ING39
Introduction à NetBeans

FIP/Mise à jour Java
2021-2022

Programme

- Initiation à l’environnement NetBeans
- Utilisation du débuggeur
- + tard: tests et JUnit
- + tard: utiliser Git via Netbeans

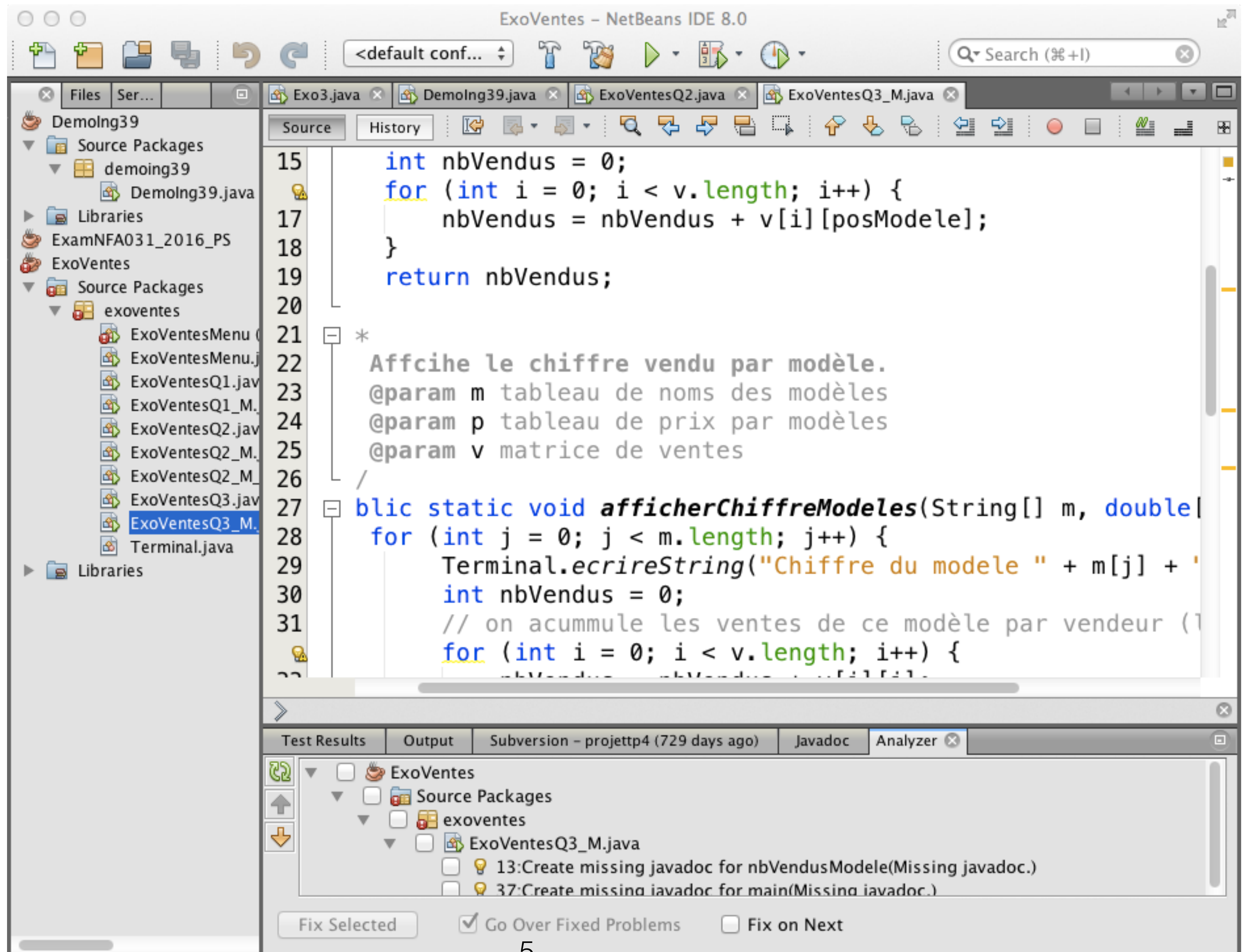
Aujourd’hui

- Mise en train sur Netbeans
  - un instrument précieux pour le développeur : le débuggeur
  - un peu de java quand même : les packages

Initiation à Netbeans

- IDE : « environnement de développement intégré »
- concurrents :eclipse, intellij, vscode
- permet d’effectuer de manière intégrée la plupart des opérations liées à la production de logiciel
    - écriture, compilation
    - test
    - publication…

