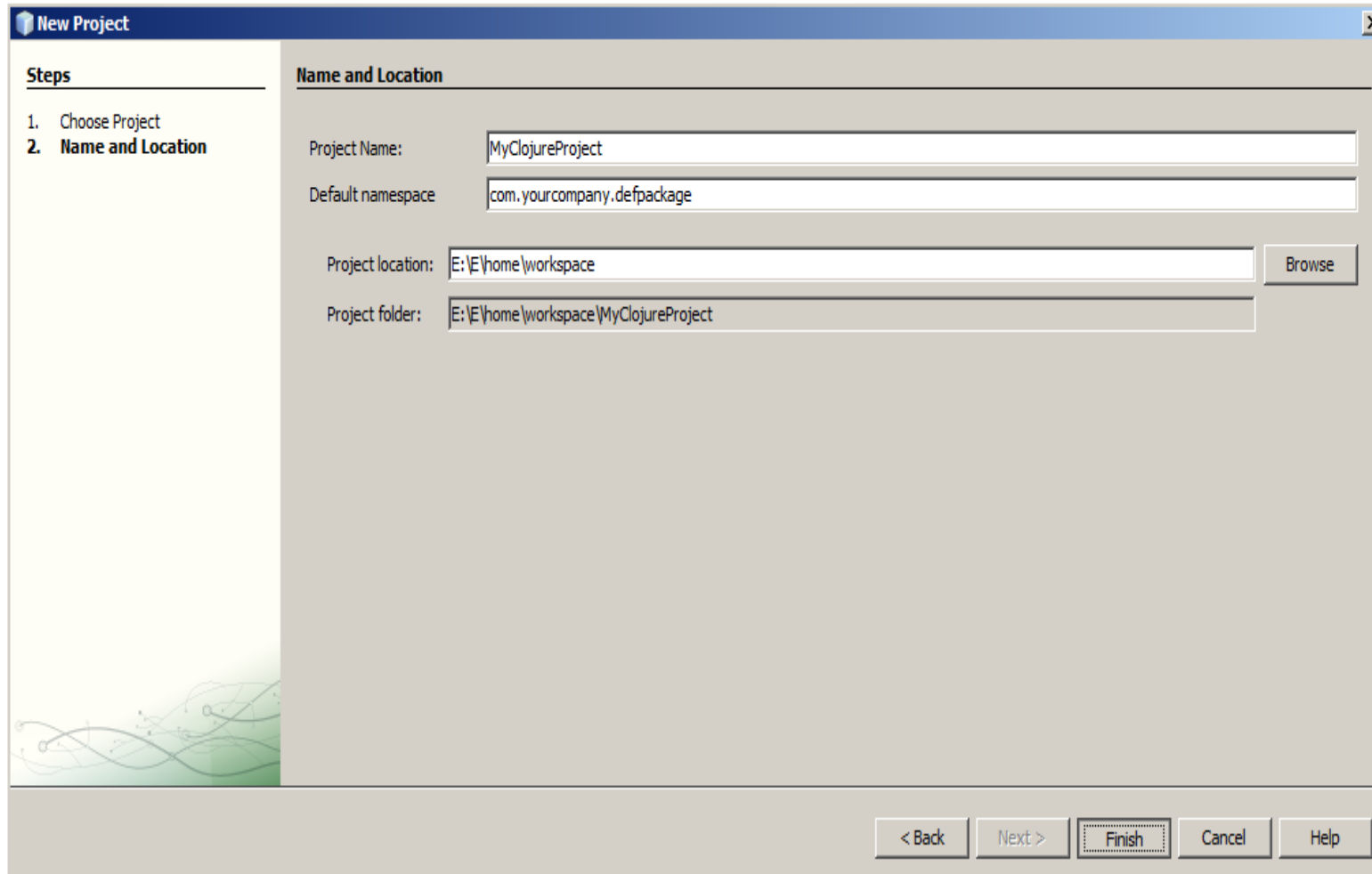
Création d'un nouveau projet Clojure



Création d'un nouveau projet Clojure (suite)



Création d'un nouveau projet Clojure (suite)



The screenshot shows a 'New Project' dialog box with a 'Steps' sidebar on the left and a main 'Name and Location' section. The 'Steps' sidebar lists two steps: '1. Choose Project' and '2. Name and Location', with the second step being the active one. The 'Name and Location' section contains four text input fields: 'Project Name' (filled with 'MyClojureProject'), 'Default namespace' (filled with 'com.yourcompany.defpackage'), 'Project location' (filled with 'E:\E\home\workspace'), and 'Project folder' (filled with 'E:\E\home\workspace\MyClojureProject'). A 'Browse' button is located to the right of the 'Project location' field. At the bottom of the dialog, there are five buttons: '< Back', 'Next >', 'Finish' (which is highlighted with a dashed border), 'Cancel', and 'Help'.

