Cours d'Environnement de Développement

Arnaud Sangnier

Partie 1

Quelques informations

- **Responsable du cours :** Arnaud Sangnier (sangnier@liafa.jussieu.fr)
- **Responsables des TP :**
  - Arnaud Sangnier
    Mardi 14h30 - 16h30 — Salle 432 C
  - Cezara Dragoi (Cezara.Dragoi@liafa.jussieu.fr)
    Lundi 8h30 - 10h30 — Salle 548C
- **Page web :**

  `http://www.liafa.jussieu.fr/~sangnier/enseignement/developpement.html`
- **Première séance de cours :** 15 Mars 2011
- **Dernier cours :** 3 Mai 2011
- **Premier TP :** Semaine du 21 Mars 2011
- **Dernier TP :** Semaine du 2 Mai 2011

À propos du cours

Prérequis

- Savoir programmer en Java
- Connaître les concepts de Programmation Orientée Objets

Notions que nous verrons

- Utilisation d'Eclipse (utilisation de base, débogueur, génération de tests)
- Implémentation de plug-ins pour Eclipse
- Notions d'UML
- Notions sur les patrons de conception

Évaluation

Première session

- Projet Java (énoncé et explications donnés le 5 Avril 2011)
- Moitié des points sur l'implémentation et moitié des points sur rapport + présentation oral de 20 minutes
- Idée du projet : Implémentation d'un tetris

Deuxième session

- Examen oral sur machine

EDI

Qu'est-ce-qu'un EDI ?

Définition

Un *Environnement de Développement Intégré* (EDI) (IDE en anglais) est un logiciel regroupant un ensemble d'outils utilisés pour le développement d'applications.

Exemple d'outils inclus dans un EDI :

- Éditeur de texte spécialisé
- Compilateur
- Débogueur
- Outils automatiques de gestion d'applications ayant plusieurs fichiers sources (projets)
- Gestionnaire de version et de sauvegarde (CVS)
- Générateur de documentation

Un peu d'histoire

Préhistoire :

- 1950-60 : Cartes perforées
- 1960-70 : Terminaux, éditeurs de texte basiques, compilateurs et débogueurs en ligne de commande
- 1970-1980 : Introduction de **makefile** et de fichiers de configurations permettant de contrôler convenablement la compilation

Avec le développement des SE ayant une interface graphique (1980-90)m les premiers EDI apparaissent (1981 Turbo Pascal)

Quelques dates :

- 1983 : Borland Turbo Pascal (DOS)
- 1987 : Borland Turbo C
- 1991 : Microsoft Visual Basic 1
- 1997 : Microsoft Visual Studio

Quelques exemples

Logiciels libres :

- Emacs, XEmacs : basique mais adaptables à tout langage
- OpenOffice.org : langages de script
- Kdevelop (KDE) : C, C++, basé sur les outils GNU
- Netbeans (Sun) : initialement conçu pour Java, maintenant C, C++, XML et HTML
- Eclipse (OTI-IBM) : Java, C/C++, PHP, HTML, etc.

Logiciels propriétaires :

- Visual Studio (Microsoft) : C/C++, .NET, C#, etc.
- JBuilder (Borland) : Java
- JCreator : Java
- WinDev (PC Soft) : application PC Pocket et mobile

ECLIPSE

Introduction

Une plateforme ouverte pour le développement

Conçu sur la base d'un EDI Java, **Eclipse** devient un EDI pour développer des EDIs et d'autres outils

Caractéristiques principales :

- Non dédié à un langage ou SE ou IHM
- Facile à comprendre mais aussi faile à étendre
- Paramétrisable selon les besoins/goûts du programmeur
- Capable d'automatiser les tâches lourdes du développement
- Utilisable pour son propre développement (*bootstrap*-able)

Ressources

1996 IBM achète OTI qui développe la suite d'EDI Visual Age (en SmallTalk) et en particulier VA4J

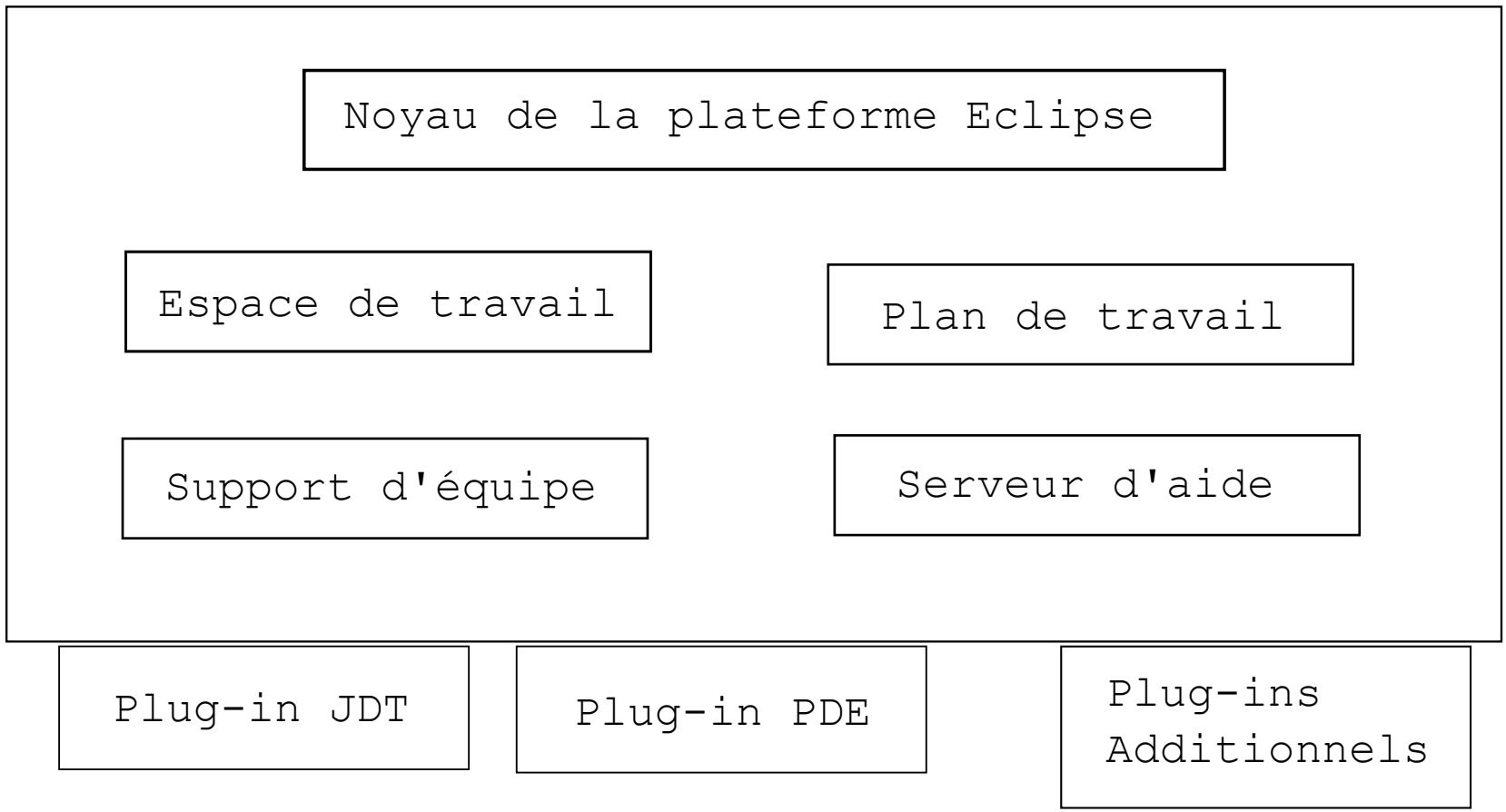
2001 Après un investissement de 40M$, IBM lance Eclipse 1, grand succès populaire car plateforme ouverte et gratuite. Le consortium Eclipse est créé (IBM, Borlanad, RedHat, SuSE, Intel, ...)

2010 Eclipse Helios 3.6

Documentation :

- `http://www.eclipse.org` (Téléchargement, cours, ...)
- Steve Holzner, Eclipse, O'Reilly 2004

Architecture d'Eclipse

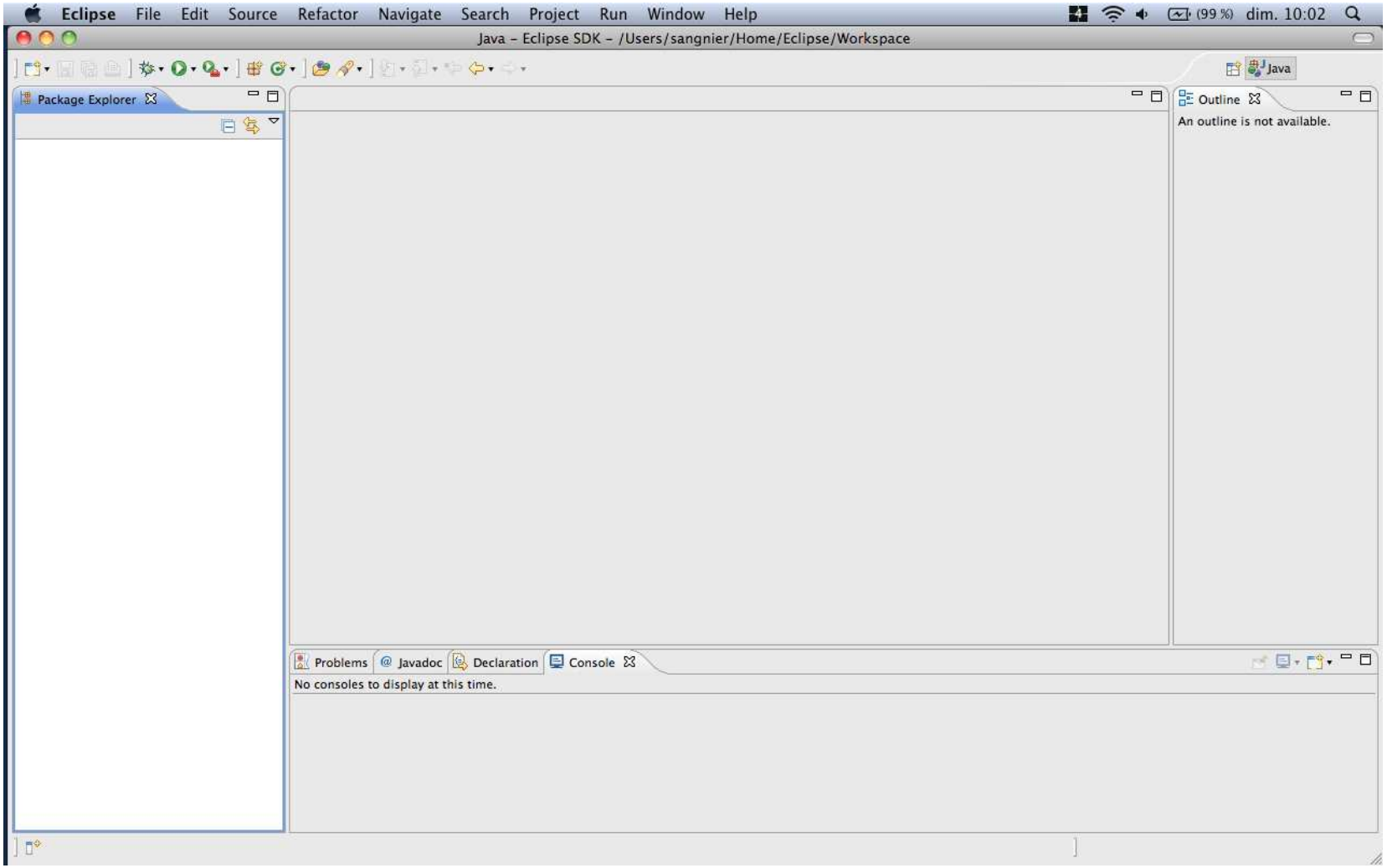


- *JDT : Java Development Toolkit*
- *PDE : Plug-in Development Environment*

Noyau de la plateforme

- Lance les outils logiciels constituant Eclipse
- Charge les plug-ins
- Premier composant à être exécuté au lancement d'Eclipse

Plan de travail



Plan de travail

- Interface graphique proposée par Eclipse aux utilisateurs
- A l'apparence d'une application native du système
- Construit sur la base du SWT (*Standard Widget Toolkit*) (fait appel aux bibliothèques logicielles natives du système d'exploitation)
- Contrairement a Java, Eclipse n'est pas indépendant du système d'exploitation à cause de cela

Espace de travail

- Gère les ressources nécessaires au travail du développeur
- Tout code développé sous Eclipse fait partie d'un Projet
- Chaque projet est dans un dossier propre dans le répertoire de travail d'Eclipse
- Le dossier du projet comporte lui-même un certain nombre de sous-dossiers

Support d'équipe

- Plug-in en charge du contrôle de version du code
- Code archivé ou extrait d'une archive
- Pas d'écrasement ou de perte des modifications réalisées par les autres membres de l'équipe
- Identique à un client CVS (*Concurrent Version System*)
- Contrôle efficace des différentes versions d'un code

Serveur d'aide

- Système de documentation extensible destiné à fournir une aide
- Les plug-ins peuvent fournir une documentation HTML au format XML qui indique comment naviguer dans l'aide

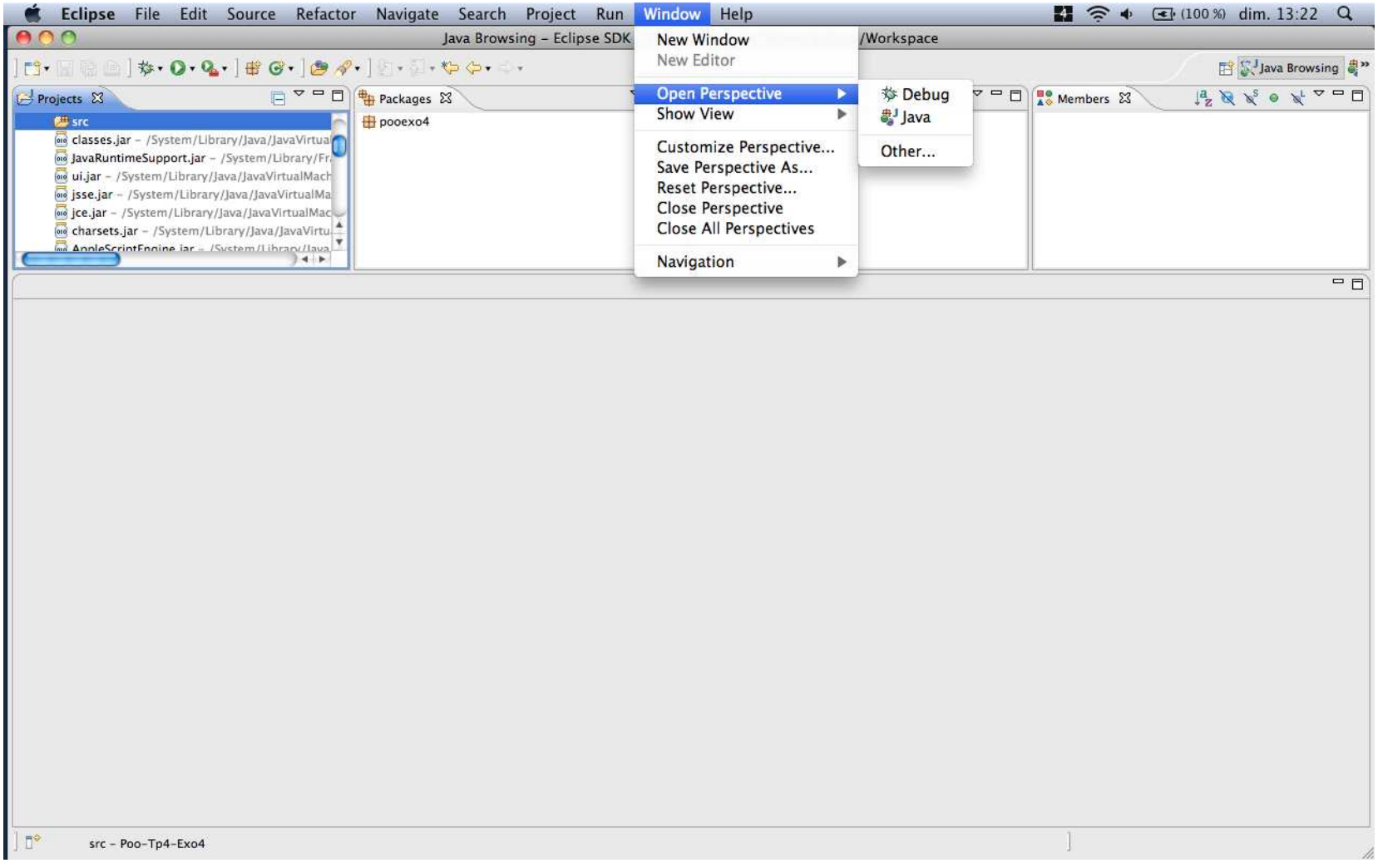
Vues

- Plan de travail : environnement multi-fenêtre
- Chaque fenêtre présente l'état du projet selon un certain point de vue
- Exemple :
  - Une fenêtre montre l'ensemble des classes
  - Une autre fenêtre permet de naviguer d'un projet à un autre
- L'éditeur est une fenêtre spécial
- Différents éditeurs selon le type de document ouvert (Java, développement dIHM)
- Fenêtre d'édition : lieu principal pour le développement de code
- JDT possède un éditeur riche en possibilités

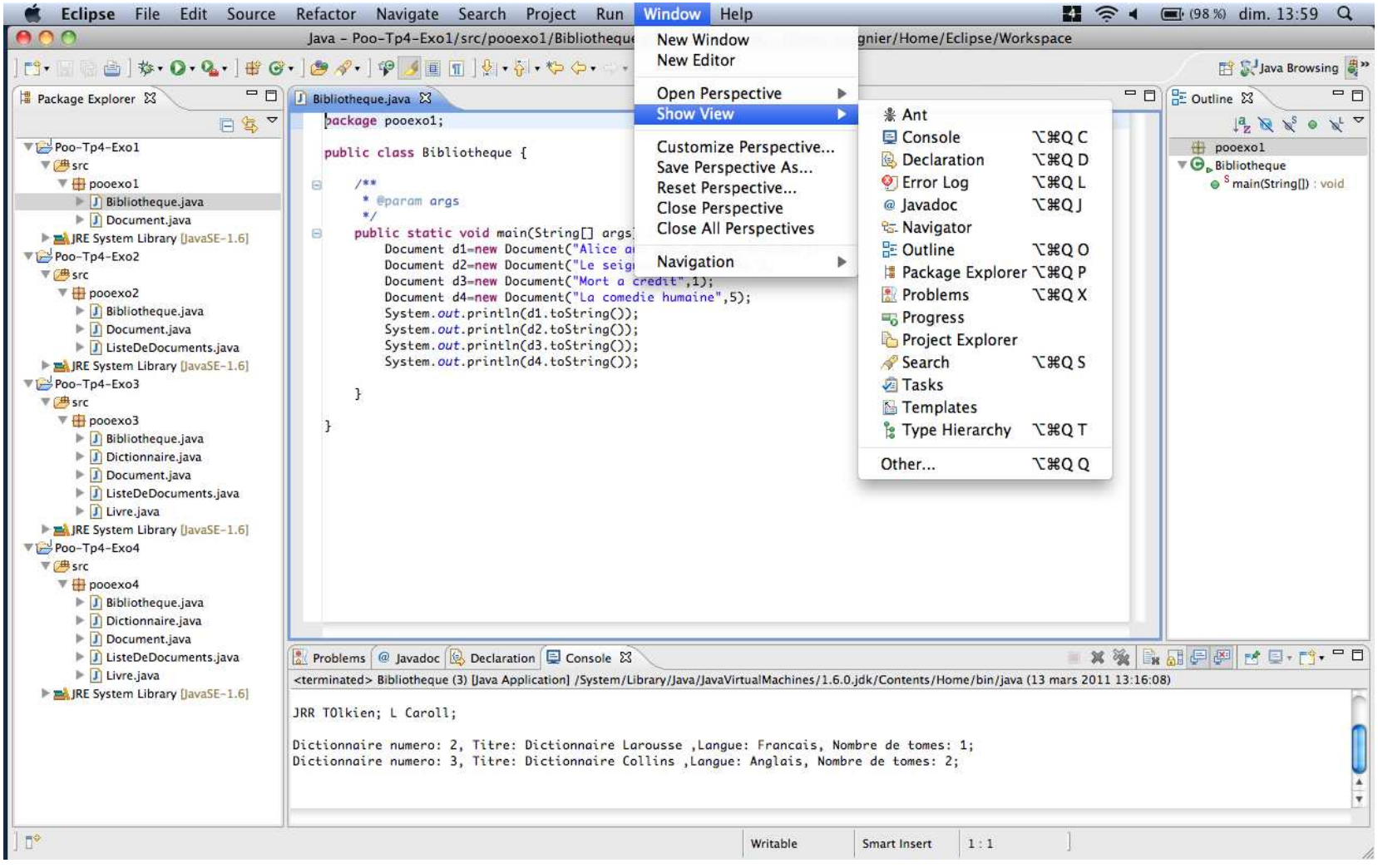
Perspectives

- En général, le programmeur ne choisit pas les vues et les éditeurs associés
- Perspectives permettent de prédéfinir un ensemble de vues et d'éditeurs automatiquement
- Par exemple :
  - Perspective Java pour une application Java
  - Perspective Débogage lorsque l'on déboque un programme
- Possibilité de créer ses propres perspectives