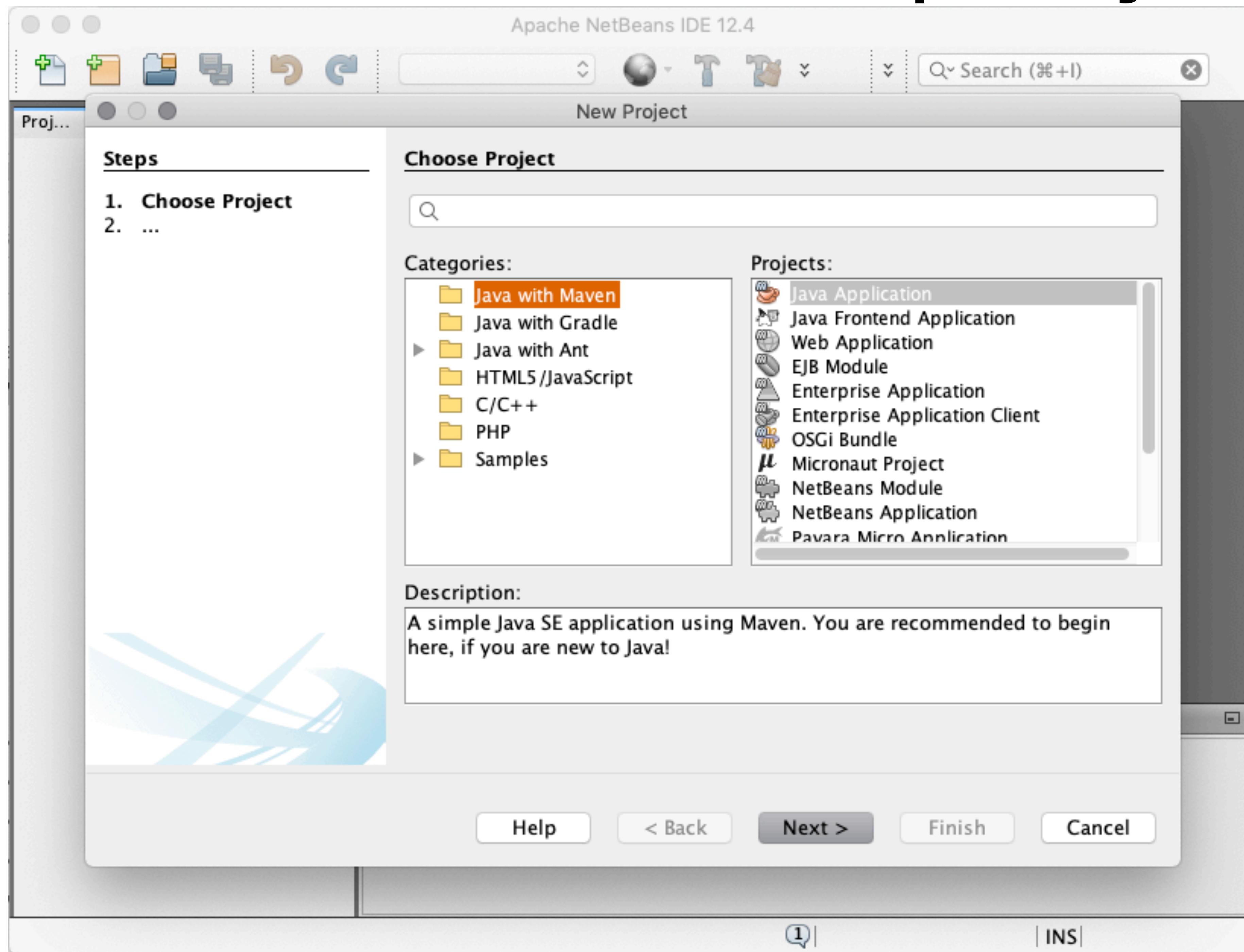
Netbeans



Notion de projet

- Dans netbeans, les fichiers sont créés dans des **projet**
- un « vrai » programme java comporte généralement beaucoup de fichiers « .java » : on les regroupe en projets

Création d'un projet



Apache NetBeans IDE 12.4

Proj...

New Java Application

Steps

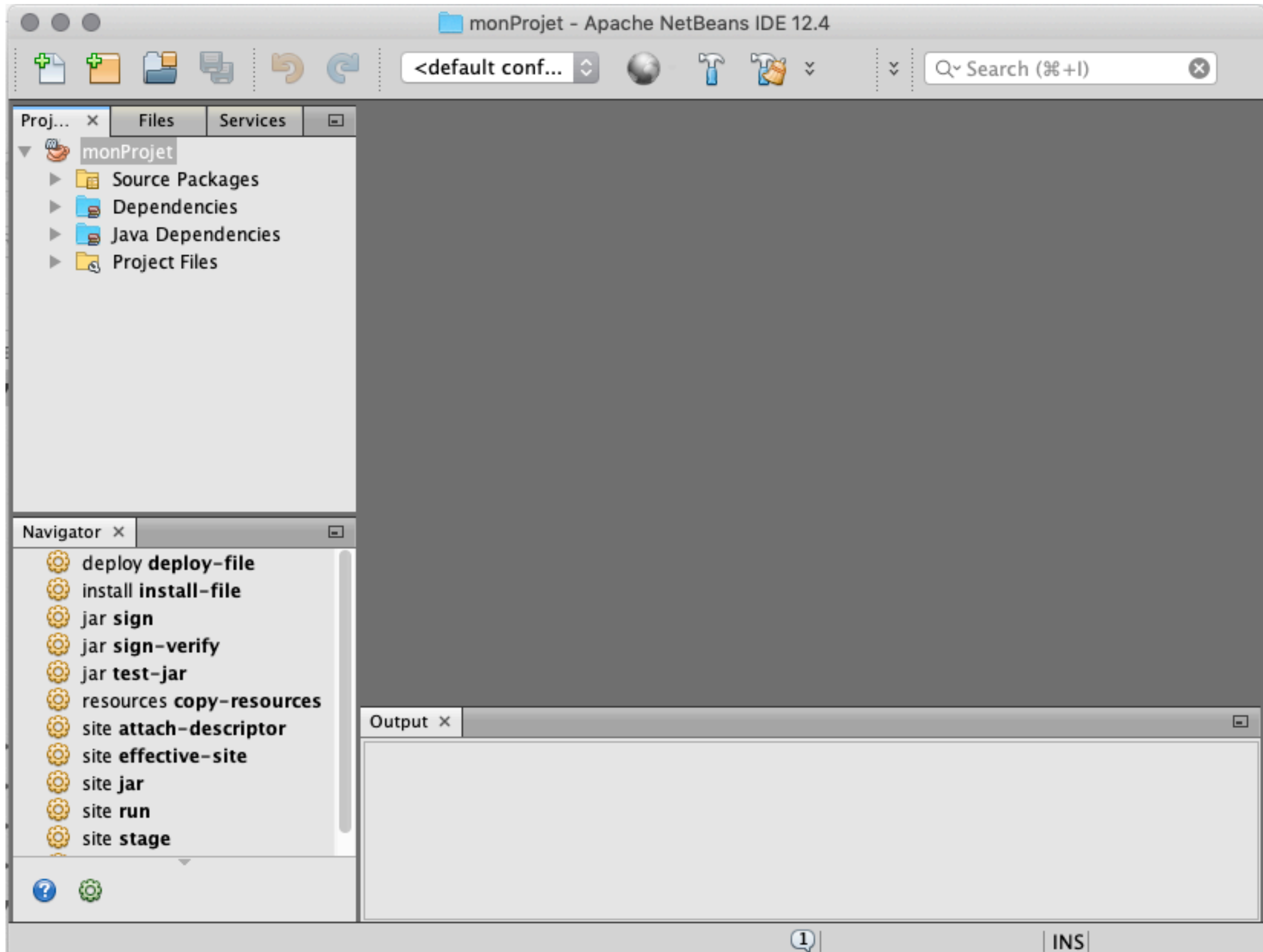
1. Choose Project
2. **Name and Location**

Name and Location

Project Name:	monProjet	
Project Location:	/Users/rosmord/Desktop	Browse...
Project Folder:	/Users/rosmord/Desktop/monProjet	
Artifact Id:	monProjet	
Group Id:	fip	
Version:	1.0-SNAPSHOT	
Package:	fip.monprojet	(Optional)