New Project

Steps

1. Choose Project
2. **Name and Location**

Name and Location

Project Name:

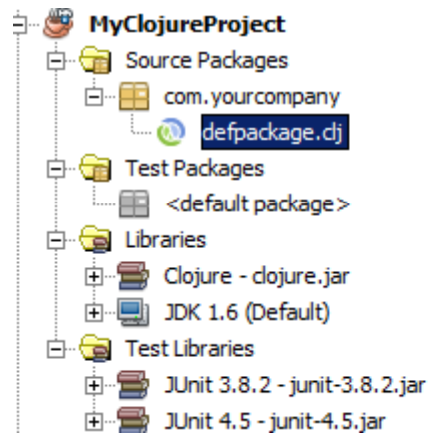
Default namespace:

Project location:

Project folder:

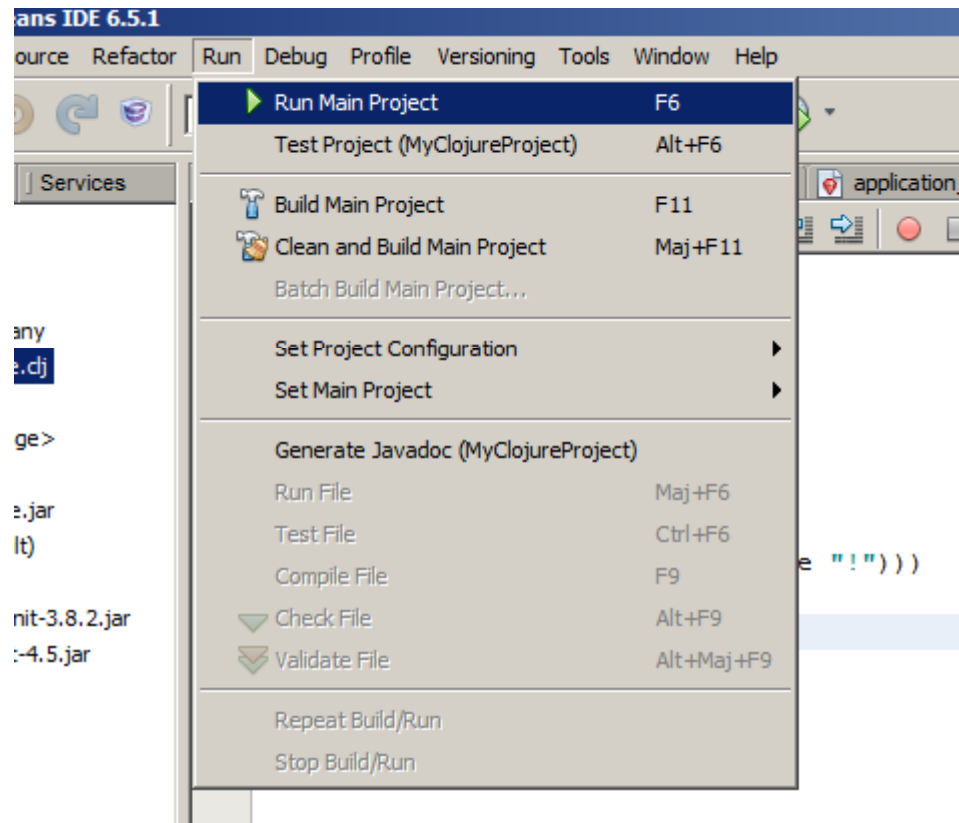
< Back Next > **Finish** Cancel Help

Création d'un nouveau projet Clojure (suite)