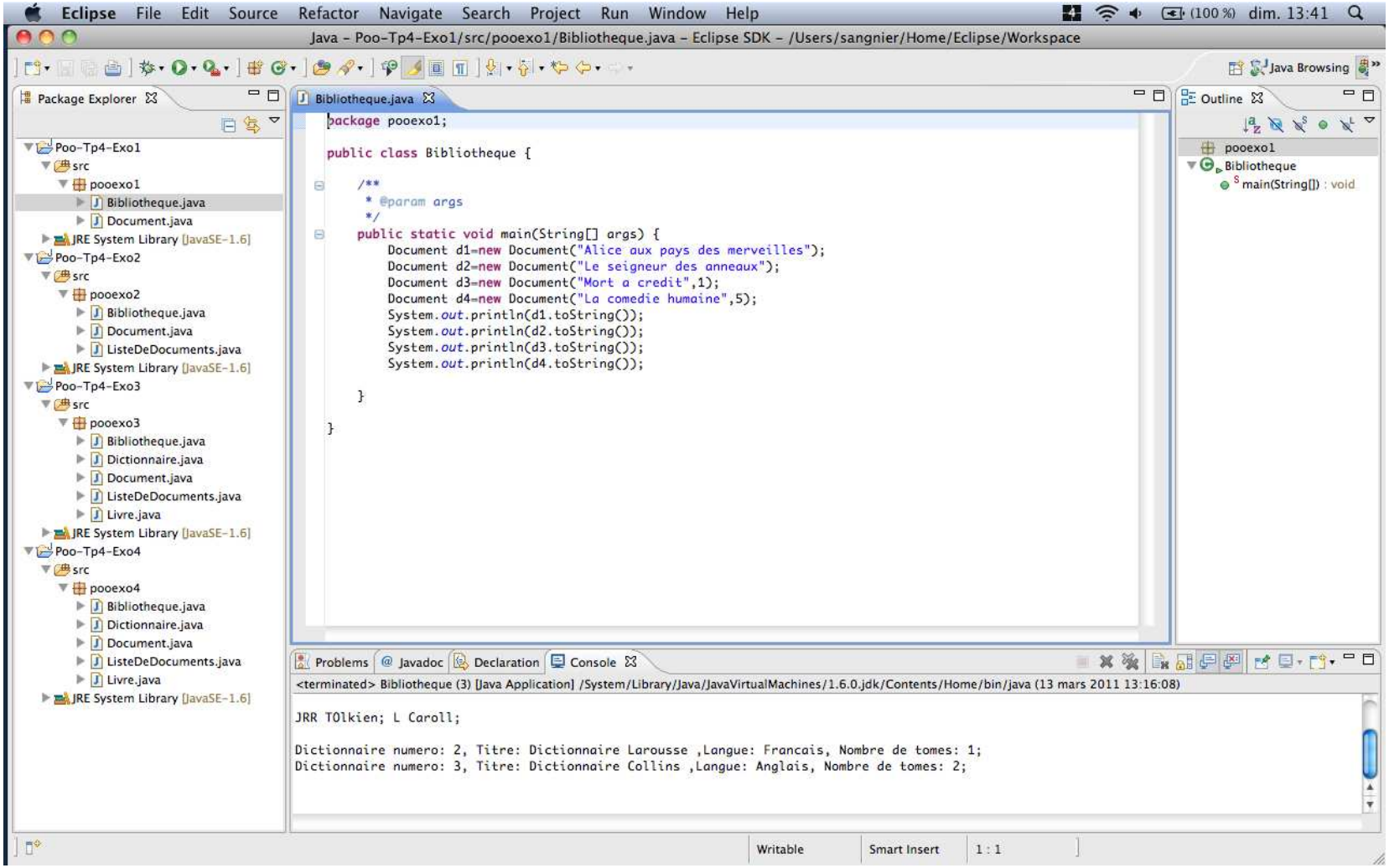
Ouvrir une perspective



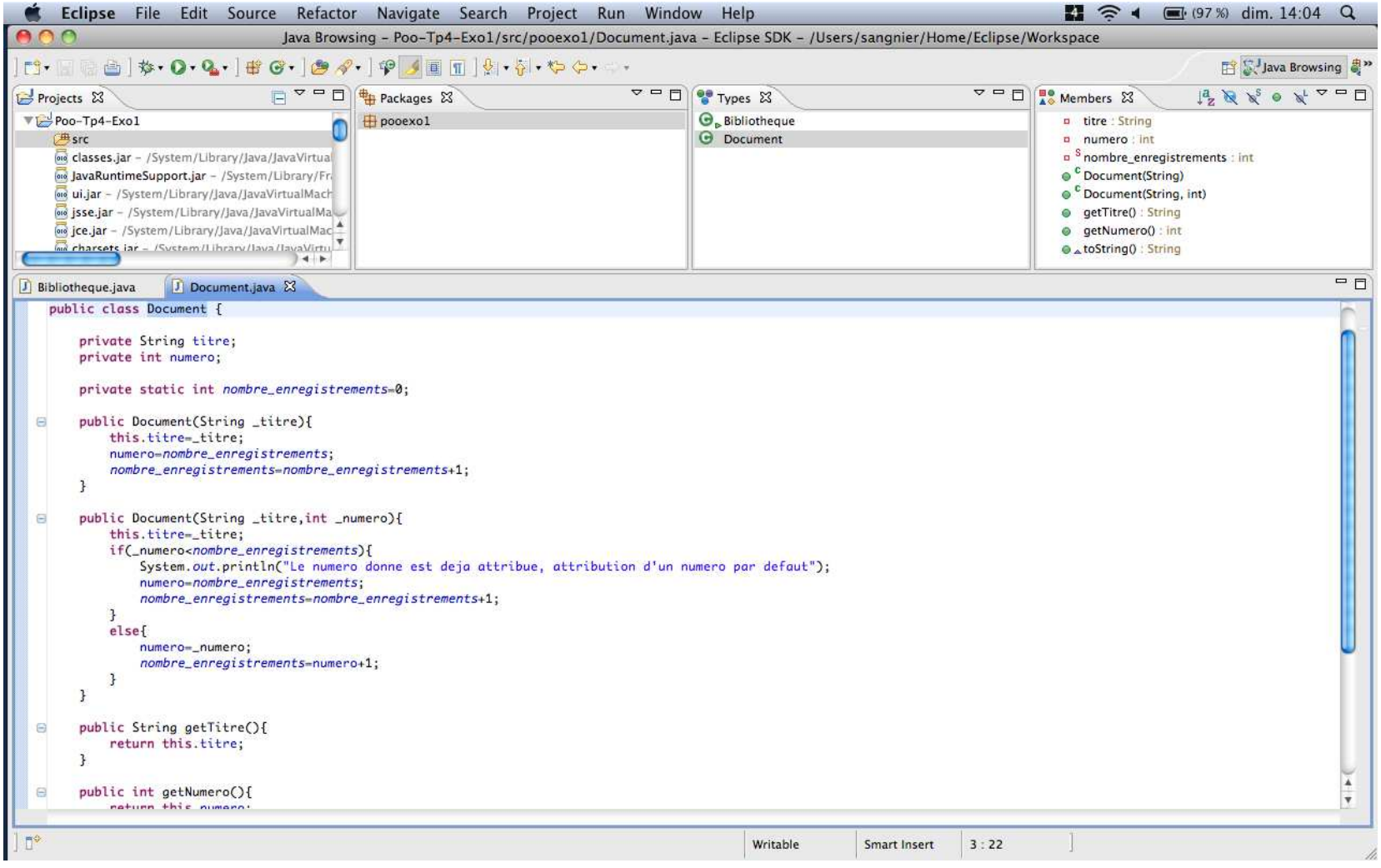
Ouvrir une vue



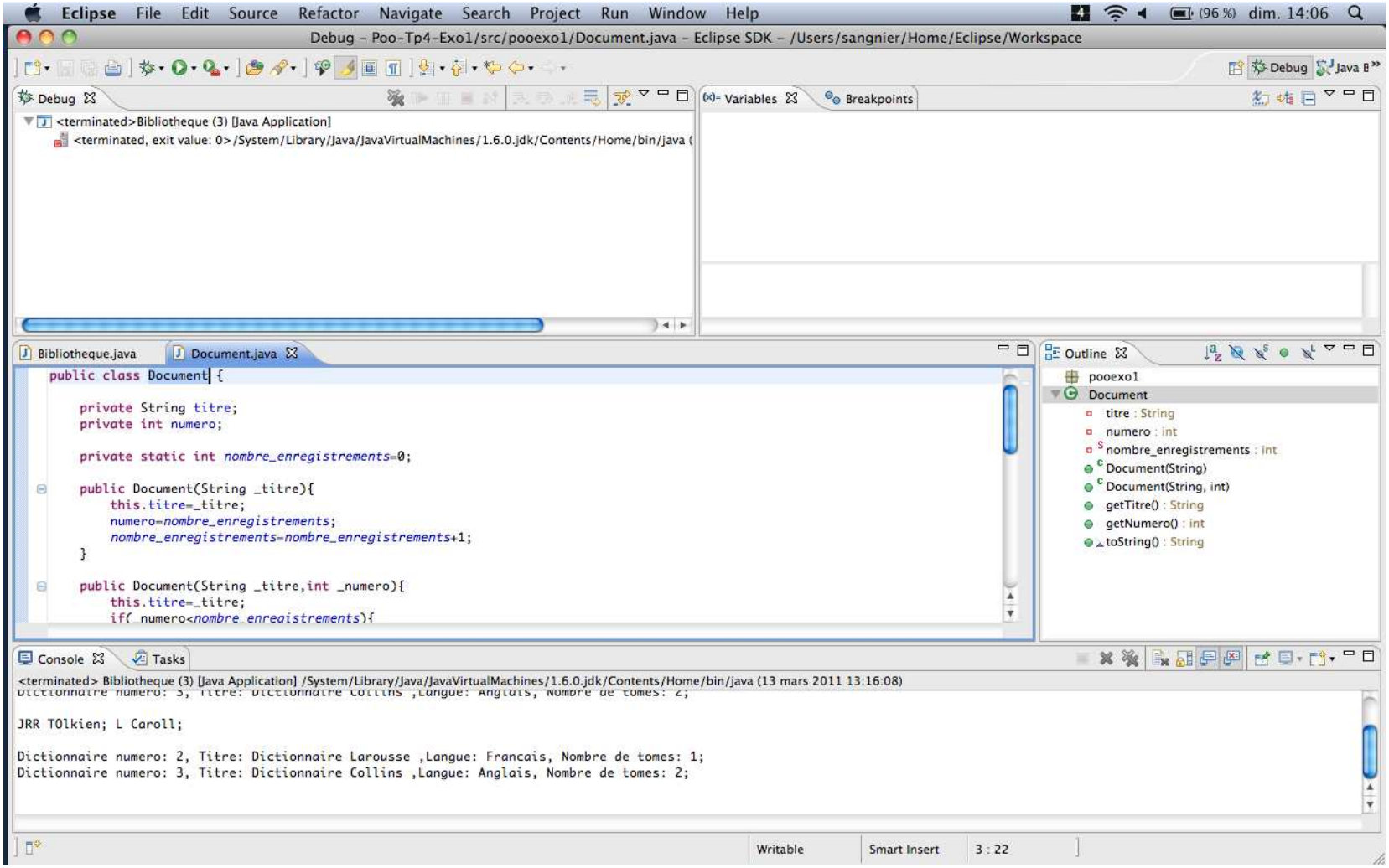
Perspective Java



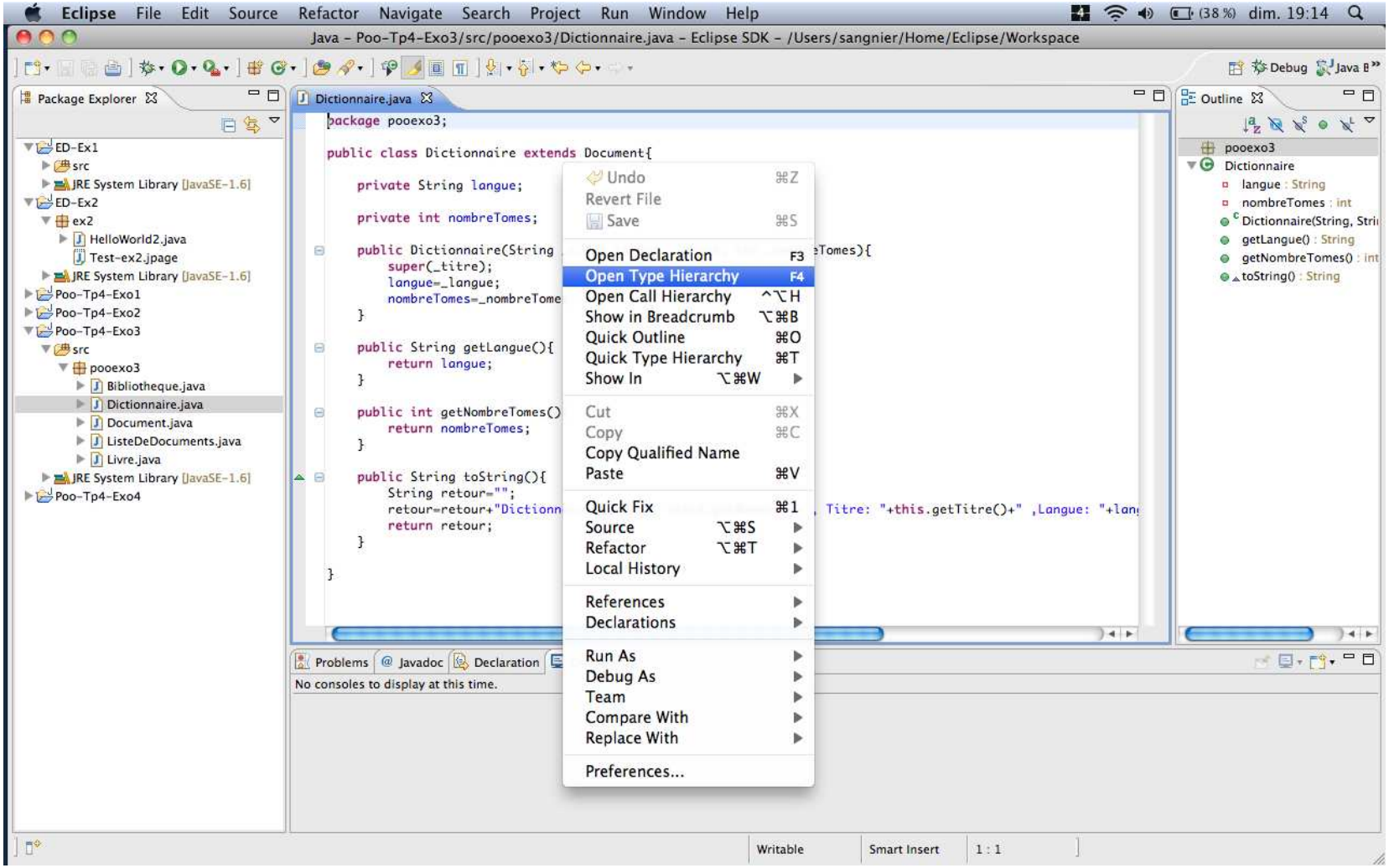
Perspective Java Browsing



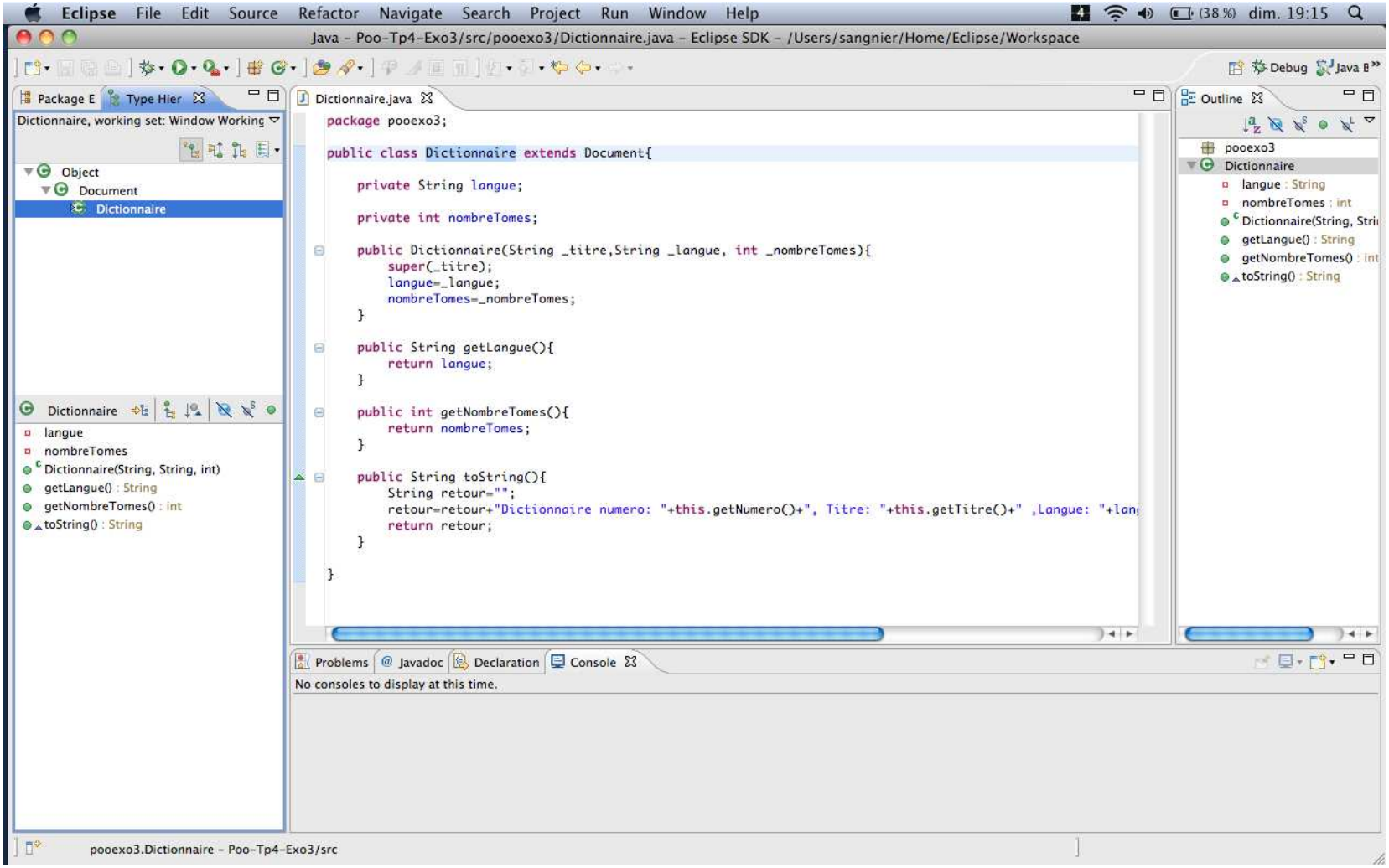
Perspective Débogage



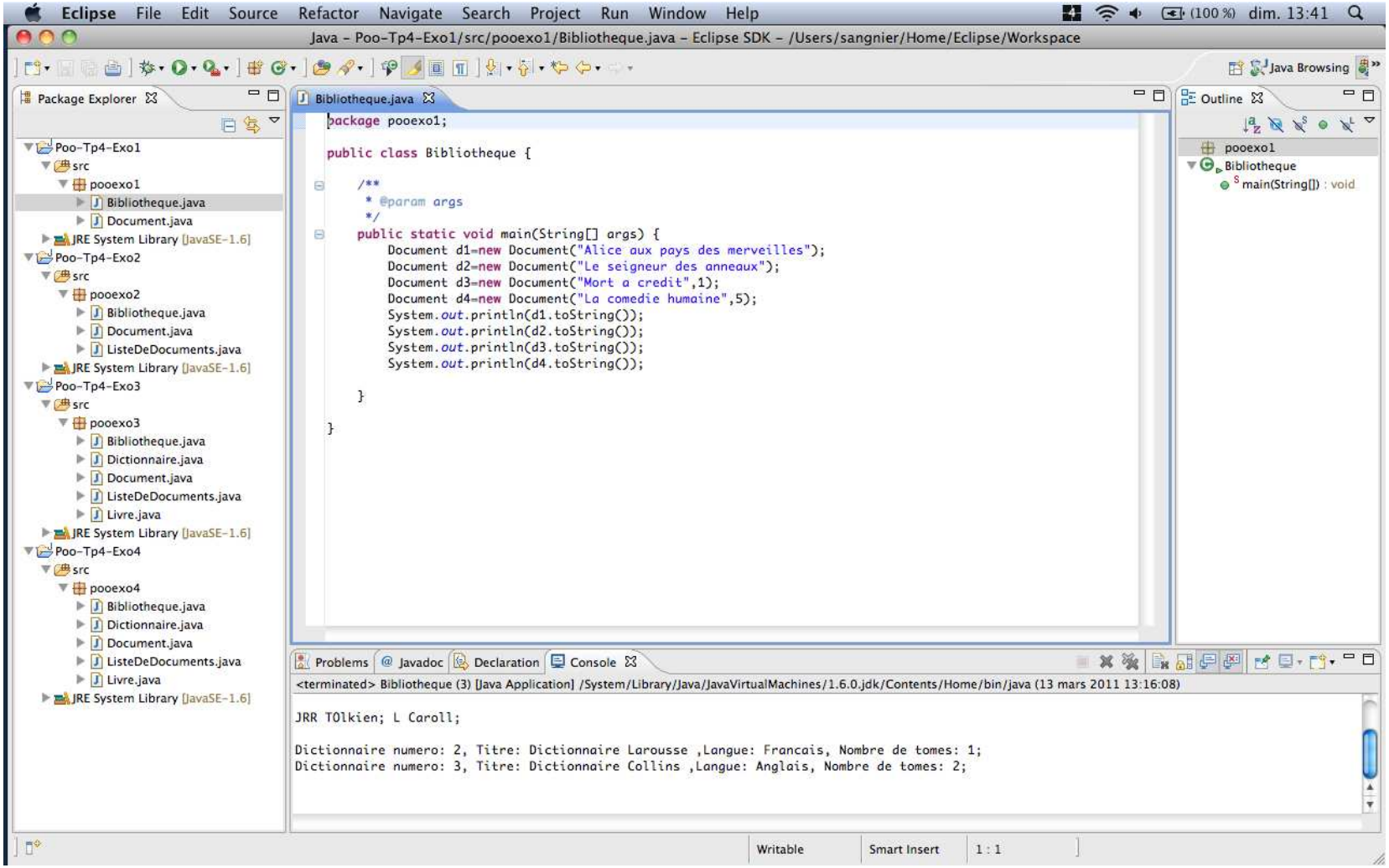
Vue Hiérarchie de Types



Vue Hiérarchie de Type



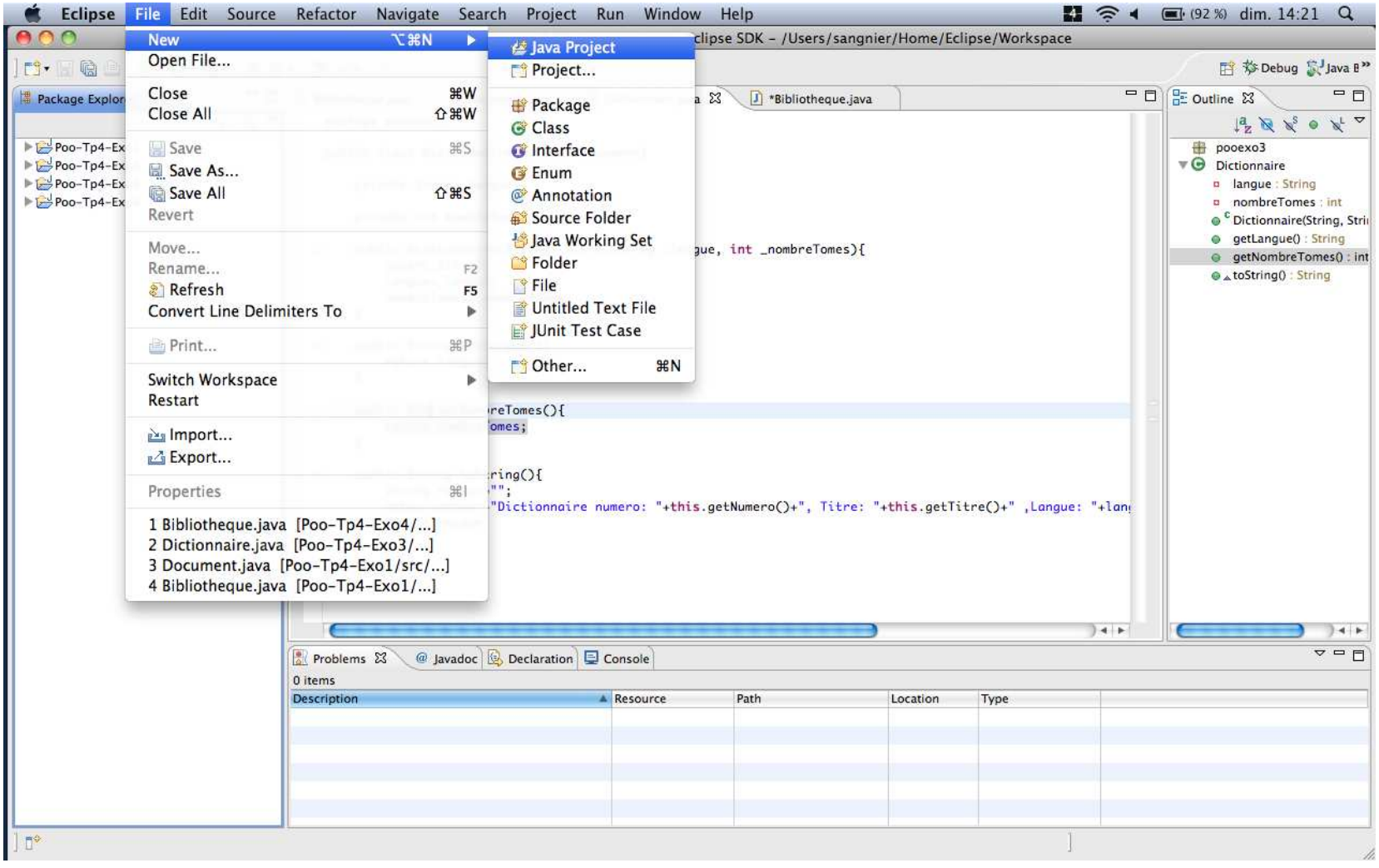
Perspective Java (II)



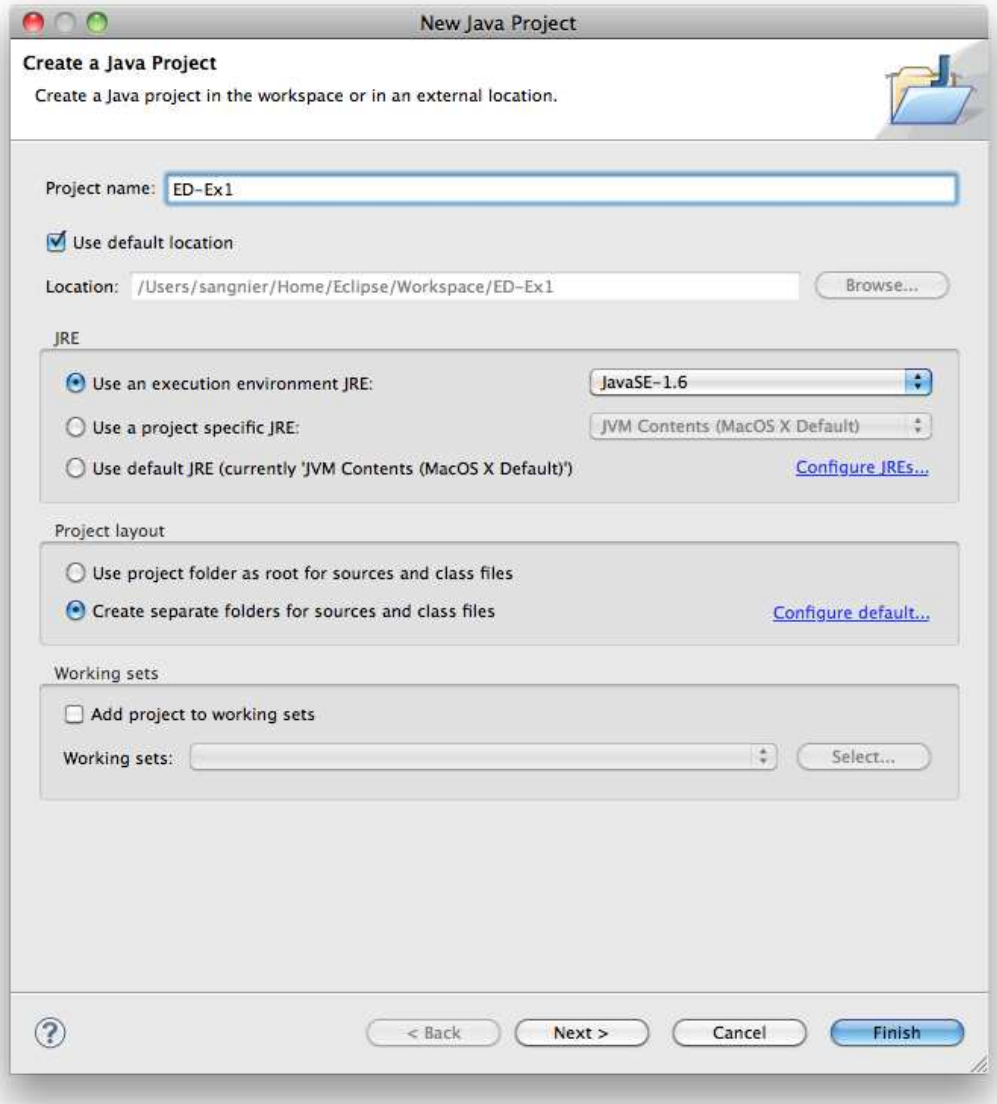
Détails de la perspective Java

- Lorqu'on ouvre cette perspective, le JDT est lancé
- Perpective utilisée pour le développement de programme Java
- Éditeur Java au milieu
- À gauche, on trouve la vue Package :
  - Propose une vue d'ensemble des paquetages développés
  - Permet de naviguer dans différents projets en cours
- À droite, la vue Structure :
  - Propose une vision hiérarchique du contenu du fichier ouvert dans l'éditeur
  - Particulièrement utile pour naviguer dans de longs fichiers de code
- Dans la partie inférieure :
  - La vue Console : montre le résultat de l'exécution du programme sur la console de sortie
  - Différentes vues pour montrer par exemple les erreurs liées