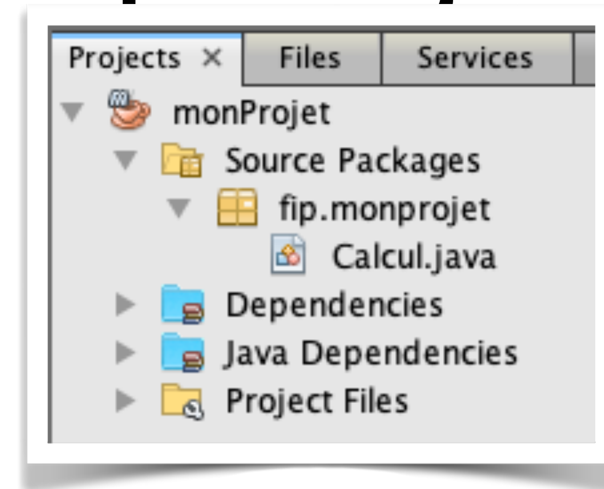
Help | < Back | Next > | **Finish** | Cancel

INS

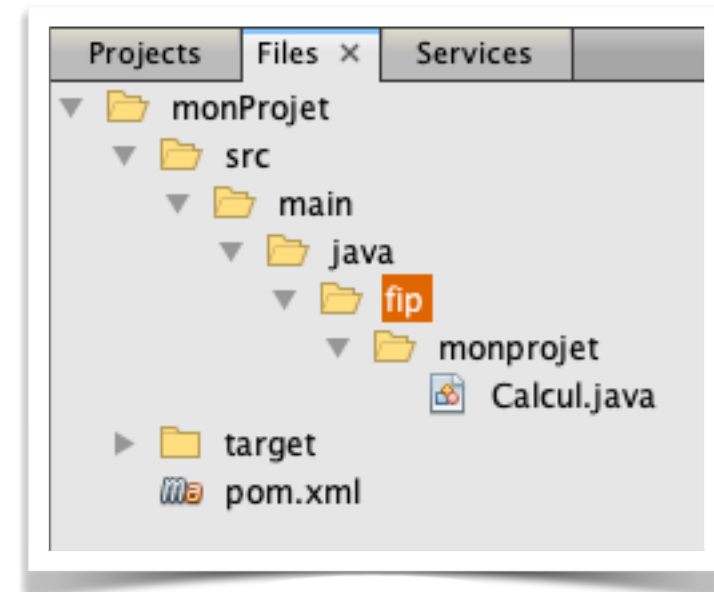

monProjet - Apache NetBeans IDE 12.4
<default conf...
Search (⌘+I)
Proj...
Files
Services
monProjet
Source Packages
Dependencies
Java Dependencies
Project Files
Navigator
deploy deploy-file
install install-file
jar sign
jar sign-verify
jar test-jar
resources copy-resources
site attach-descriptor
site effective-site
site jar
site run
site stage
Output
INS

Structure d'un projet

- contient un fichier de configuration pom.xml
- contient un dossier src où on place les sources
- les « .class » sont automatiquement stockés dans le dossier « target »
- netbeans utilise le logiciel **maven** pour gérer la compilation

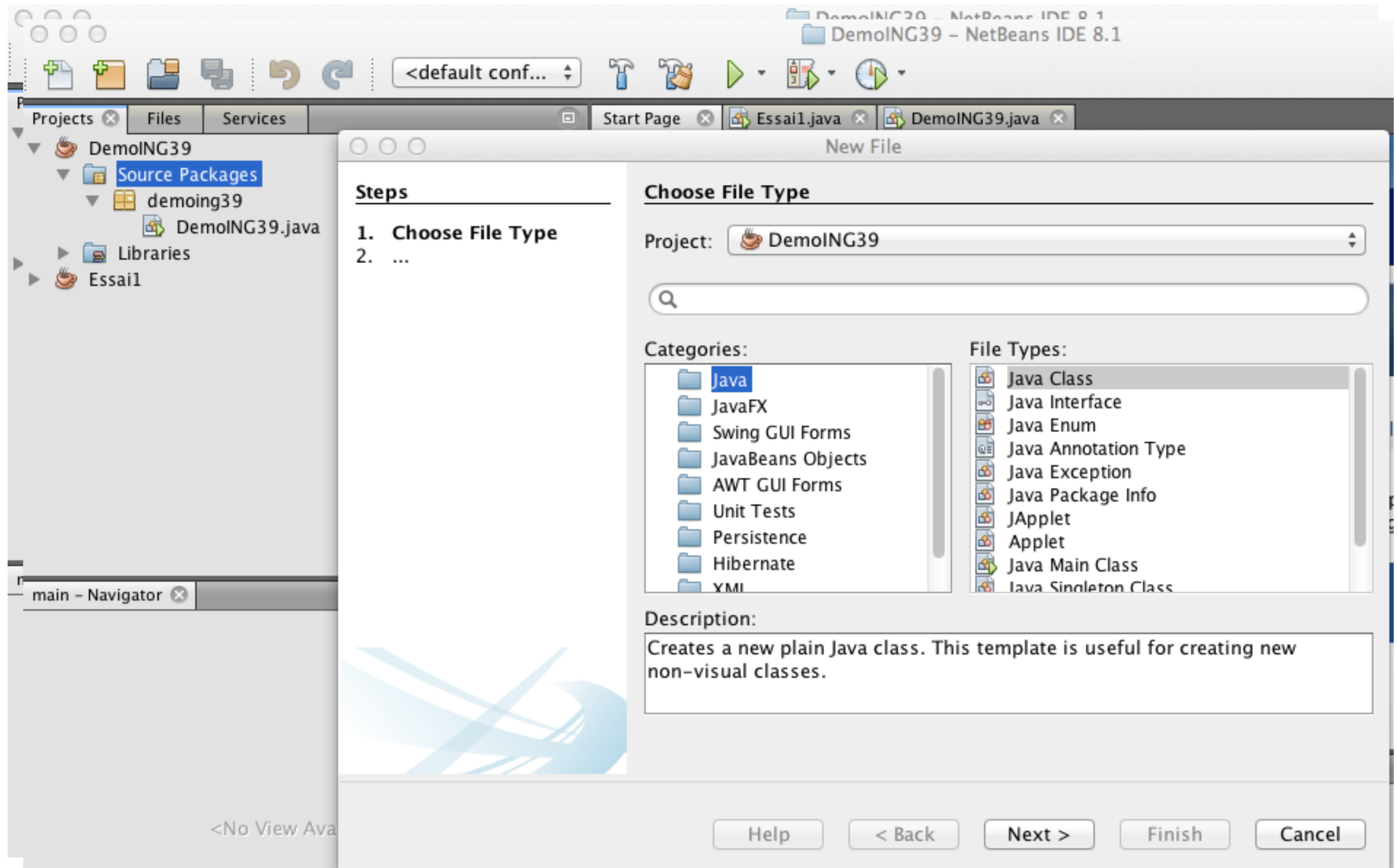


vue logique : onglet « Projects»