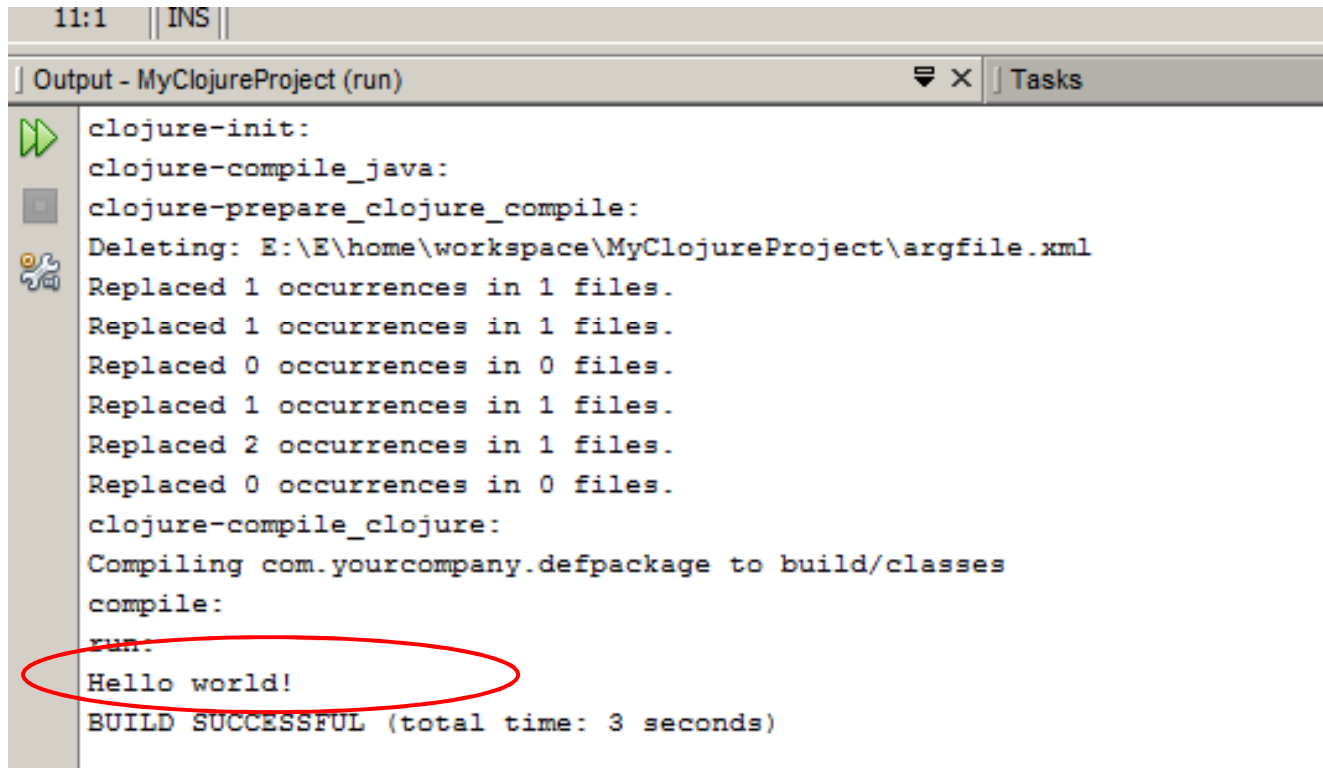


```
1 (comment
2   Sample clojure source file
3 )
4 (ns com.yourcompany.defpackage
5   (:gen-class))
6
7 (defn -main
8   ([greetee]
9    (println (str "Hello " greetee "!"))
10    ([] (-main "world"))
11
```

Création d'un nouveau projet Clojure (suite)



Création d'un nouveau projet Clojure (suite)