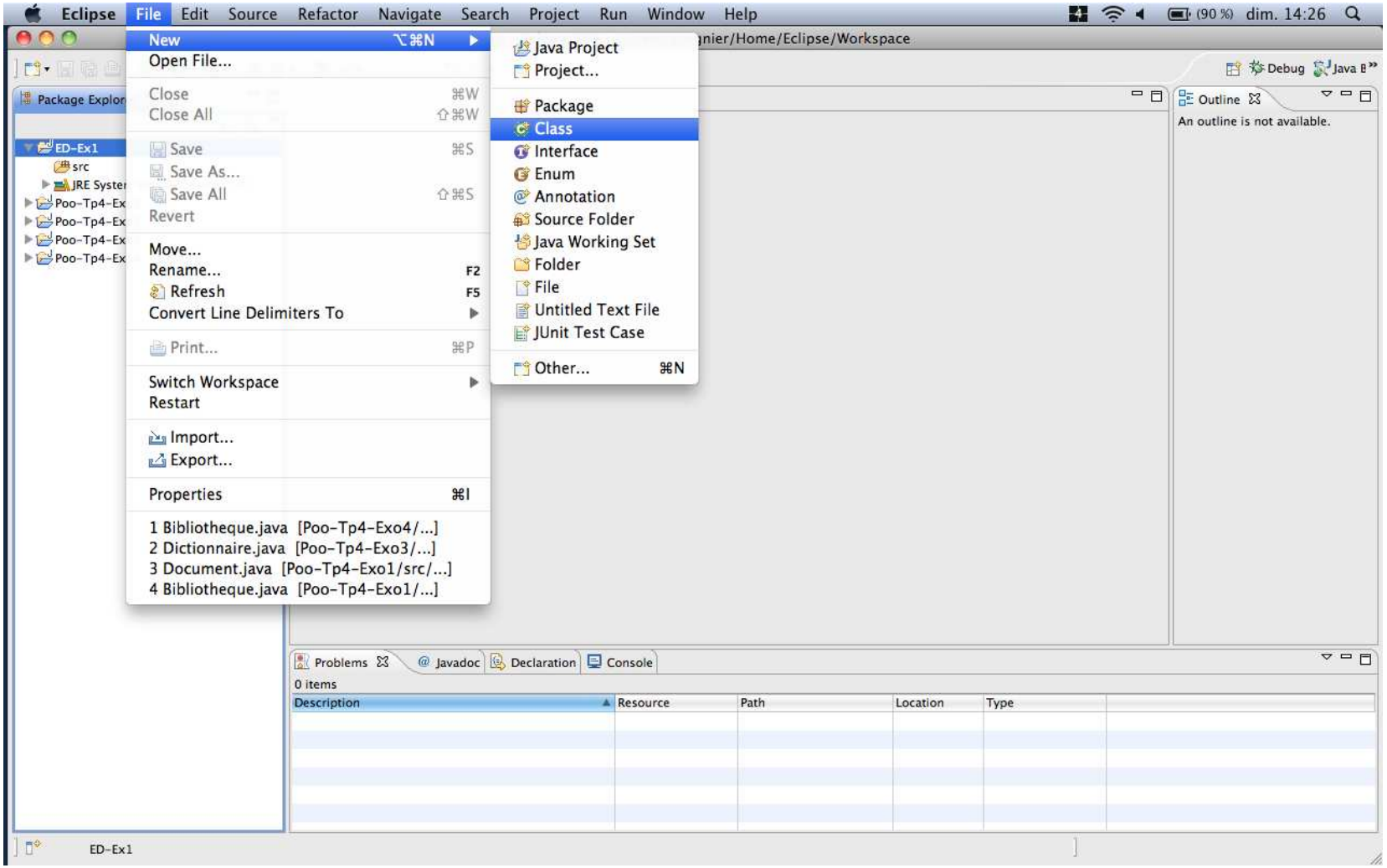
Création d'un nouveau projet



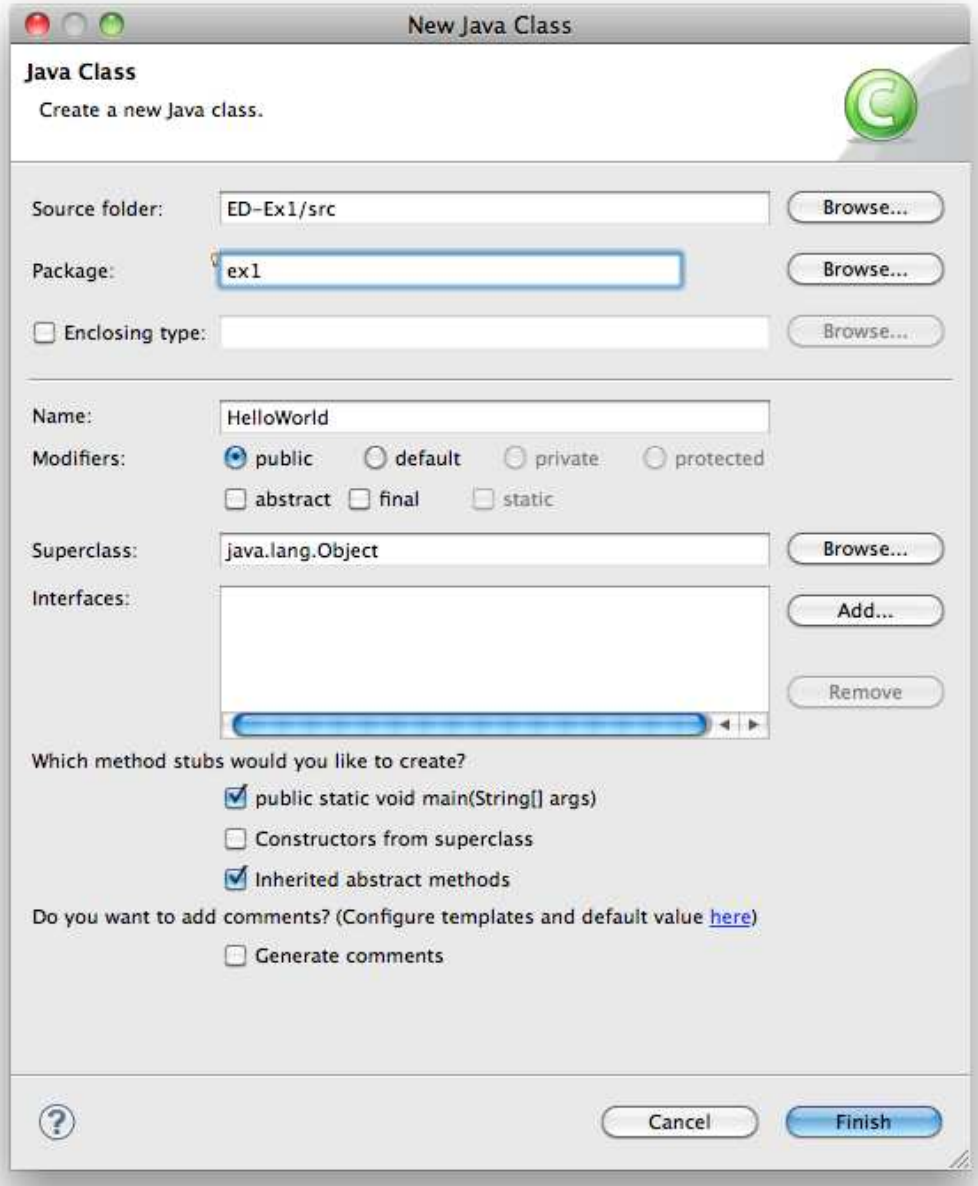
Création d'un nouveau projet



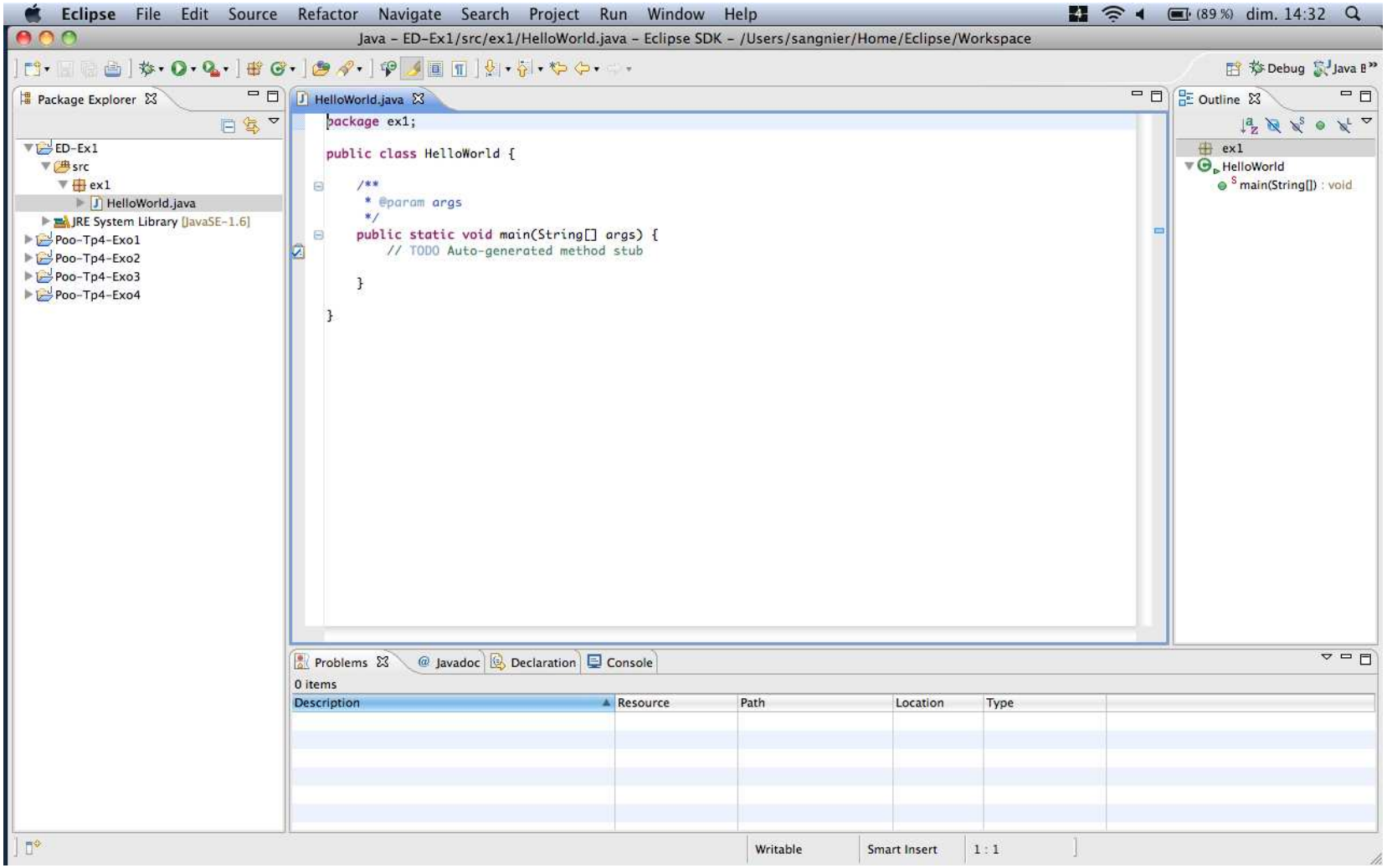
Création d'une nouvelle classe



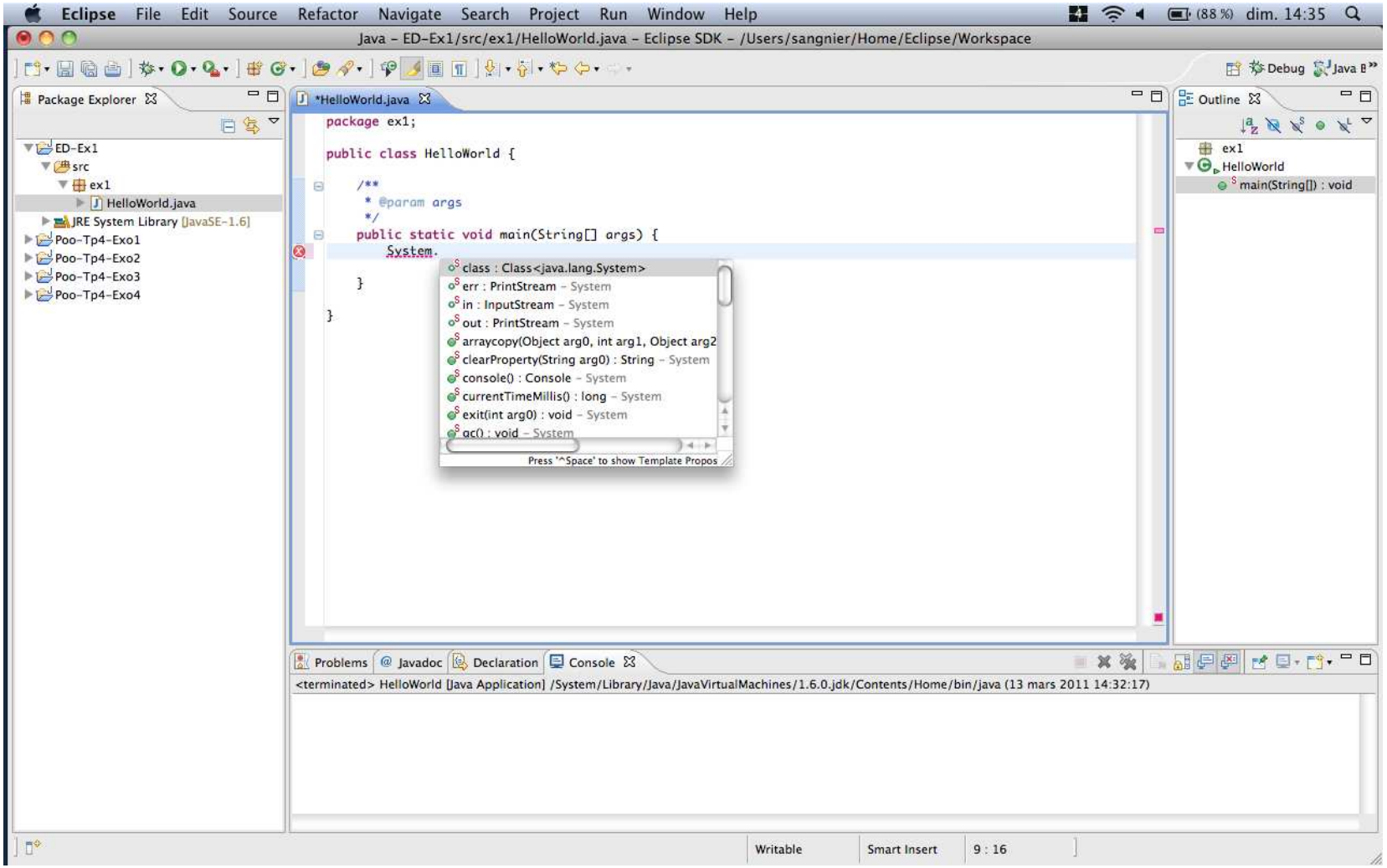
Création d'une nouvelle classe



Création d'une nouvelle classe

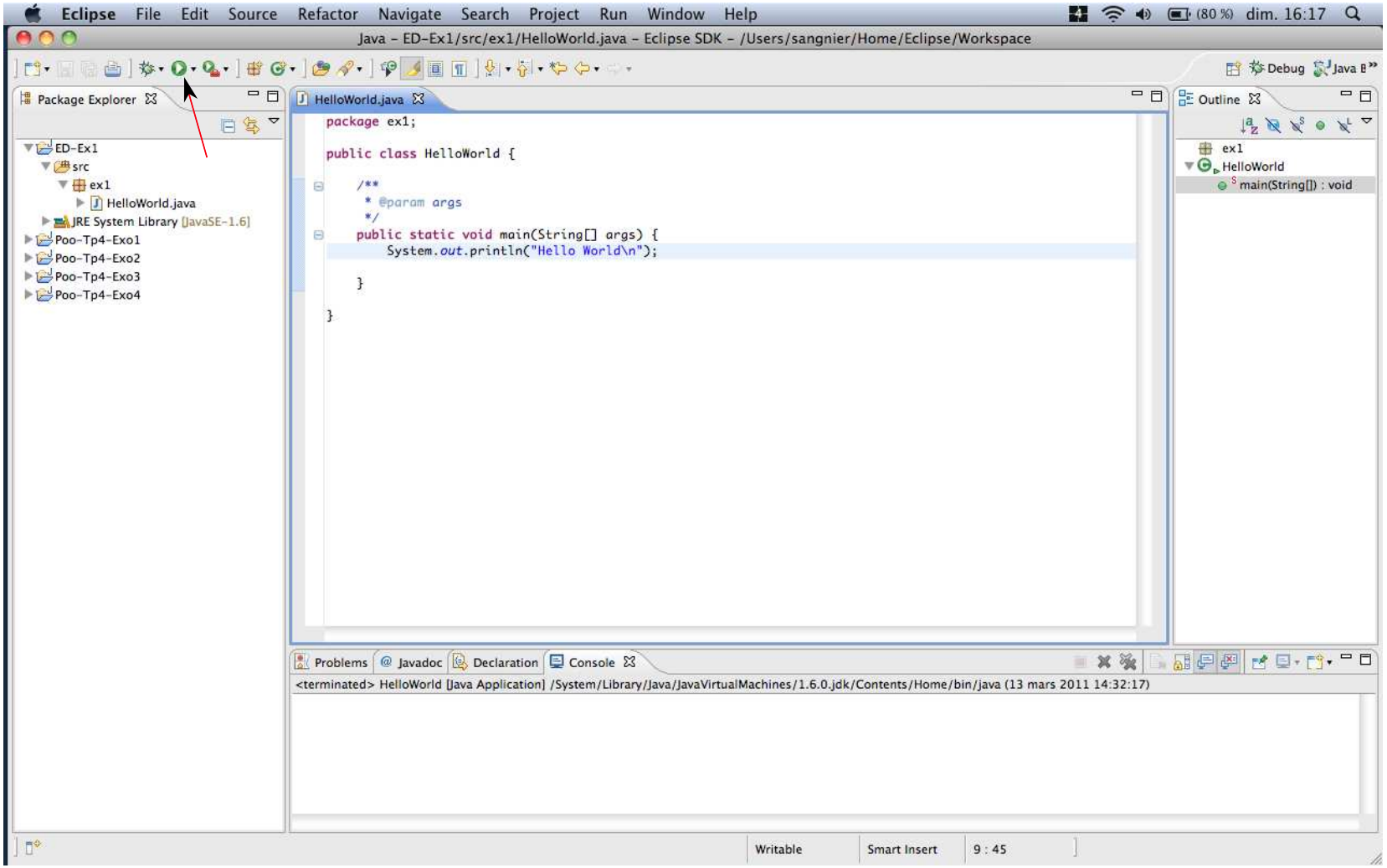


Utilisation de l'assistant de code

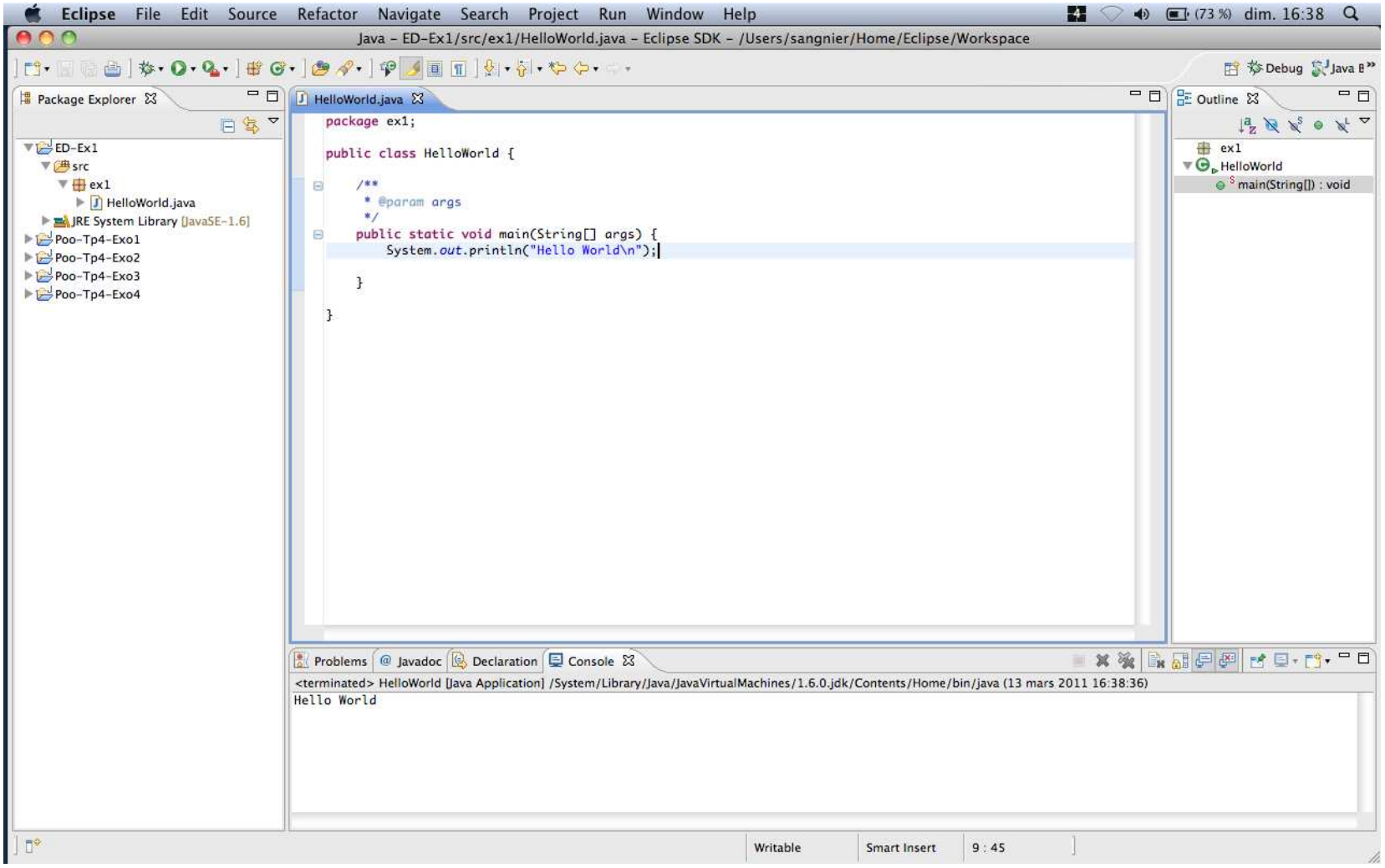


L'assistant s'ouvre tout seul après tout point et peut aussi être appelé par la touche Ctrl+Espace