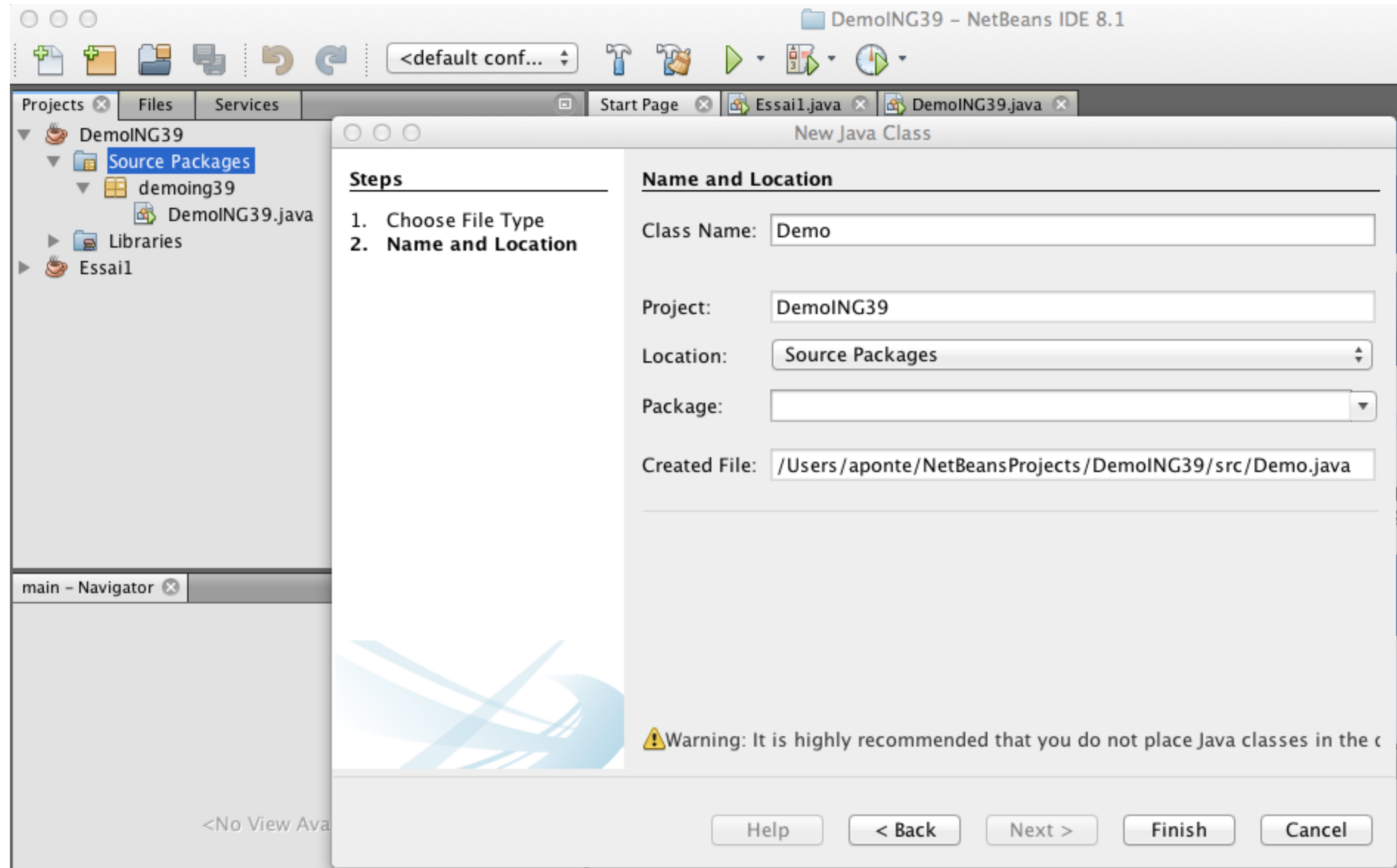


vue réelle: onglet (« Files »)

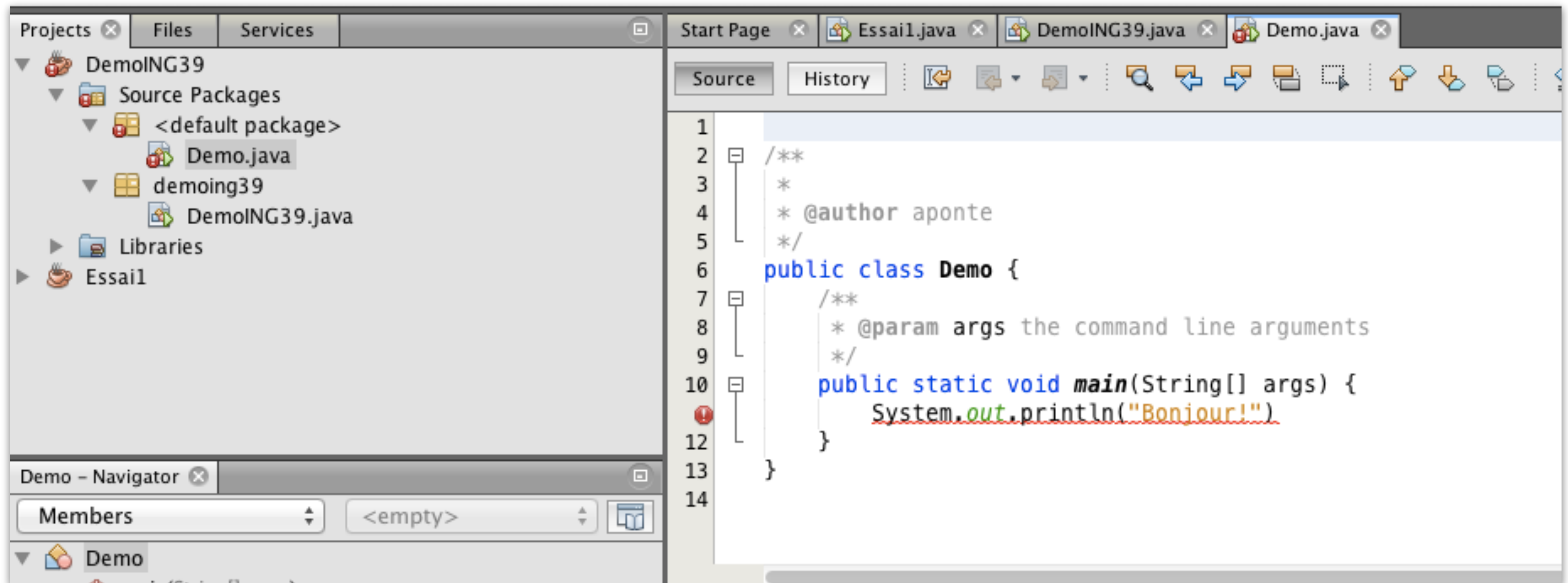
Créer une classe



Créer une classe (suite)



Compilation



- le code est compilé « à la volée »
- les erreurs de syntaxe sont signalées en « temps réel ».