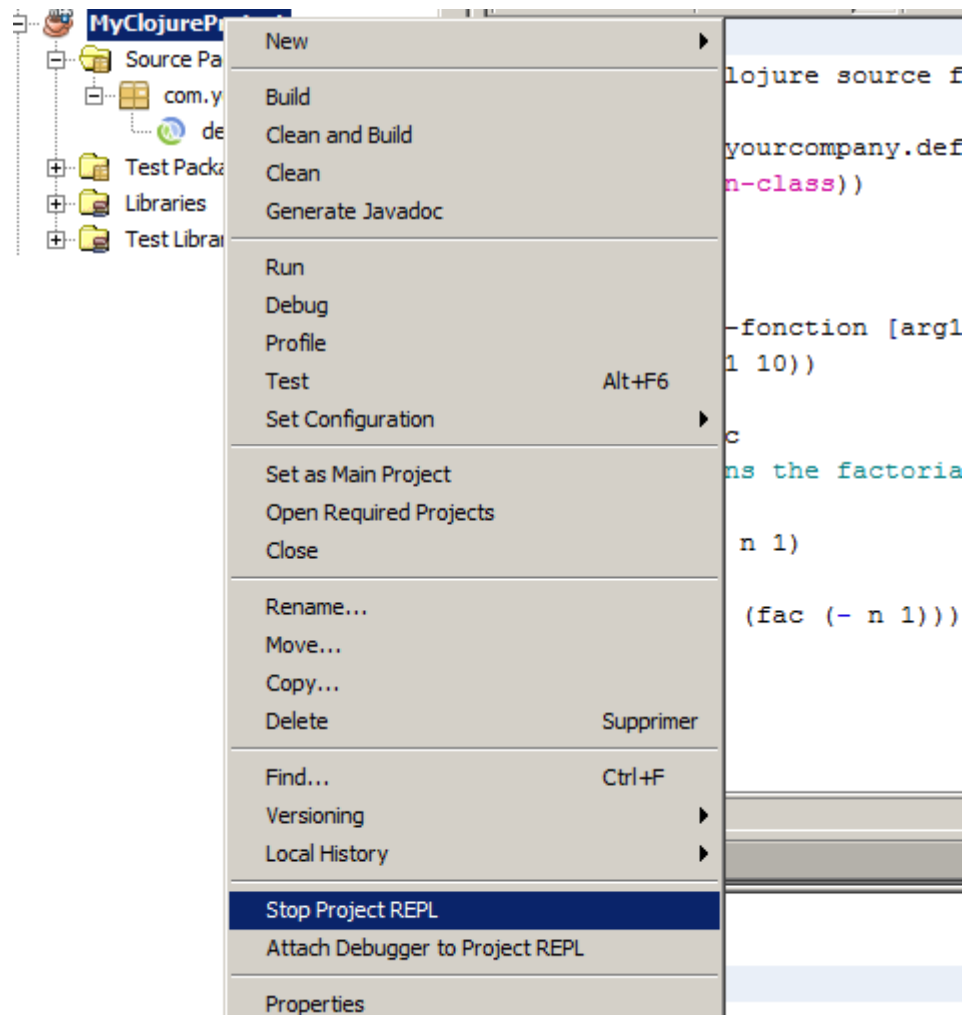


```
11:1 | INS |
Output - MyClojureProject (run)
clojure-init:
clojure-compile_java:
clojure-prepare_clojure_compile:
Deleting: E:\E\home\workspace\MyClojureProject\argfile.xml
Replaced 1 occurrences in 1 files.
Replaced 1 occurrences in 1 files.
Replaced 0 occurrences in 0 files.
Replaced 1 occurrences in 1 files.
Replaced 2 occurrences in 1 files.
Replaced 0 occurrences in 0 files.
clojure-compile_clojure:
Compiling com.yourcompany.defpackage to build/classes
compile:
run:
Hello world!
BUILD SUCCESSFUL (total time: 3 seconds)
```

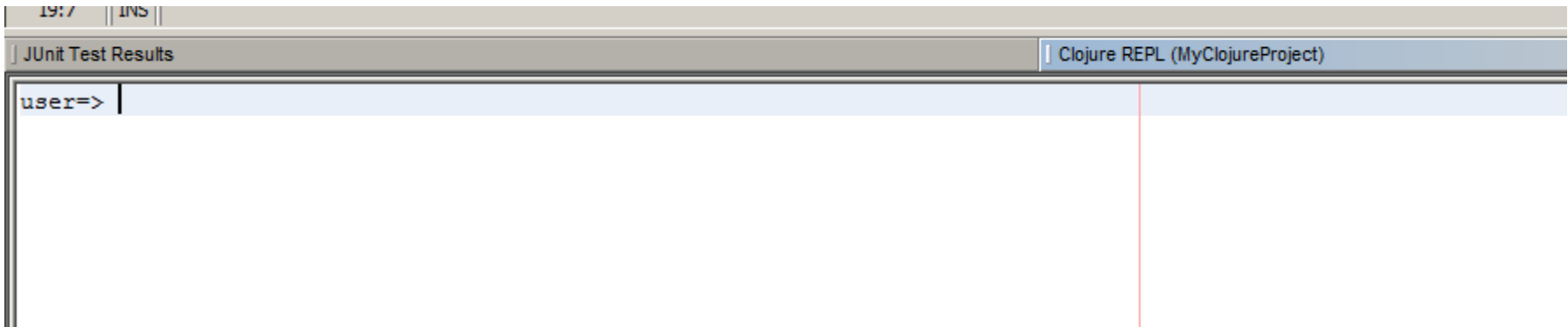
Console Clojure interactive

- ▶ Clojure vient avec une console qui permet d'écrire et d'exécuter de manière interactive du code Clojure
 - ▶ Boucle de lecture, évaluation et affichage (*REPL : read-eval-print loop*)
- ▶ Comment utiliser cette console avec Netbeans ?