Exécution premier programme



Résultat de l'exécution du programme dans la console

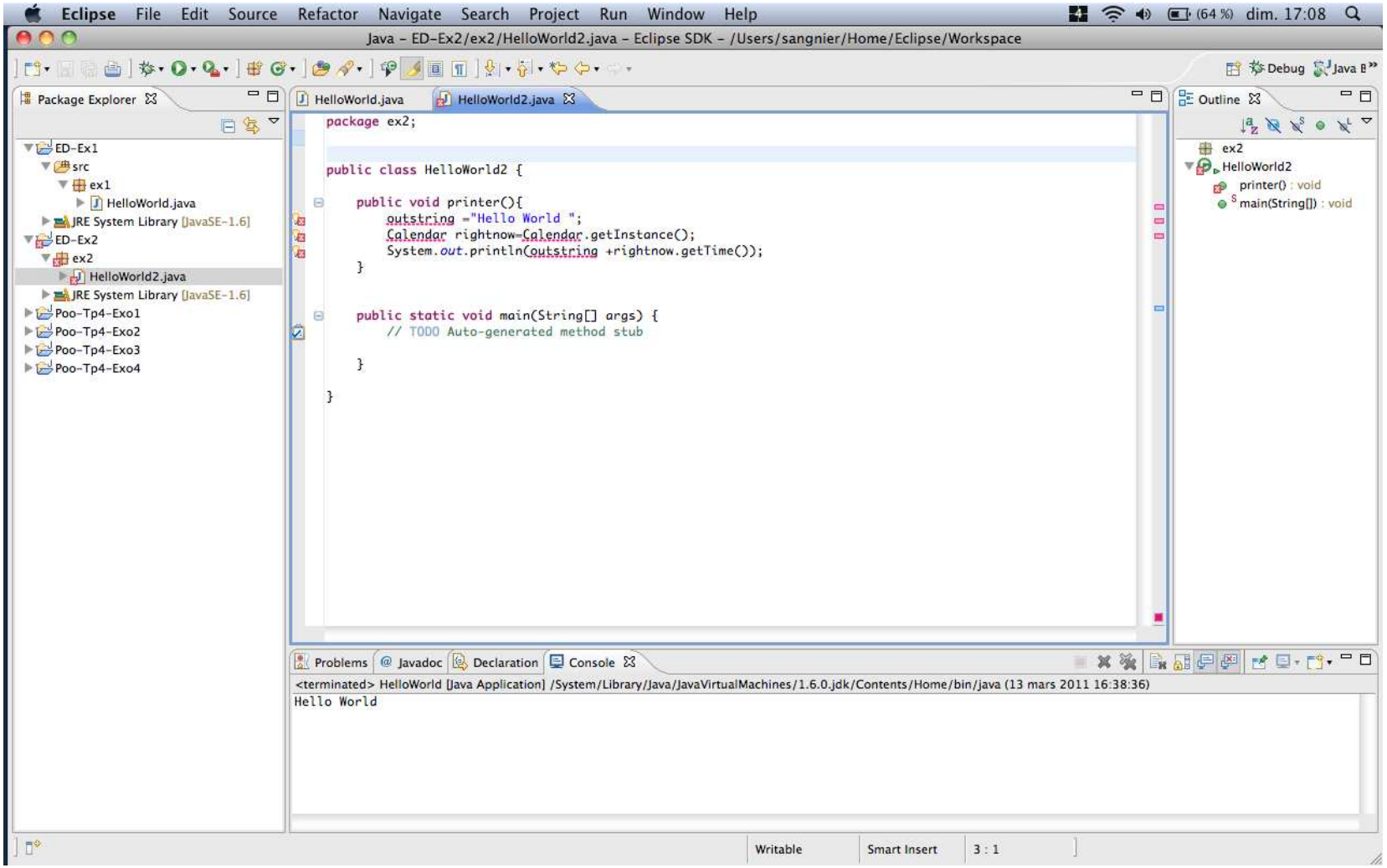


Quelques remarques

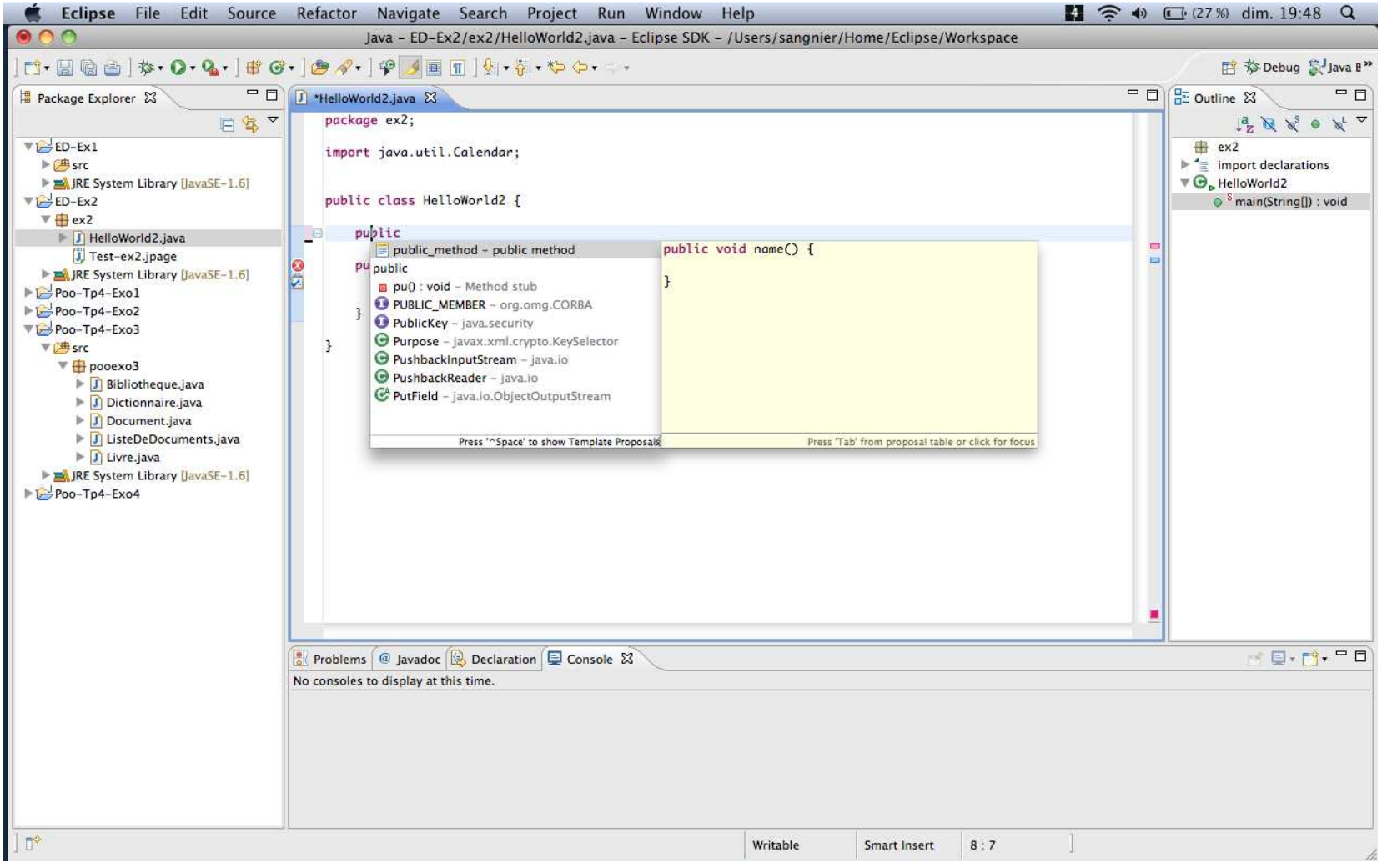
Où se trouve votre programme :

- Dans votre workspace, un répertoire ED-Ex1 a été créé
- Si vous avez cochez la case "Create separate folders for sources and class files" lors de la création de projet, alors dans ED-Ex1, vous avez deux sous répertoires :
  1. src qui contient les fichiers .java
  2. bin qui contient les fichiers .class
- Dans src et bin vous avez un répertoire ex1 (nom du package)
- HelloWorld.class se trouve alors dans ED-Ex1/bin/ex1
- Pour l'exécuter, vous devez vous mettre dans ED-Ex1/bin et faire : java ex1.HelloWorld
- Ou faire :
  java -classpath [VotreWorkspace]/ED-Ex1/bin :. ex1.HelloWorld

Un autre exemple



Utilisation de l'assistant de code

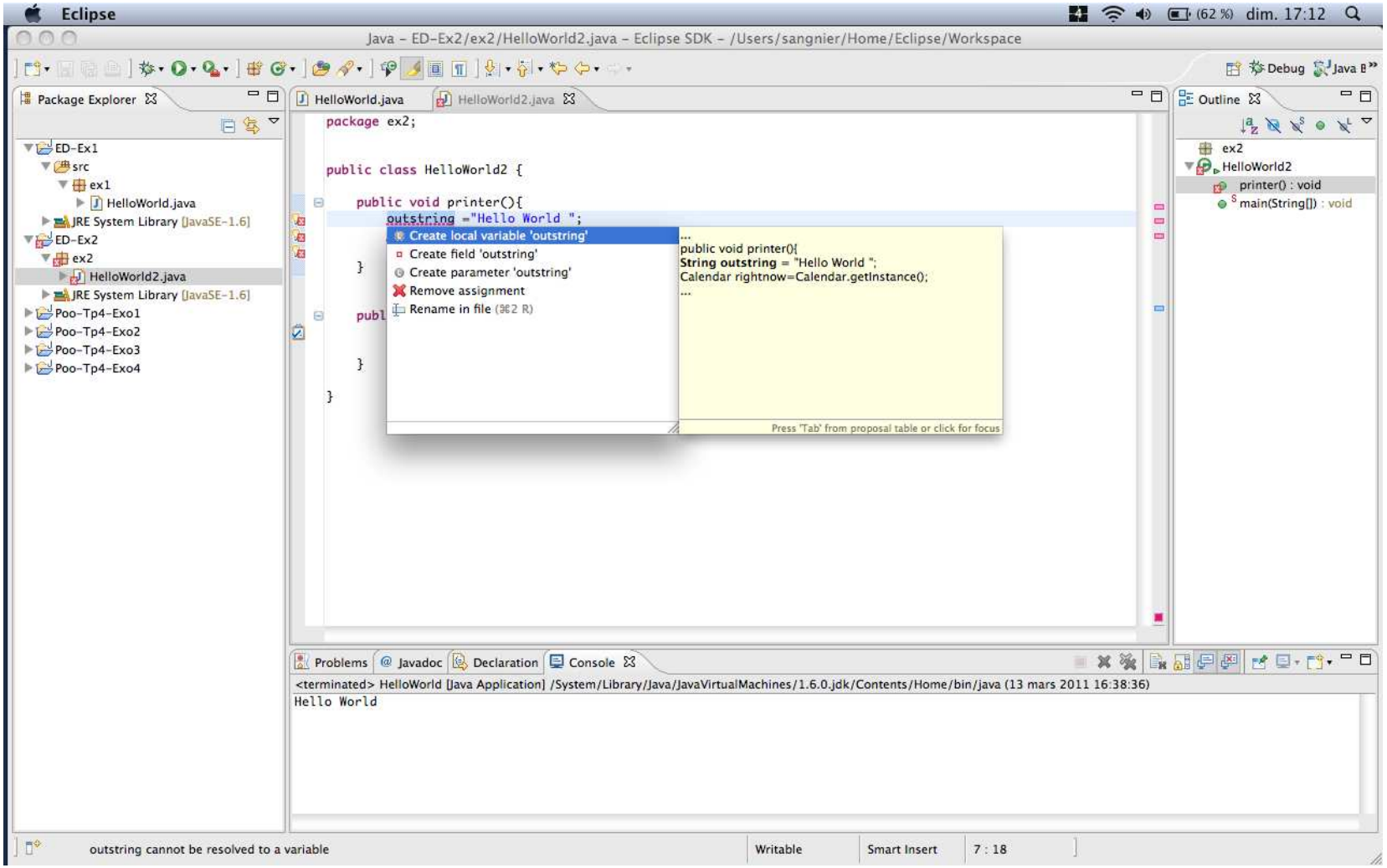


Résultat obtenu avec Ctrl+Espace

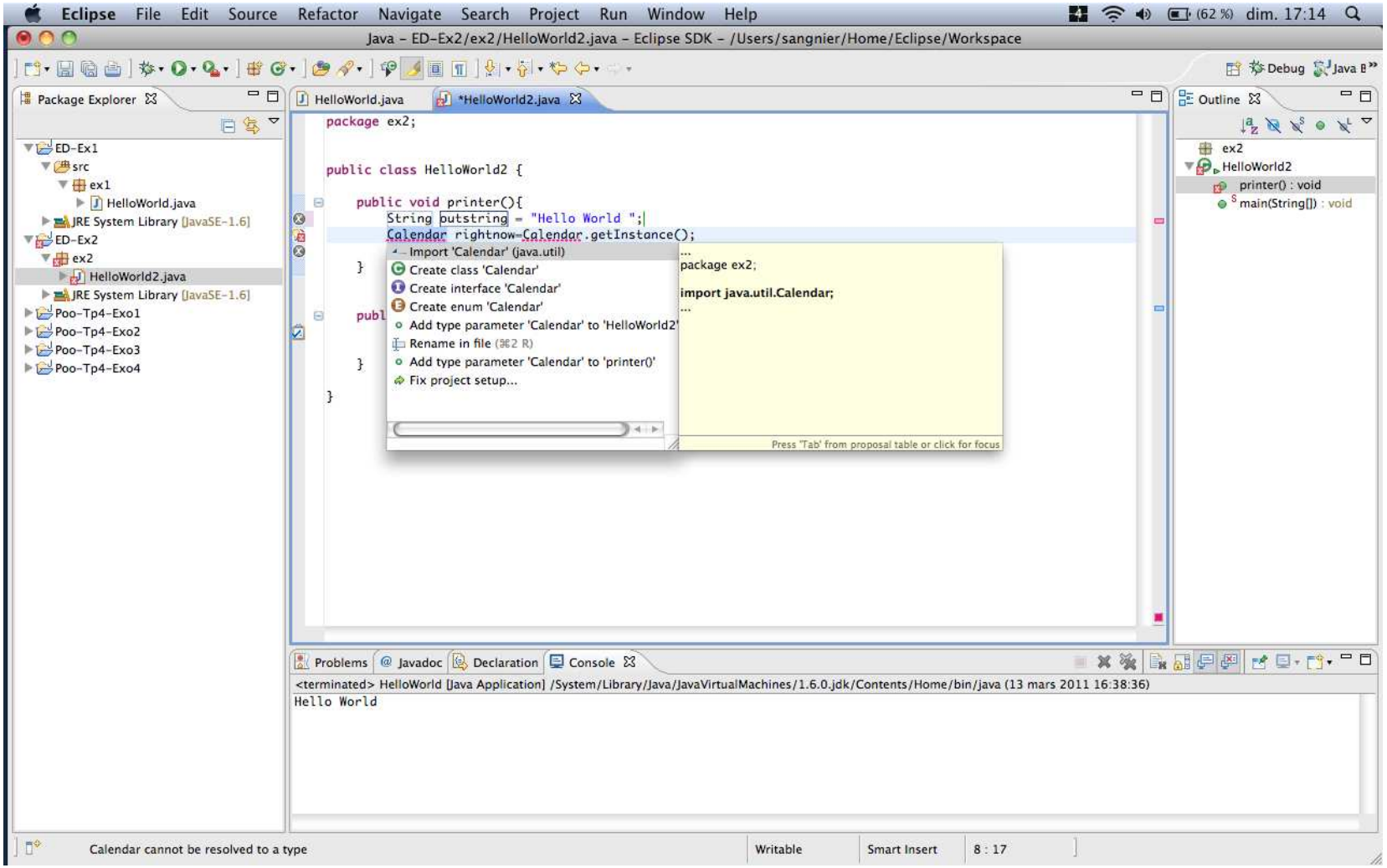
Utilisation du correcteur rapide (Quickfix)

- Outil permettant au JDT de formuler des propositions pour résoudre des erreurs simples
- Les erreurs sont indiquées dans la marge gauche de la fenêtre d'édition
- En les survolant, Eclipse indique le type de l'erreur
- En cliquant dessus, le correcteur rapide propose différentes solutions de corrections

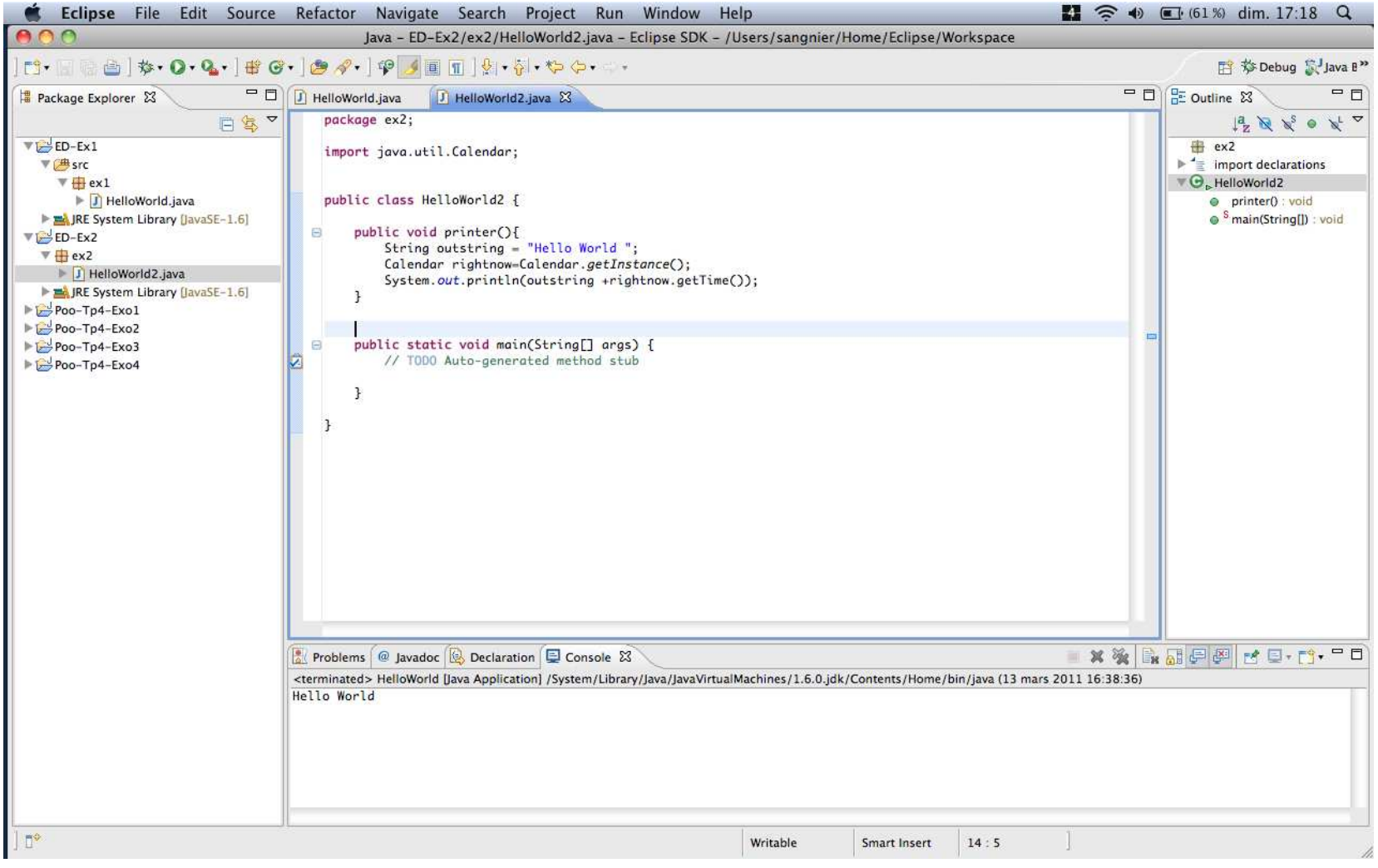
Exemples d'utilisation du correcteur rapide



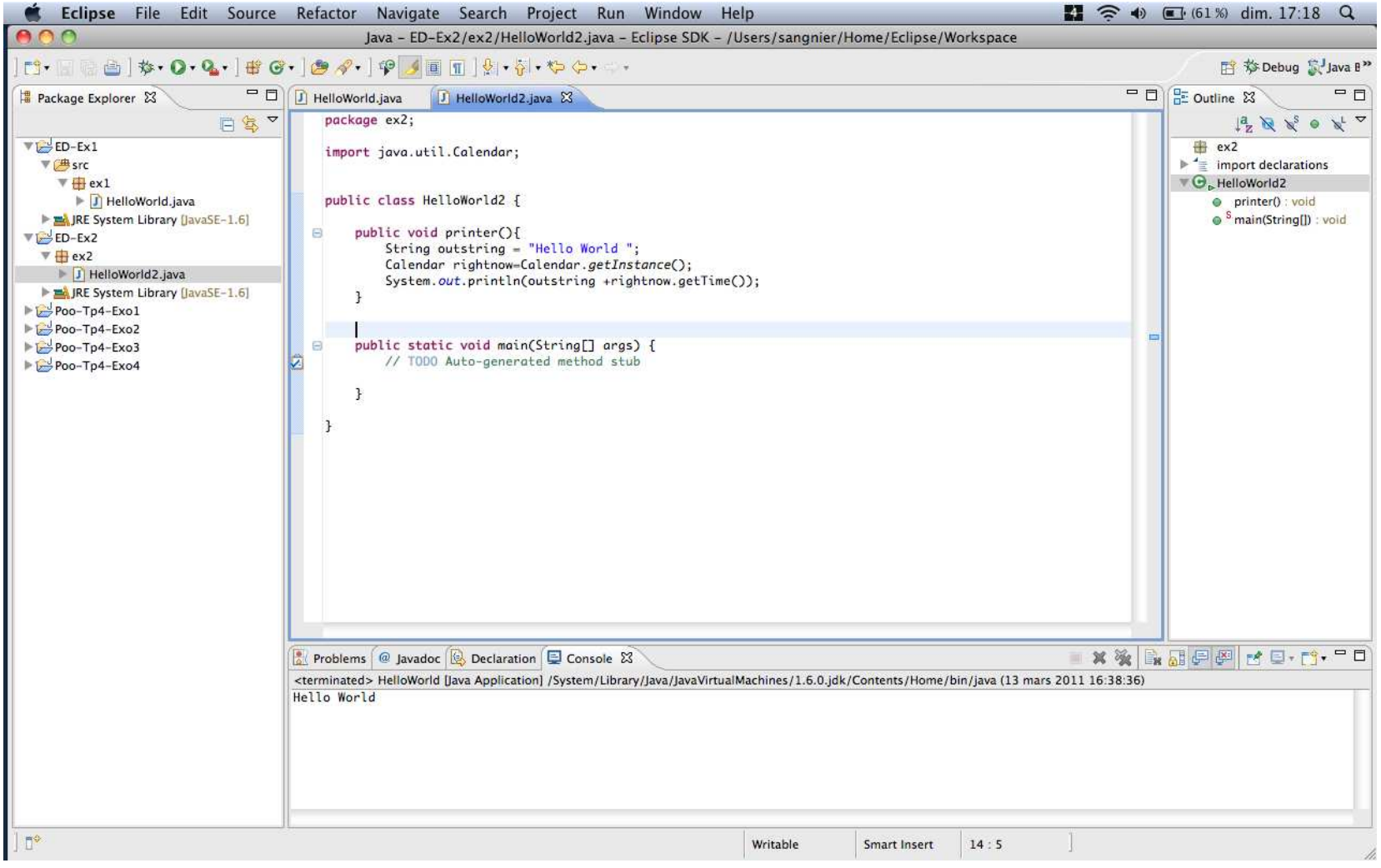
Exemples d'utilisation du correcteur rapide