Lancer le programme

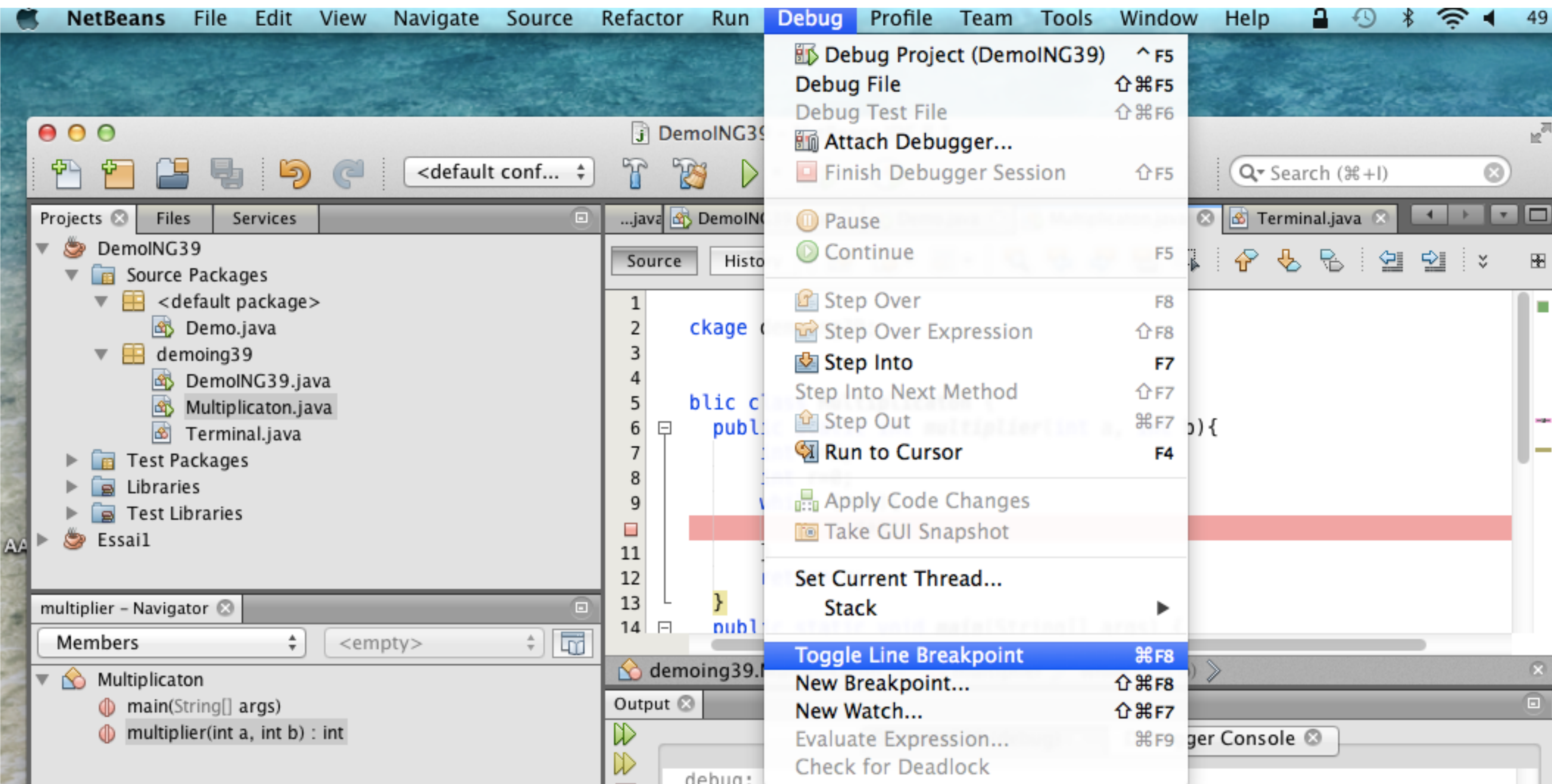
- Sélectionner le fichier qui contient le « main »
- clic droit, plus « Run File »

Le debugger

Sert à visualiser la valeur des variables au cours de l'exécution.

Principes de base:

- on pose des « points d'arrêt » (*breakpoints*);
- quand le programme atteint un point d'arrêt, il est suspendu;
- on peut alors visualiser les variables, exécuter le programme en mode « pas à pas », etc…


NetBeans File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Debug Project (DemoING39) ^F5
Debug File ⇧⌘F5
Debug Test File ⇧⌘F6
Attach Debugger...
Finish Debugger Session ⇧F5
Pause
Continue F5
Step Over F8
Step Over Expression ⇧F8
Step Into F7
Step Into Next Method ⇧F7
Step Out ⌘F7
Run to Cursor F4
Apply Code Changes
Take GUI Snapshot
Set Current Thread...
Stack
Toggle Line Breakpoint ⌘F8
New Breakpoint... ⇧⌘F8
New Watch... ⇧⌘F7
Evaluate Expression... ⌘F9
Check for Deadlock
Projects Files Services
DemoING39
Source Packages
<default package>
Demo.java
demoing39
DemoING39.java
Multiplicaton.java
Terminal.java
Test Packages
Libraries
Test Libraries
Essai1
multiplier – Navigator
Members
<empty>
Multiplicaton
main(String[] args)
multiplier(int a, int b) : int
<default conf...
Search (⌘+I)
Terminal.java
Source
Output

ligne actuelle

Variables et arrêts

sorties

DemoING39 – NetBeans IDE 8.1

Projects | Files | Services | Debugging

'main' at line breakpoint Multiplicaton.java : 10
Multiplicaton.multiplier:10
Multiplicaton.main:19

multiplier – Navigator

Members | <empty>

Multiplicaton
- main(String[] args)
- multiplier(int a, int b) : int

...java | DemoING39.java | Demo.java | Multiplicaton.java | Terminal

Source | History

```
6      public static int multiplier(int a, int b){
7          int n=0;
8          int r=0;
9          while (n<b){
10             r = r+a;
11         }
12         return r;
13     }
14     public static void main(String[] args) {
15         System.out.println("On calcule?");
16         int x= Terminal.lireInt();
17         int y = Terminal.lireInt();
18         System.out.print(x + " x "+y + "=" );
```

demoing39.Multiplicaton > multiplier > while (n < b) >

Variables | Breakpoints | Output

DemoING39 (debug) | Debugger Console

```
debug:
On calcule?
2
3
2 x 3=
```

DemoING39 (debug) running...