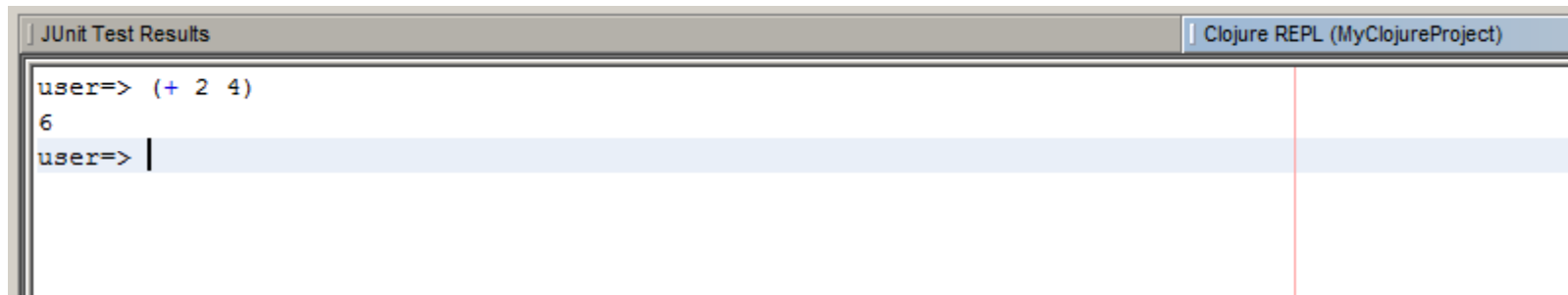
Console Clojure interactive (suite)



Console Clojure interactive (suite)



Console Clojure interactive (suite)

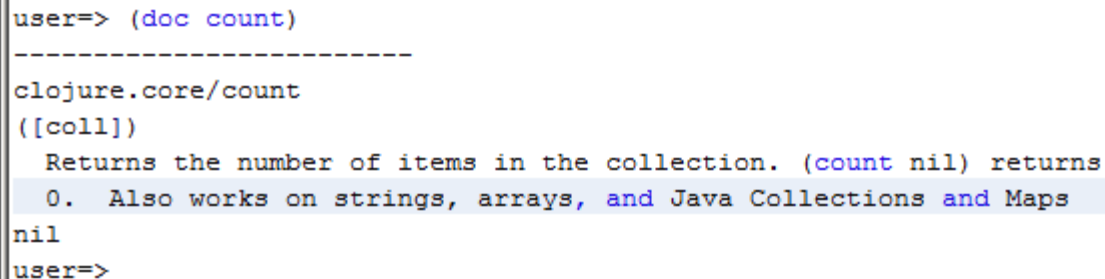


The screenshot shows a Clojure REPL window titled "Clojure REPL (MyClojureProject)". The prompt "user=>" is followed by the expression "(+ 2 4)", which has been evaluated to return the value "6". A new prompt "user=>" is visible on the next line, indicating the REPL is ready for further input. The window also has a tab labeled "JUnit Test Results" on the left.

```
user=> (+ 2 4)
6
user=> |
```

Console Clojure et documentation des fonctions offertes

- ▶ Afin de connaître l'utilité d'une fonction/macro offerte par Clojure, il suffit d'appeler la fonction **doc** suivie du nom de la fonction/macro désirée
- ▶ Exemple :



```
user=> (doc count)
-----
clojure.core/count
([coll])
  Returns the number of items in the collection. (count nil) returns
  0. Also works on strings, arrays, and Java Collections and Maps
nil
user=>
```

Références

- ▶ Livre : *Programming Clojure*. Stuart Halloway 2009.
- ▶ <http://clojure.org>
- ▶ <http://clojure.blip.tv/>
- ▶ <http://jnb.ociweb.com/jnb/jnbMar2009.html>
- ▶ http://en.wikibooks.org/wiki/Clojure_Programming