Après correction



Après correction

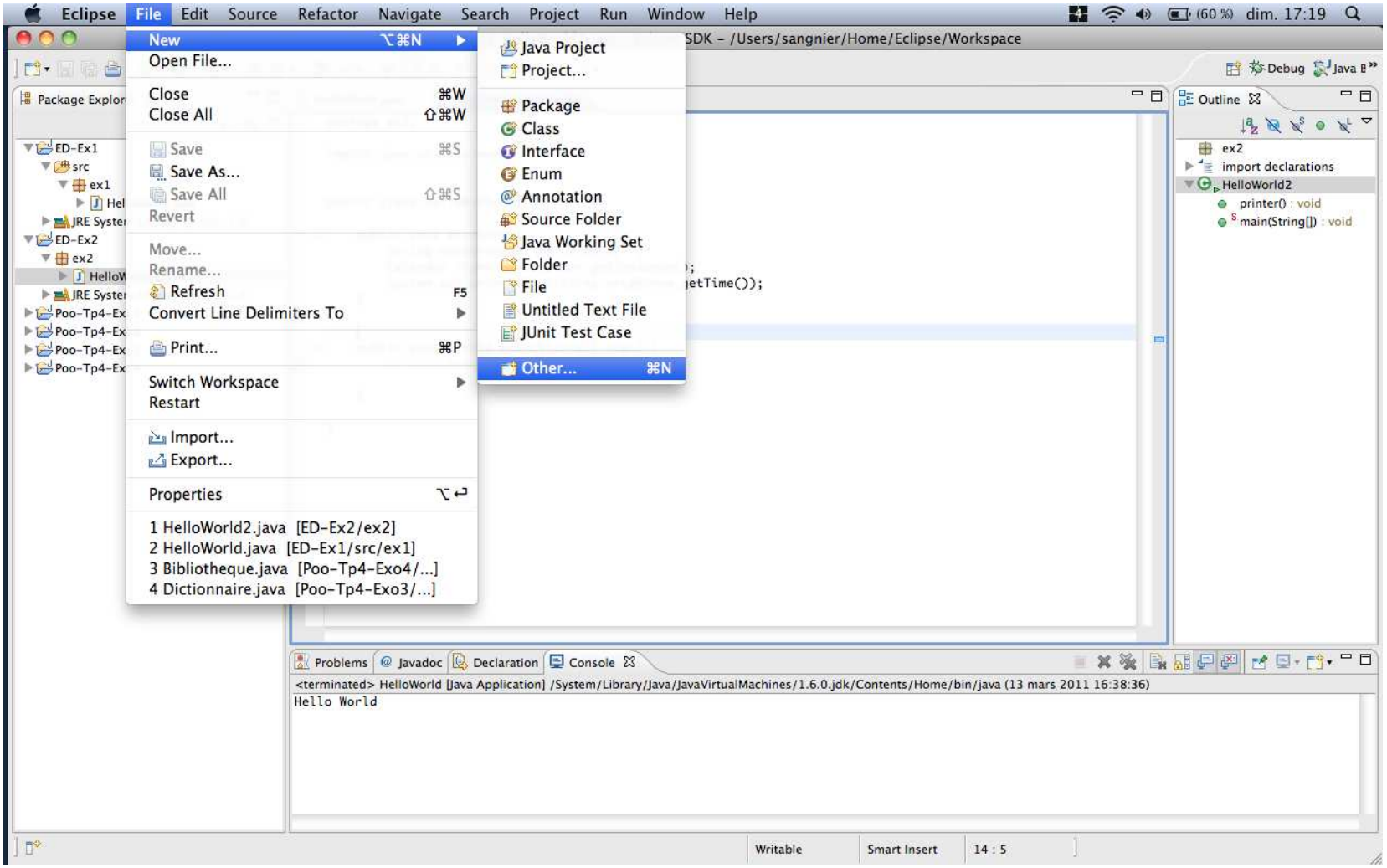


Question : Comment tester la méthode printer sans écrire le main ?

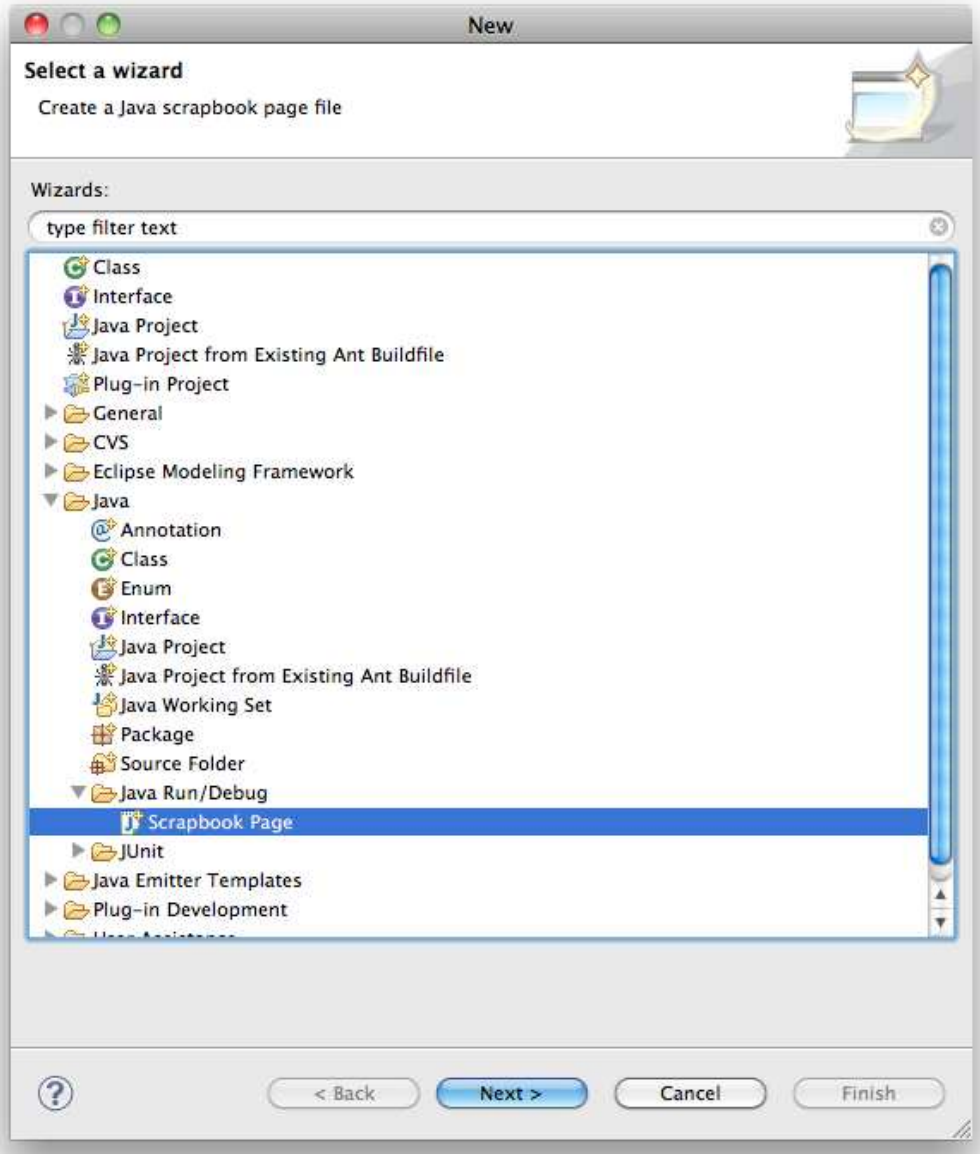
Les testeurs de code

- Il est possible d'exécuter du code dans un projet Java sans disposer d'une méthode main
- Pour cela on utilise une page de **Testeur de code**
- Ces pages permettent d'exécuter du code, partiellement à la volée