DemoING39 – NetBeans IDE 8.1

arrêt, pause, continuer

'main' at line breakpoint Multiplicaton.java : 10

```java
    public static int multiplier(int a, int b){
        int n=0;
        int r=0;
        while (n<b){
            r = r+a;
        }
        return r;
    }
    public static void main(String[] args) {
        System.out.println("On calcule?");
        int x= Terminal.lireInt();
        int y = Terminal.lireInt();
        System.out.print(x + " x "+y + "=" );
```

Name	Type	Value
a	int	2
b	int	3
n	int	0
r	int	0

variables au 1er tour

DemoING39 – NetBeans IDE 8.1

Projects | Files | Services | Debugging

'main' at line breakpoint Multiplicaton.java : 10
Multiplicaton.multiplier:10
Multiplicaton.main:19

multiplier – Navigator

Members | <empty>

Multiplicaton
main(String[] args)
multiplier(int a, int b) : int

...java | DemoING39.java | Demo.java | Multiplicaton.java | Terminal.java

Source | History

```java
6      public static int multiplier(int a, int b){
7          int n=0;
8          int r=0;
9          while (n<b){
10             r = r+a;
11         }
12         return r;
13     }
14     public static void main(String[] args) {
15         System.out.println("On calcule?");
16         int x= Terminal.lireInt();
17         int y = Terminal.lireInt();
18         System.out.print(x + " x "+y + "=" );
```

demoing39.Multiplicaton > multiplier > while (n < b) >

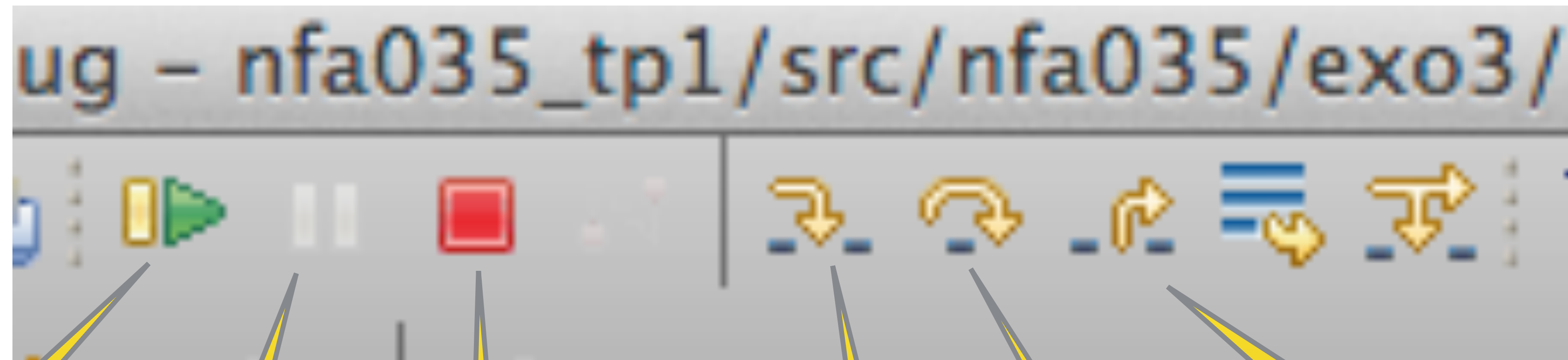
Variables | Breakpoints | Output

Name	Type	Value
<Enter new watch>		
Static		
a	int	2
b	int	3
n	int	0
r	int	8

DemoING39 (debug) running... 10:1 INS

Après quelques tours, n reste non modifié.

Débugger : barre de commande



continuer l'exécution

pause

Tuer le programme

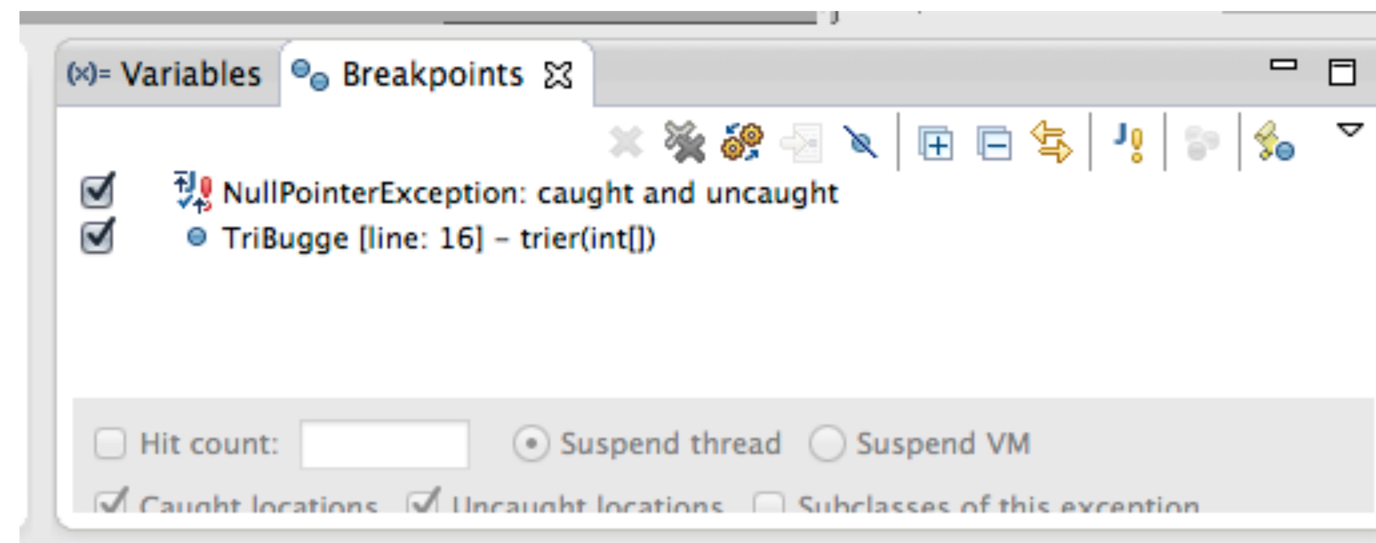
exécuter en « entrant » dans les fonctions