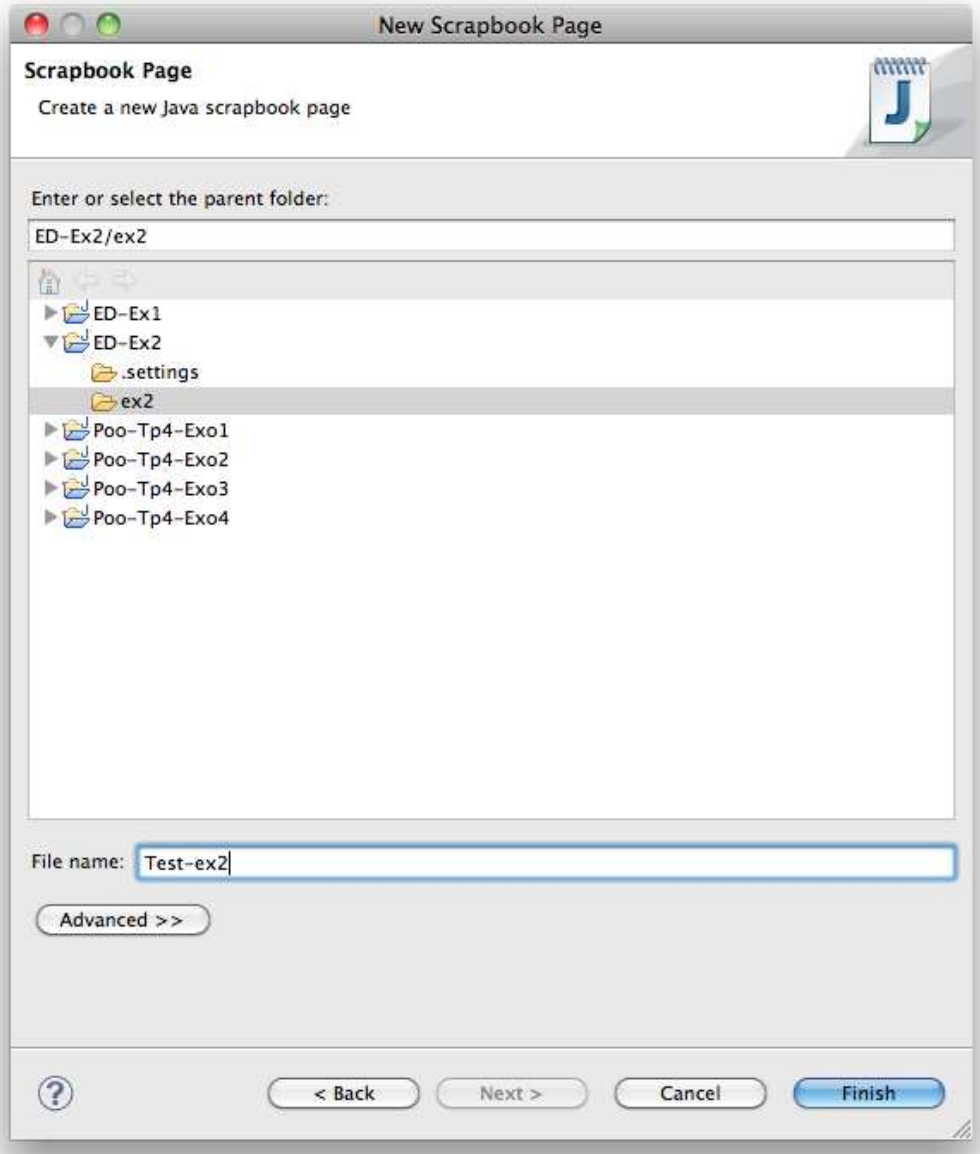
Ouvrir une page de testeur de code



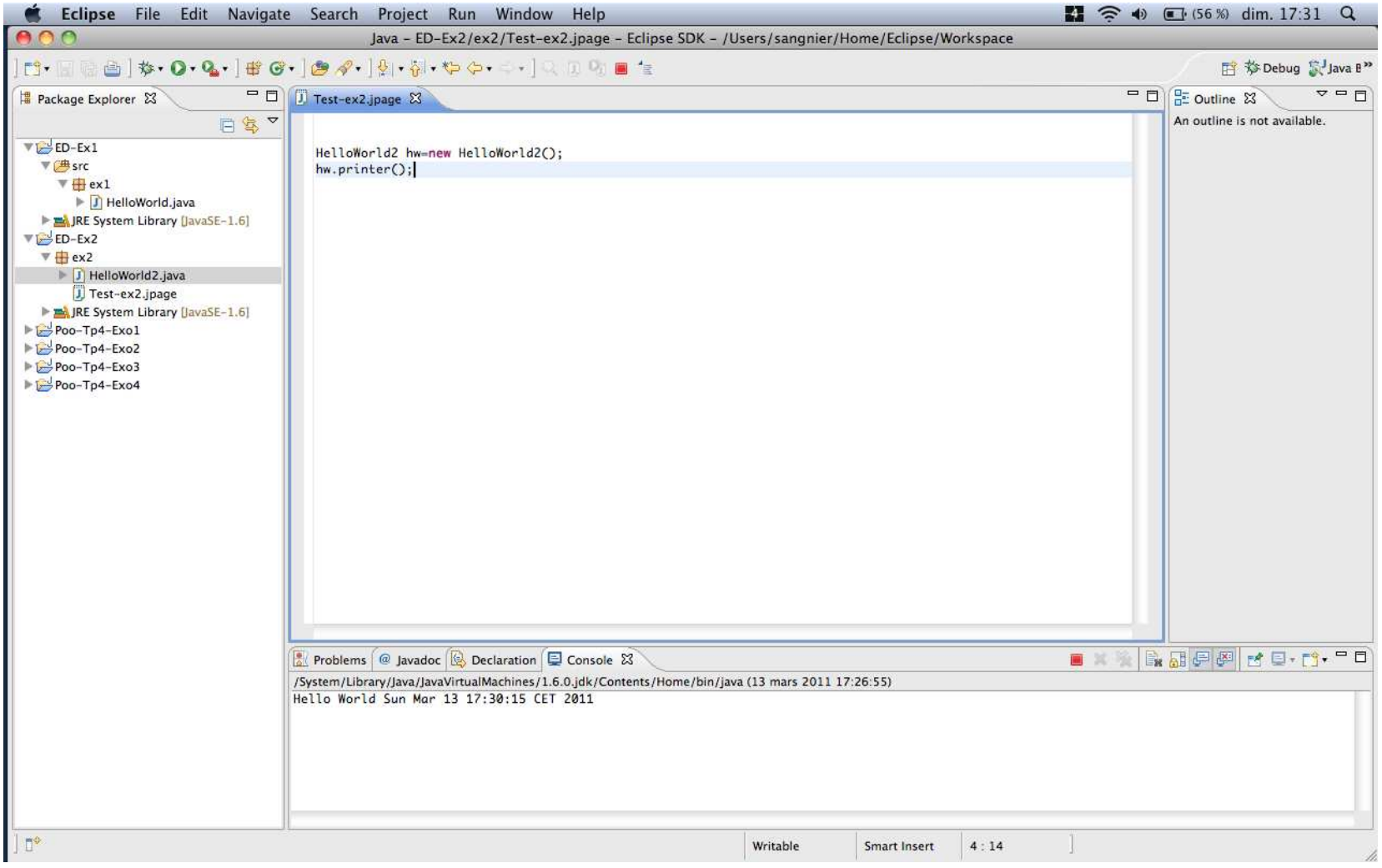
Ouvrir une page de testeur de code



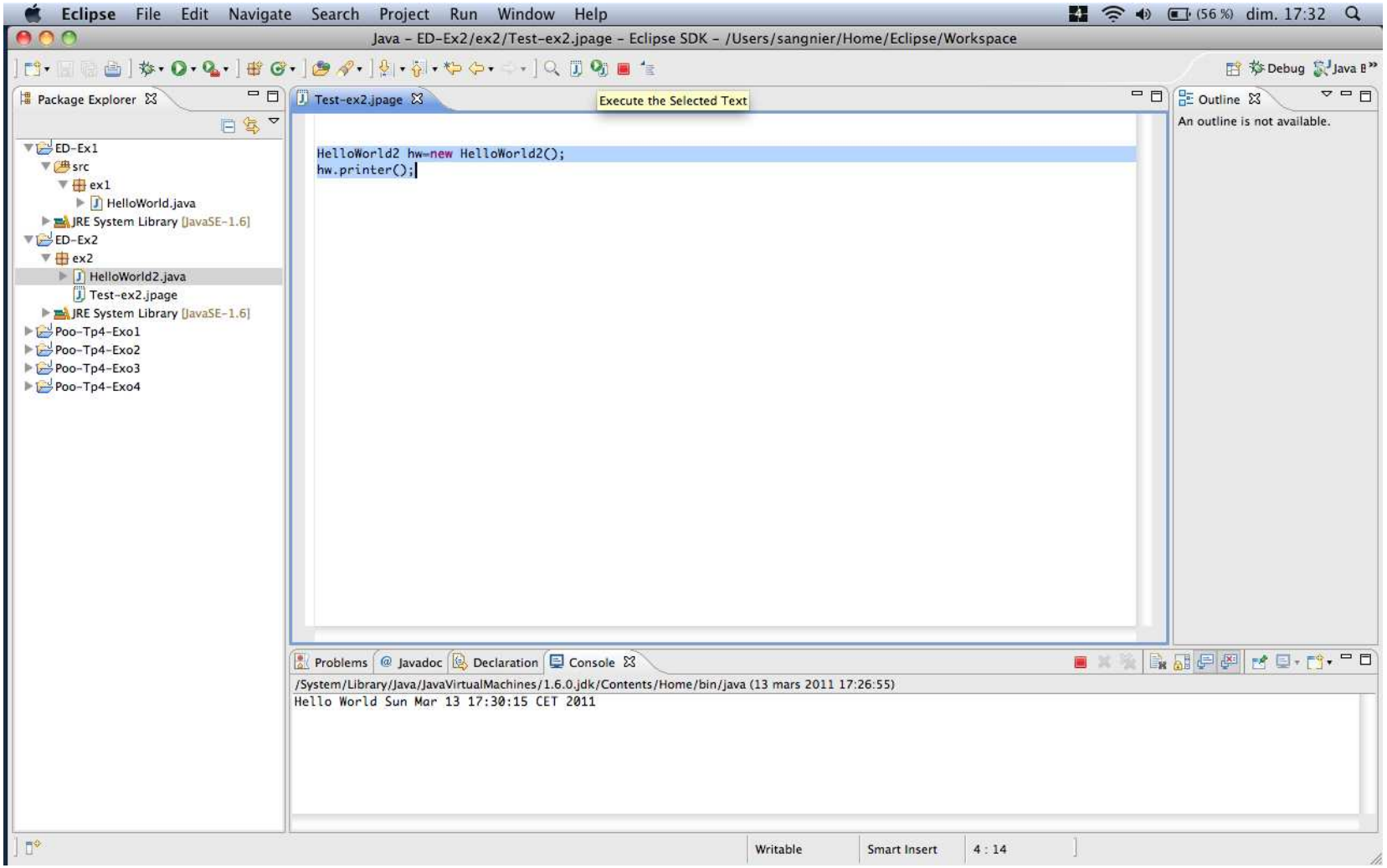
Ouvrir une page de testeur de code



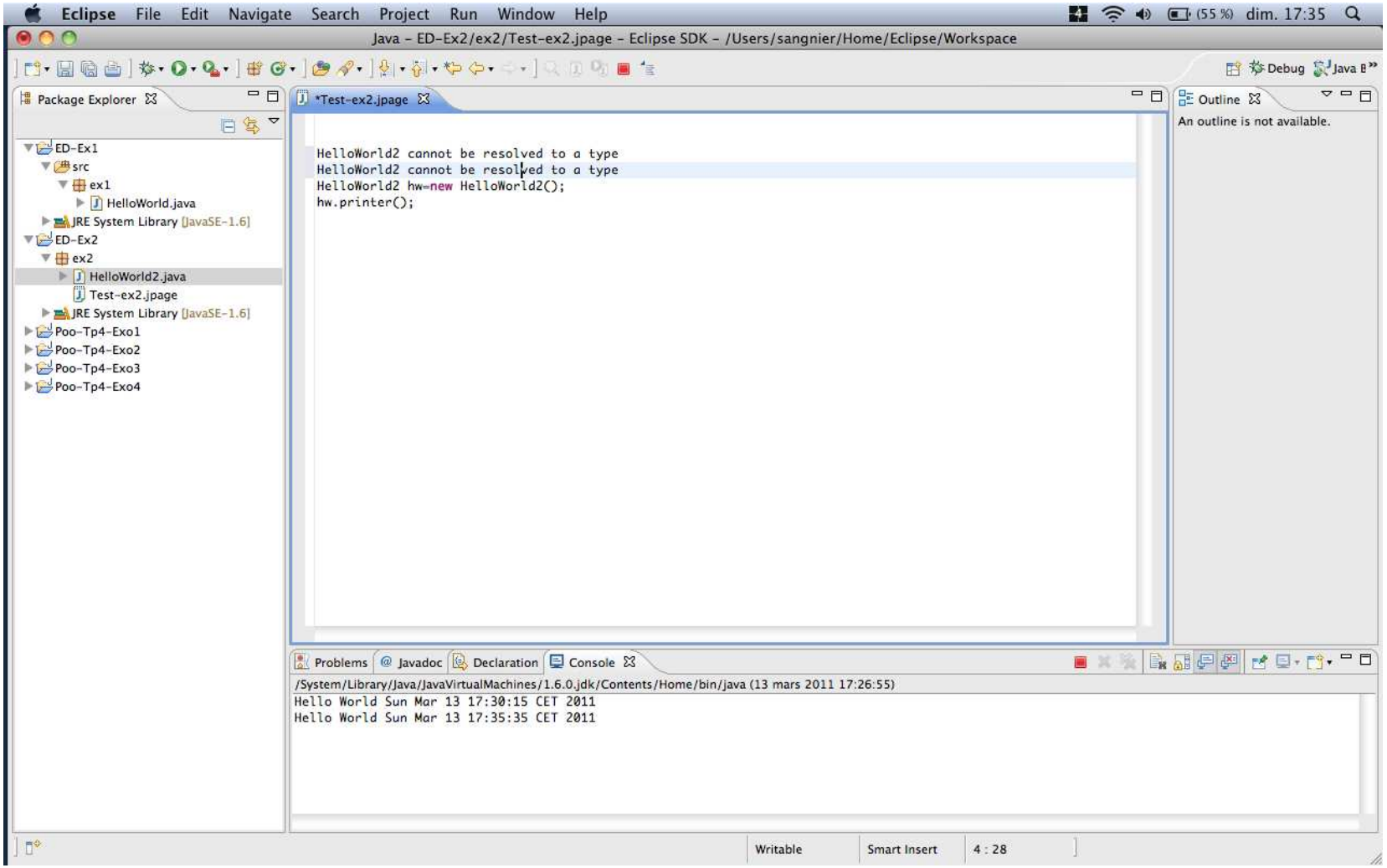
Ouvrir une page de testeur de code



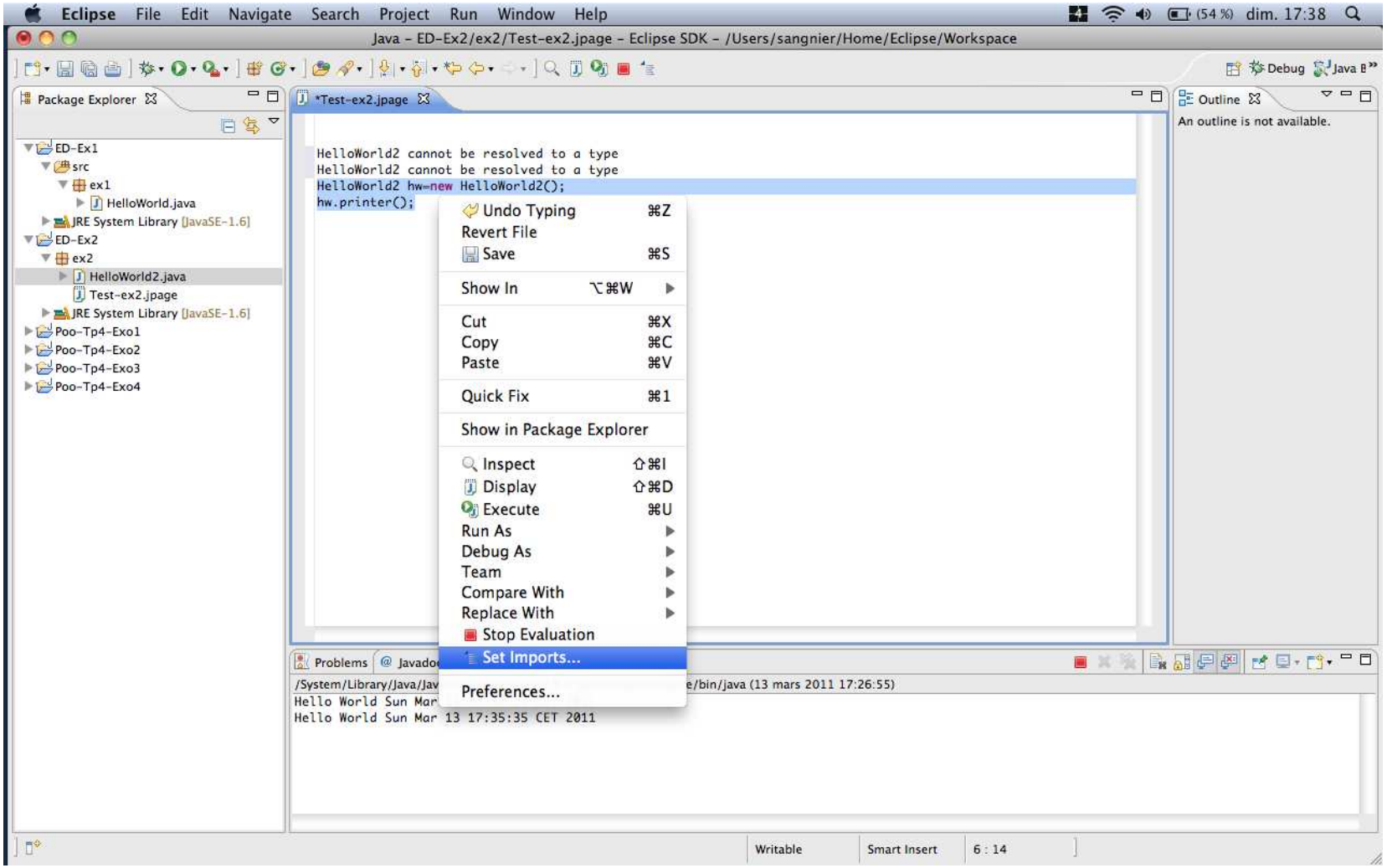
Exécuter le test sélectionné



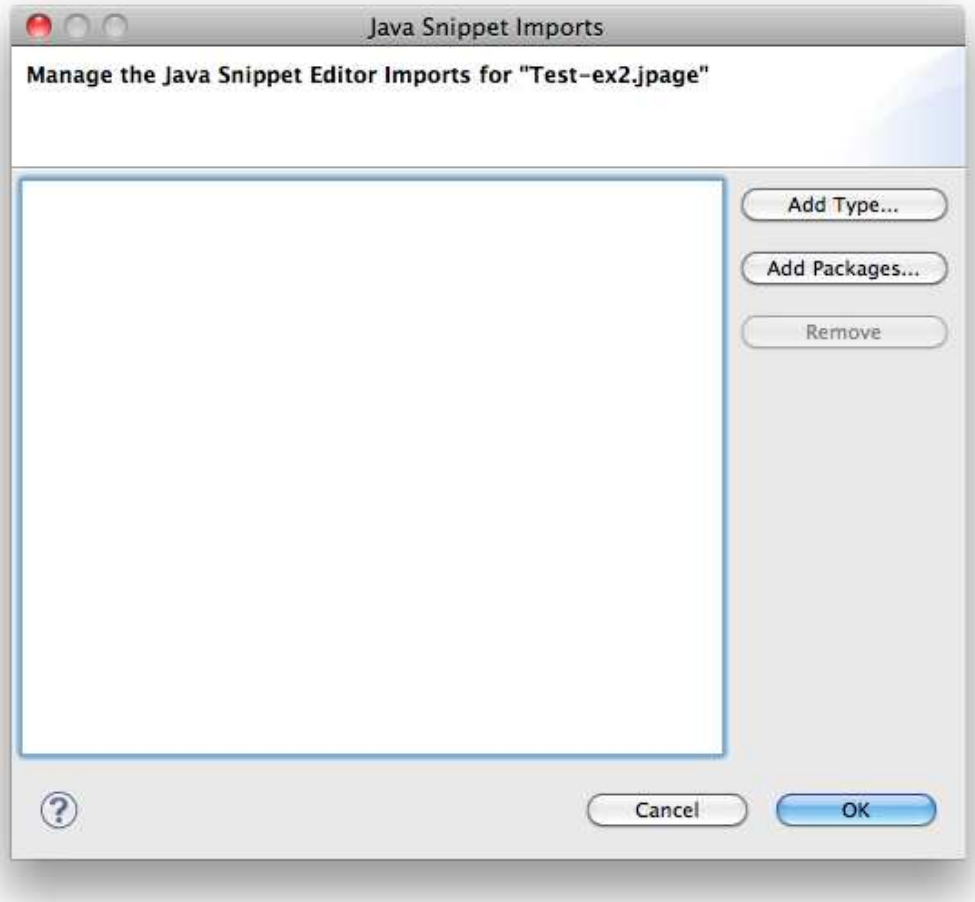
Les erreurs sont écrites dans le fichier