ligne à ligne

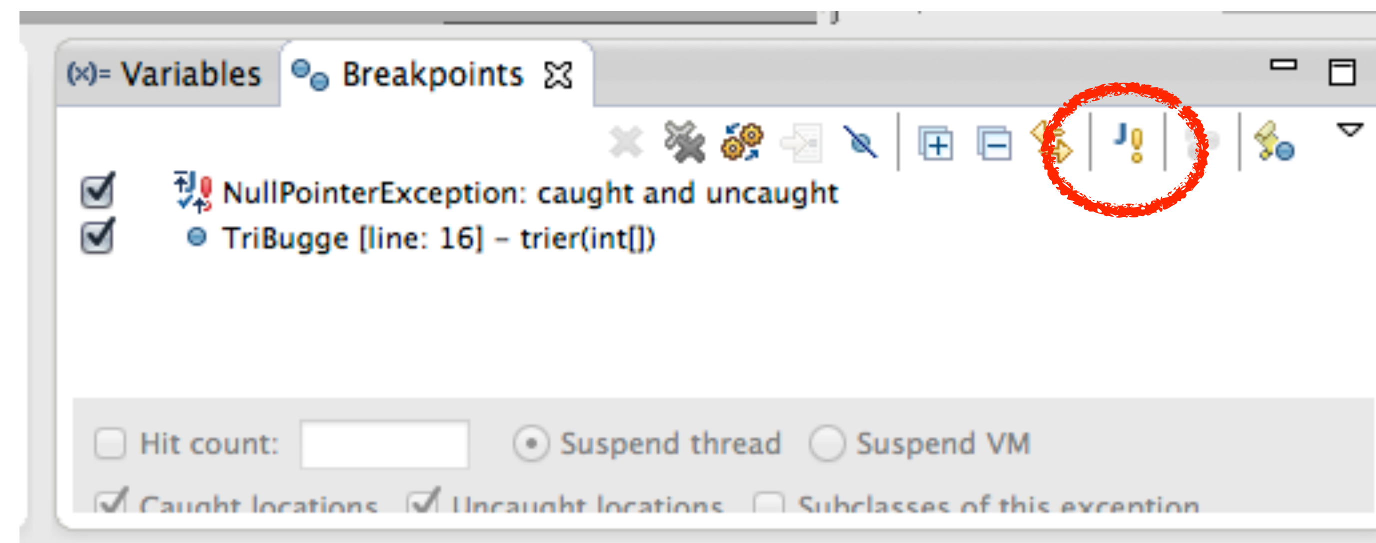
termine la méthode actuelle

Debugger : gestion des points d'arrêt

- On peut supprimer ou désactiver les points d'arrêt



- On peut poser des points d'arrêt sur des exceptions (très utile!)



Les packages

- Problème : un « vrai » programme utilise beaucoup de classes
- il utilise aussi souvent plusieurs bibliothèques téléchargées sur le web, qui contiennent elles-même des classes
- risque (certitude !!) de *collisions* dans les noms : on aura plusieurs classes avec le même nom!

Exemple

- Cinq classes qui s’appellent « Element »
- Deux qui s’appelent List…
- Comment les distinguer ?

Java™ Platform
Standard Ed. 7

All Classes

Packages

java.applet

ECGenParameterSpec
ECKey
ECParameterSpec
ECPoint
ECPrivateKey
ECPrivateKeySpec
ECPublicKey
ECPublicKeySpec
EditorKit
Element
Element
Element
Element
Element
ElementFilter
ElementIterator
ElementKind
ElementKindVisitor6
ElementKindVisitor7
Elements
ElementScanner6

Overview Package Class Use Tree

Prev Next Frames No Frames

Java™ Platform, Stan
API Specification

This document is the API specification for

See: Description

Packages

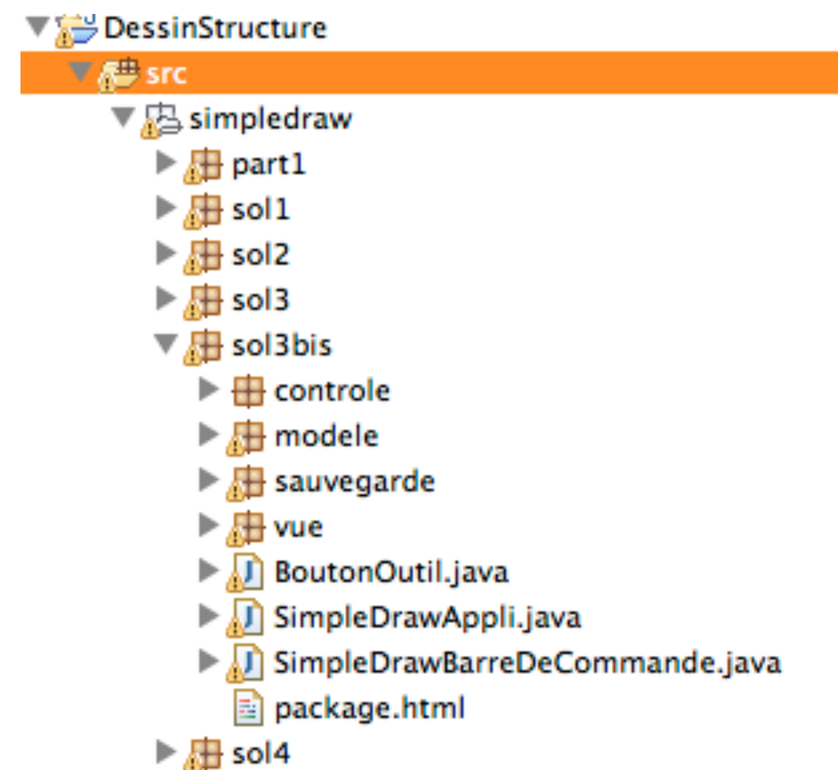
Package
java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event
java.awt.font
java.awt.geom

Solution : les package

- On regroupe les classes de manière thématique dans des *packages*
- par convention, le nom d'un package commence par une **minuscule**
- C'est très proche des *dossiers* que vous utilisez pour regrouper les fichiers
- Un package peut contenir des classes ou d'autres packages
- Exemples:
    - java.text : package contenant les classes pour manipuler du texte
    - javax.swing : package de base pour les classes d'interface utilisateur
    - javax.swing.border : package contenant toutes les classes représentant des « bords » de fenêtre

Exemple

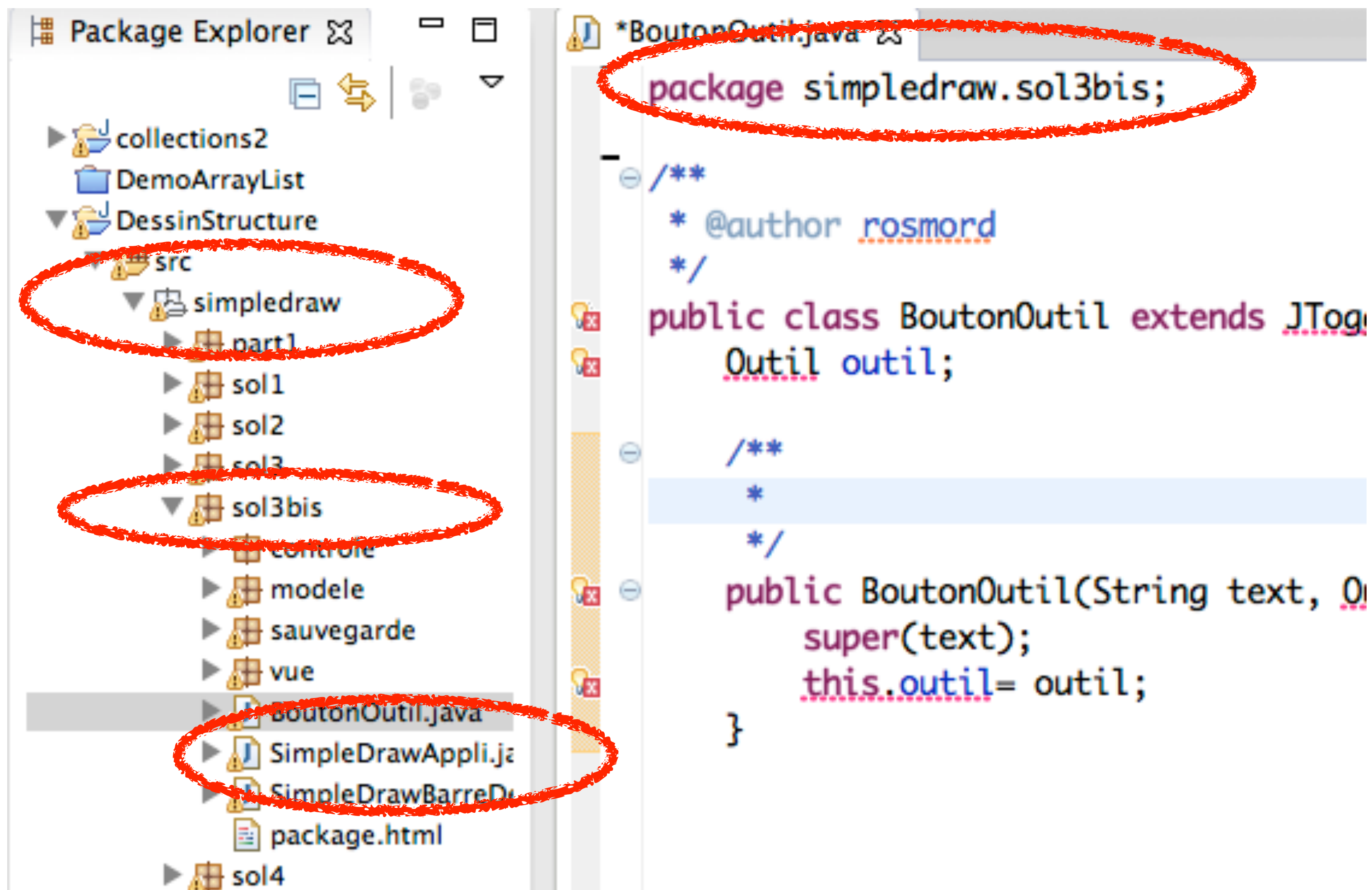
- Logiciel de dessin:
  - un package pour le « modele » (la représentation du dessin en mémoire)
  - un package pour la « vue » (son affichage)
  - un package pour le code de sauvegarde du dessin sur fichier



Modes d'organisation

- par thème : dans une application de gestion de notes, un package pour la gestion des étudiants, un autre pour ce qui concerne les matières…
- par couche : un package pour l'interface utilisateur, un package pour la logique du programme, un package pour l'accès aux données

Création de packages



- On crée un dossier par package en respectant la hiérarchie
- Dans chaque classe, on déclare son package
- Chic, eclipse le fait tout seul !

Utilisation d'une classe dans un autre package que le sien

- On peut toujours donner à une classe son nom complet:

public static int somme(**java.util.List** maListe) {…}

- mais c'est pénible
- autre solution : import.

import

- entre la ligne qui déclare le package et le début de la classe, on peut *importer* des classes

- c'est purement syntaxique: ici signifie que dans la classe BoutonOutil, « Outil » désigne la classe simpledraw.sol3bis.controle.Outil

```java
package simpledraw.sol3bis;

import javax.swing.JToggleButton;

import simpledraw.sol3bis.controle.Outil;

/**
 * @author rosmord
 */
public class BoutonOutil extends JToggleButton {
    Outil outil;
```

import

- On peut utiliser le caractère « * » pour importer toutes les classes d'un package (mais pas celles des sous packages)
- ex: import java.util.*; // import List, Set….

Le package par défaut

- Une classe qui n’a pas explicitement de package est dans le package par défaut.
- Il n’a pas de nom, le pauvre
- C’est celui que vous avez utilisé jusqu’à présent…
- mais c’est fini: on ne peut pas importer une classe qui est dans le package par défaut
- du coup, il faut éviter de l’utiliser

Noms « réels » des classes

- le nom complet d'une classe est le nom de la classe, précédé de celui du package qui la contient
- un sous package a comme nom le nom de son parent, suivi d'un « . », suivi du nom du package
- Exemples
  - classe **java.awt.List** : représente une liste dans une interface graphique awt.
    - package java.awt, dans le package « java » (pour les bibliothèques standards)
  - « classe » **java.util.List** : liste d'éléments en mémoire

public, private et rien

- public : la classe et la méthode est visible par tout le monde

- private : une méthode ou un champ private n’est visible que depuis la classe où il est défini

- protected (on en parle plus tard)

- (rien) : quand une méthode n’est ni publique ni private, elle est « publique dans son package, private ailleurs »