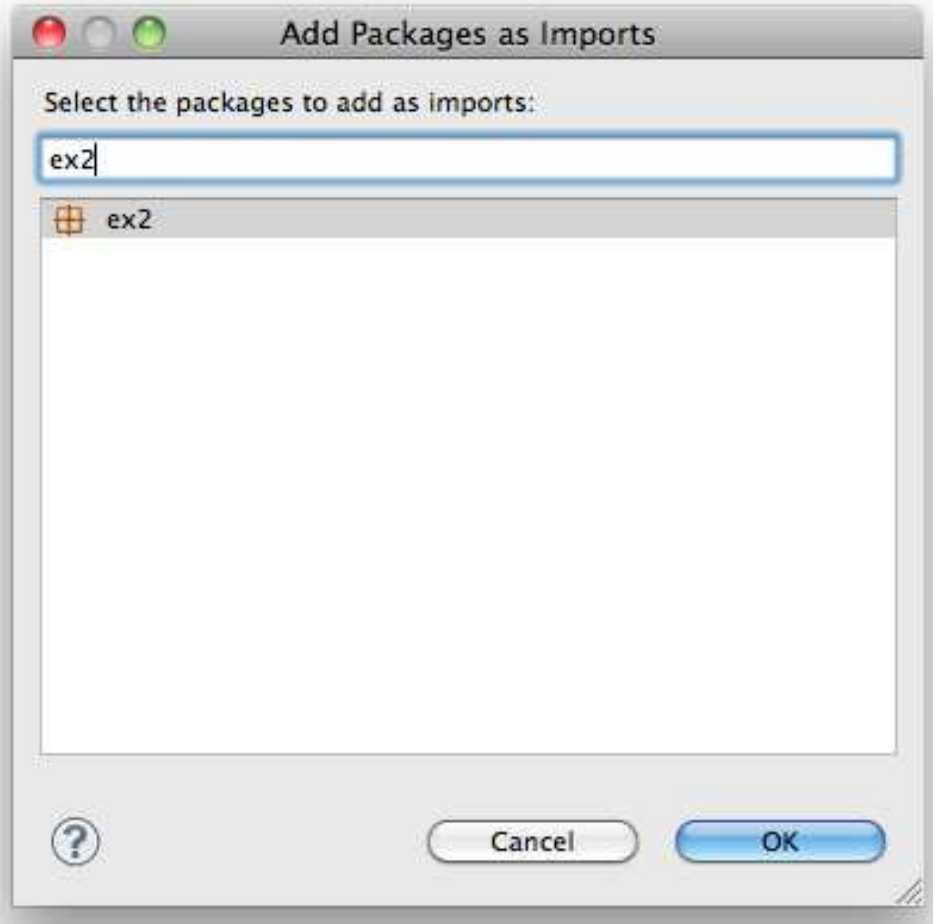
Faire les imports corrects



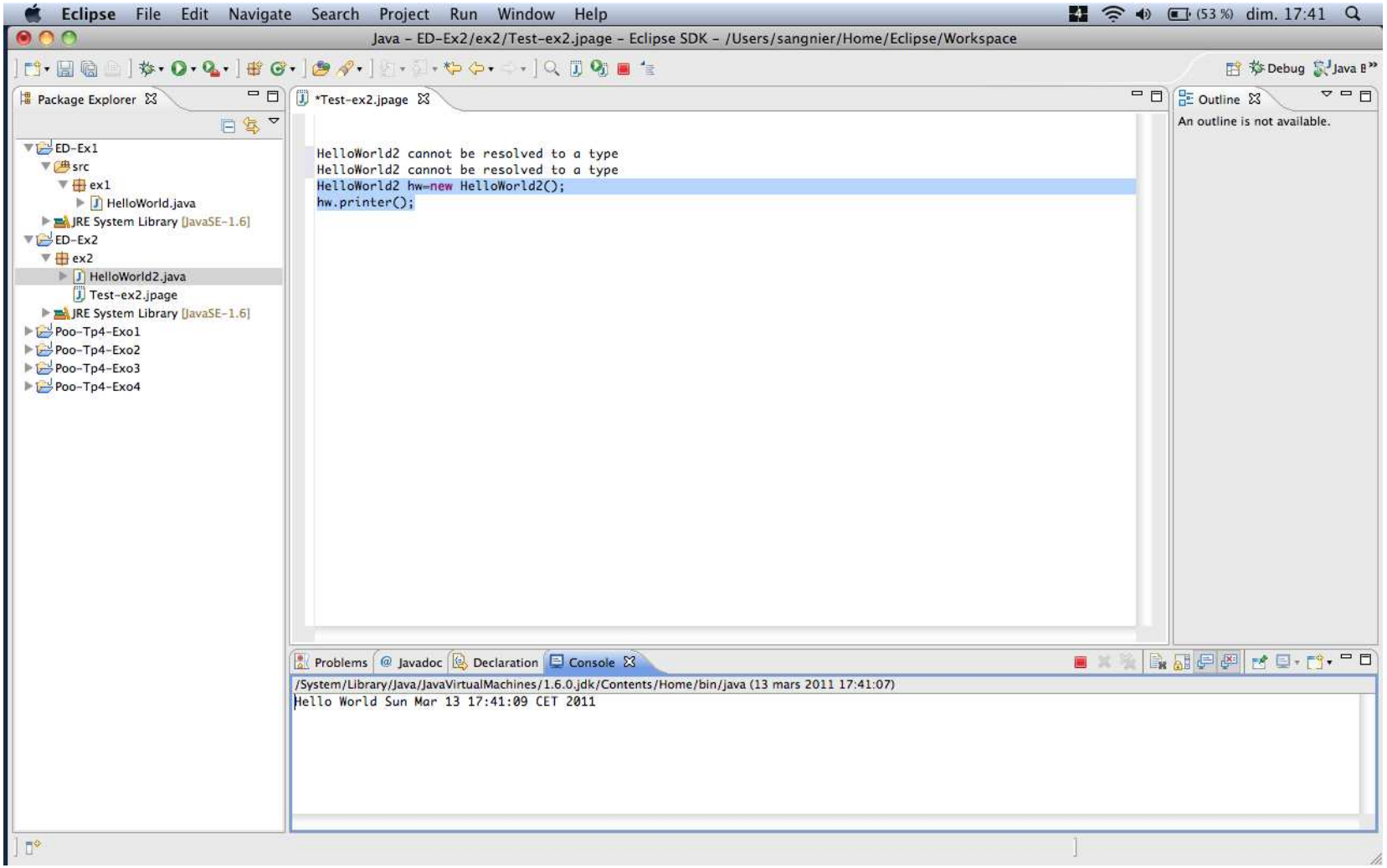
Faire les imports corrects



Faire les imports corrects

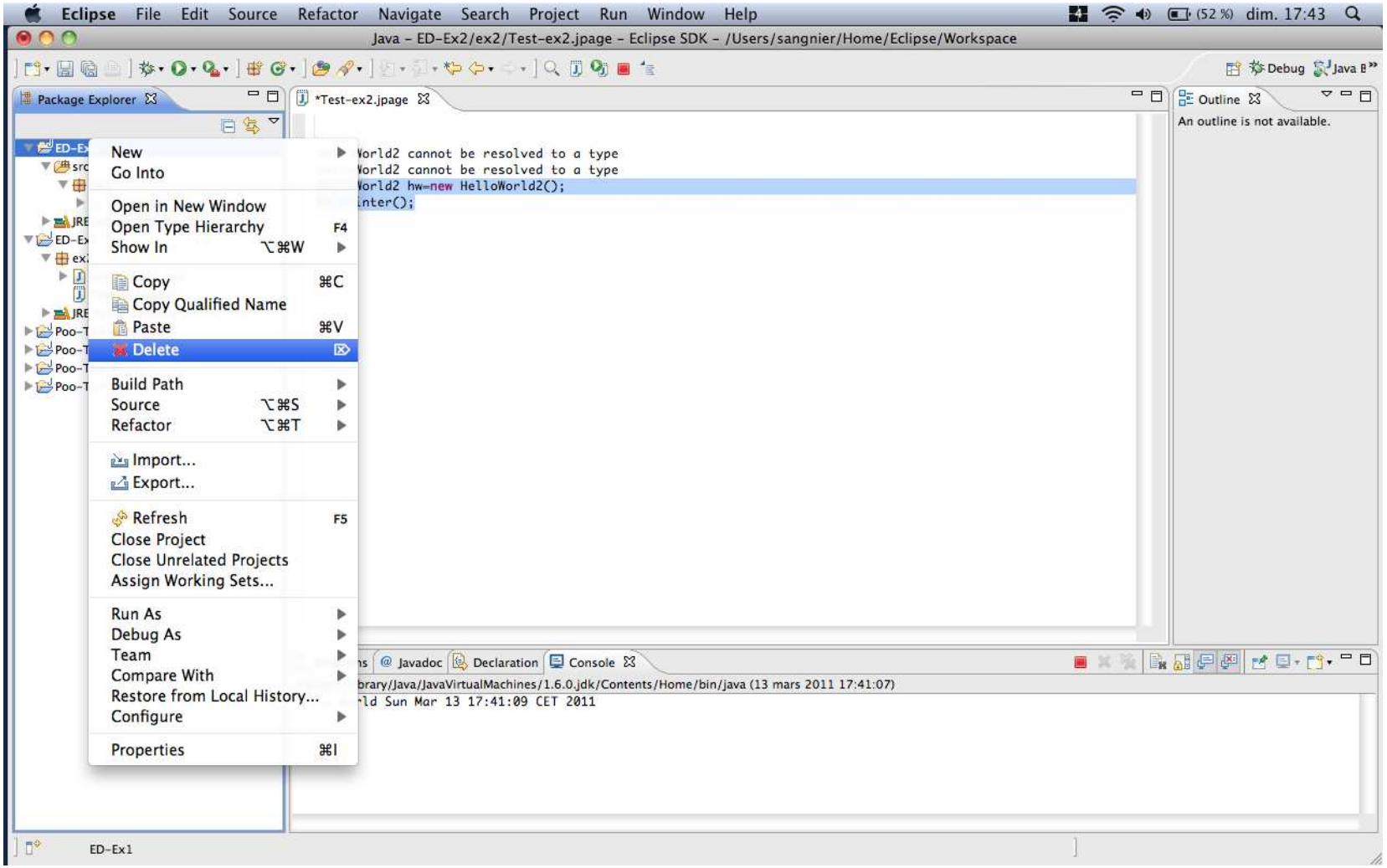


Résultat du test dans la console

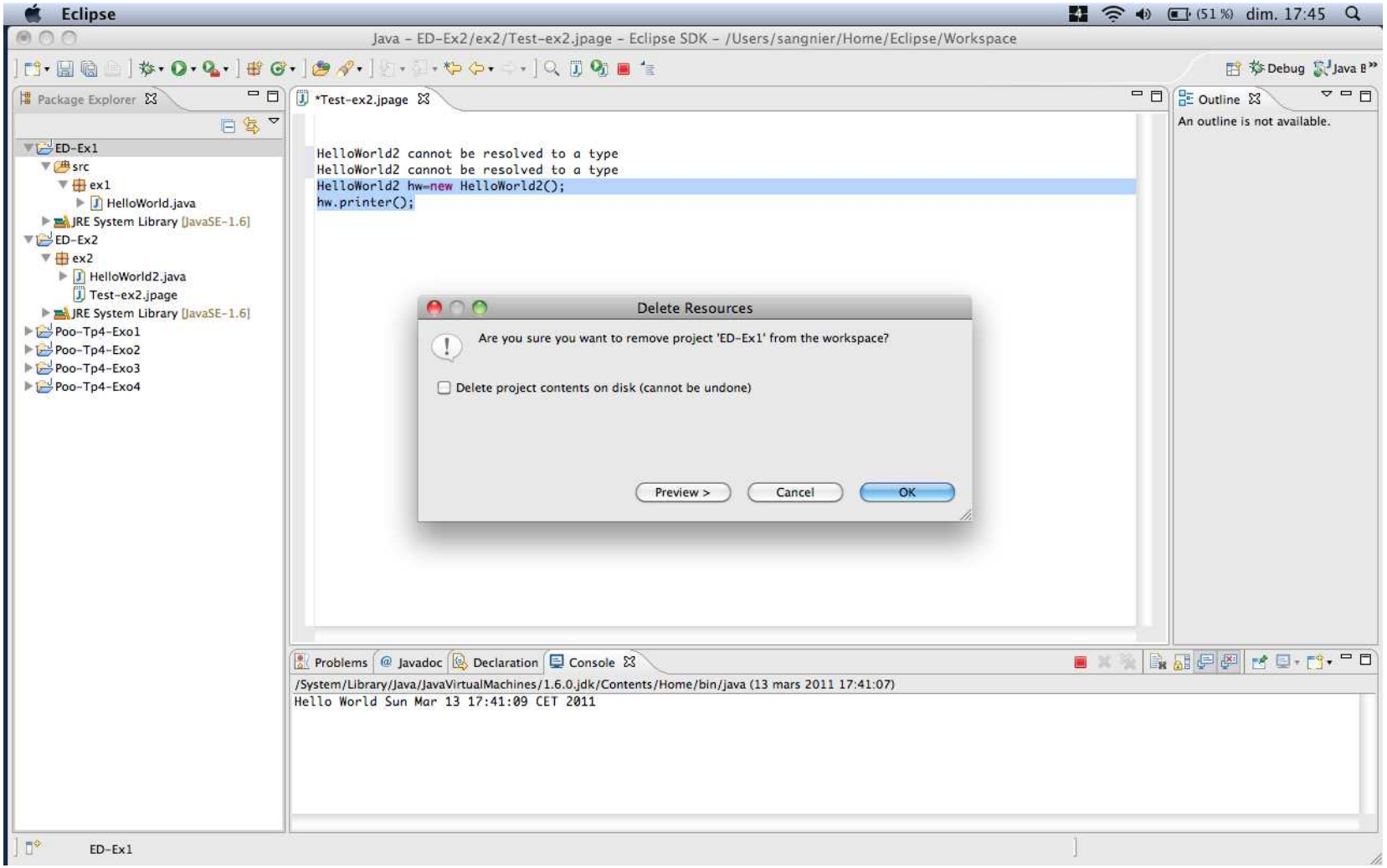


Appuyer sur le carré rouge au dessus de la console pour arrêter la phase d'exécution de la page de testeur de code

Alléger la vue package

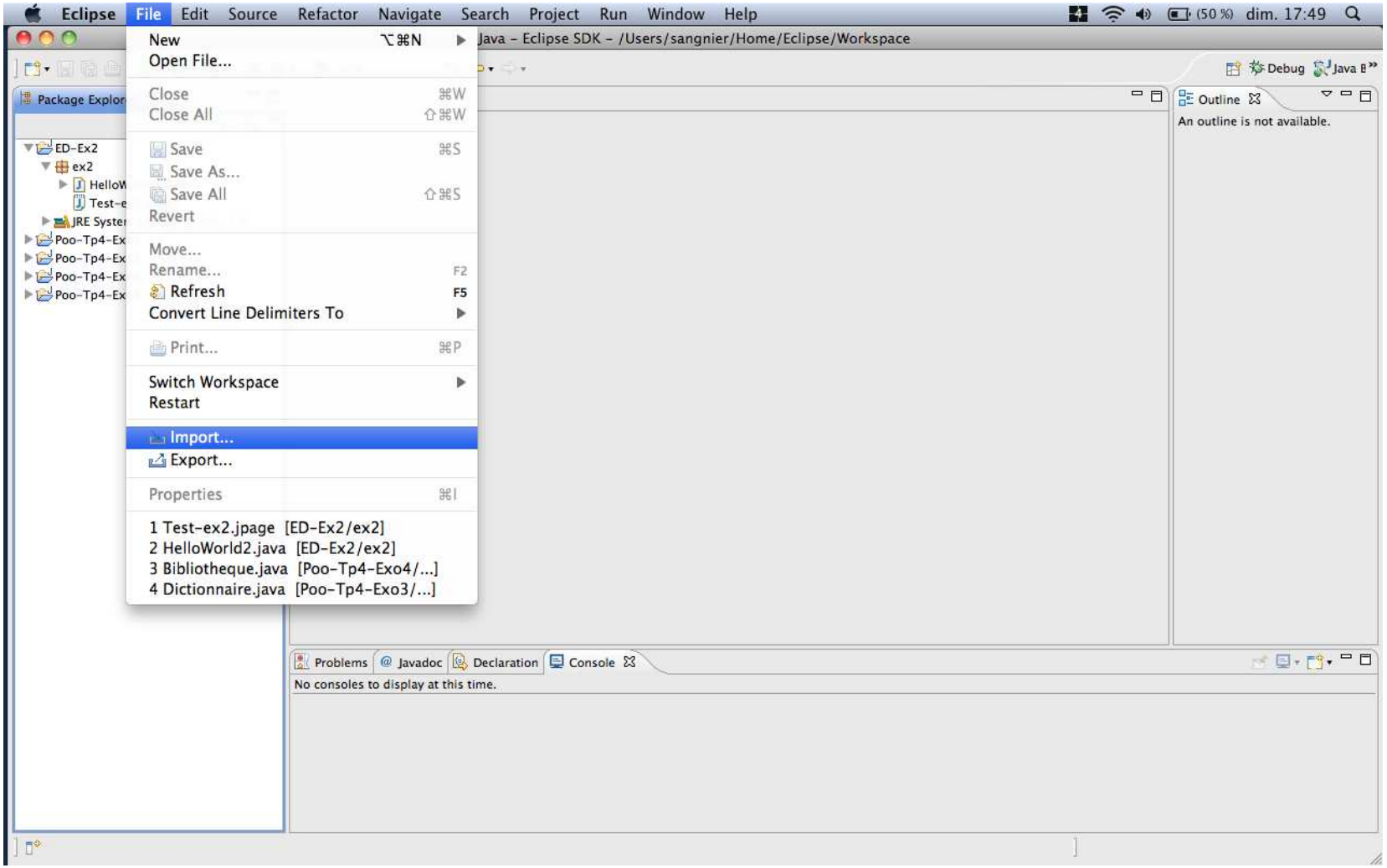


Alléger la vue package

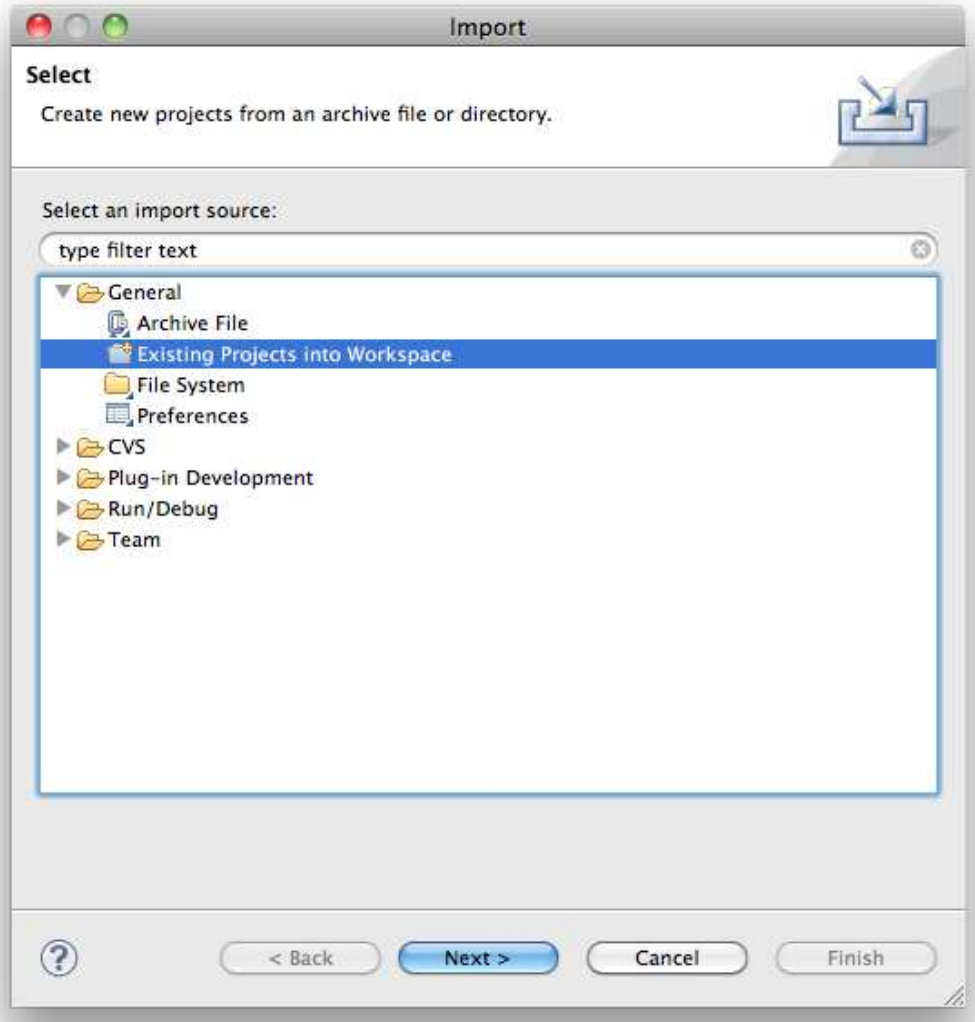


Attention : Ne pas effacer votre projet du disque si vous voulez encore le garder

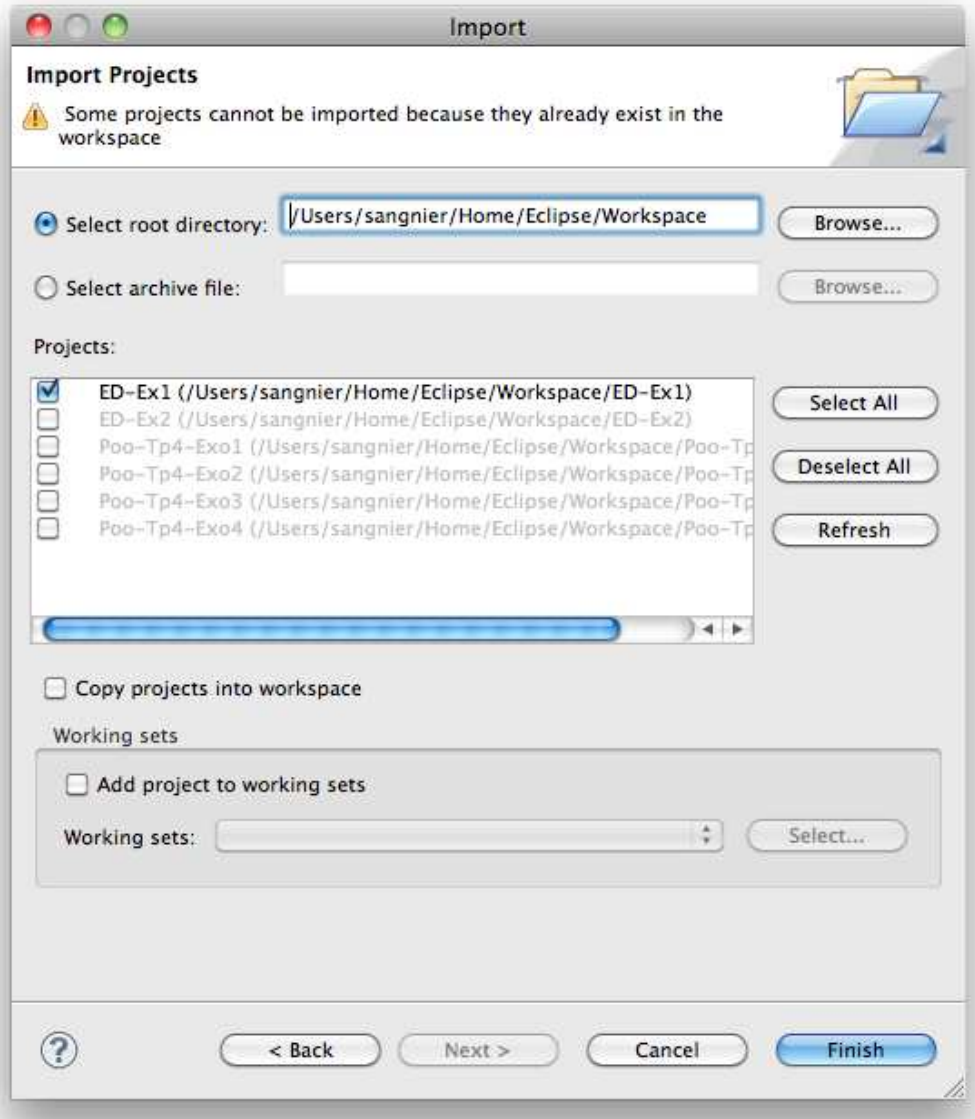
Réouvrir un projet fermé



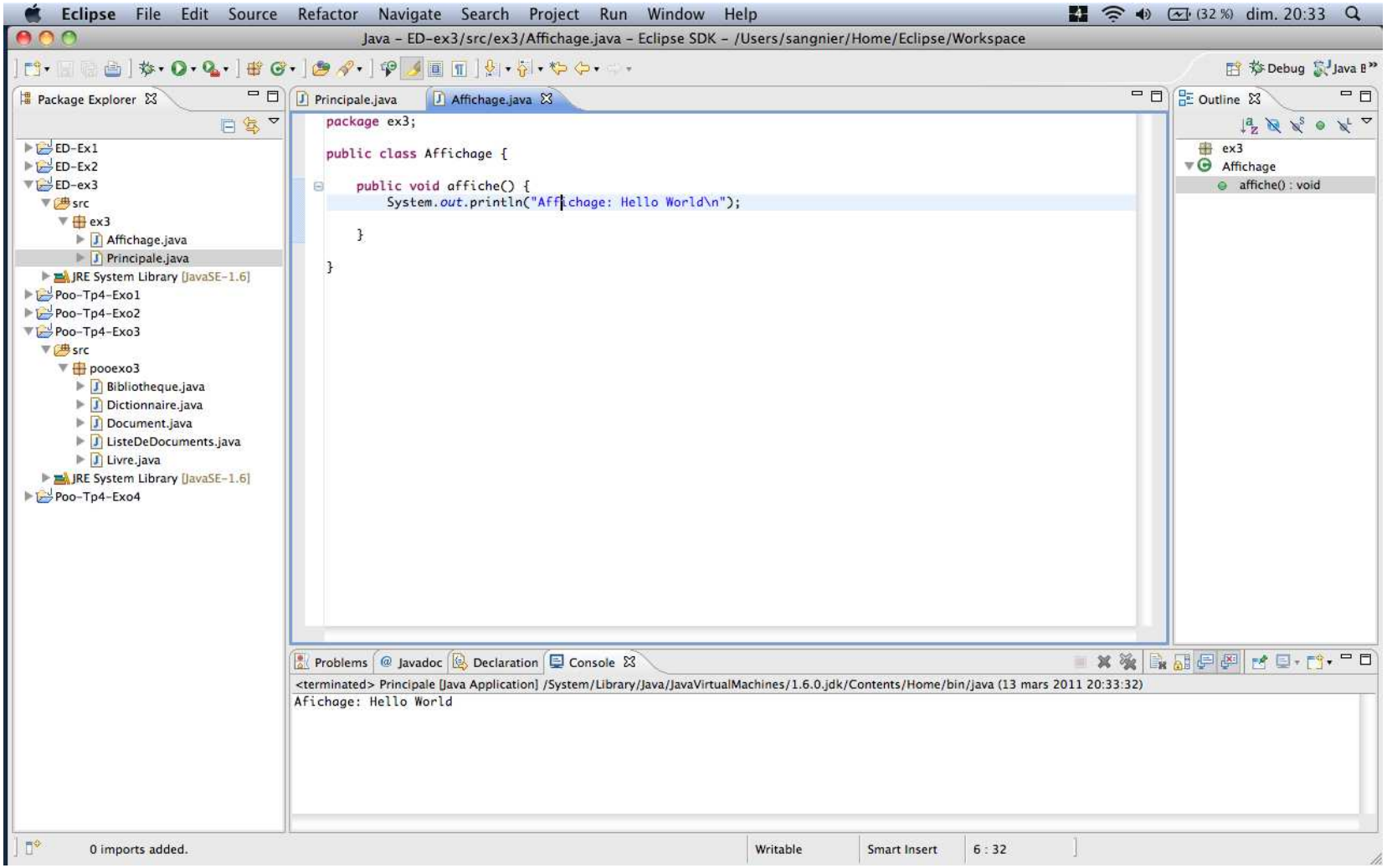
Réouvrir un projet fermé



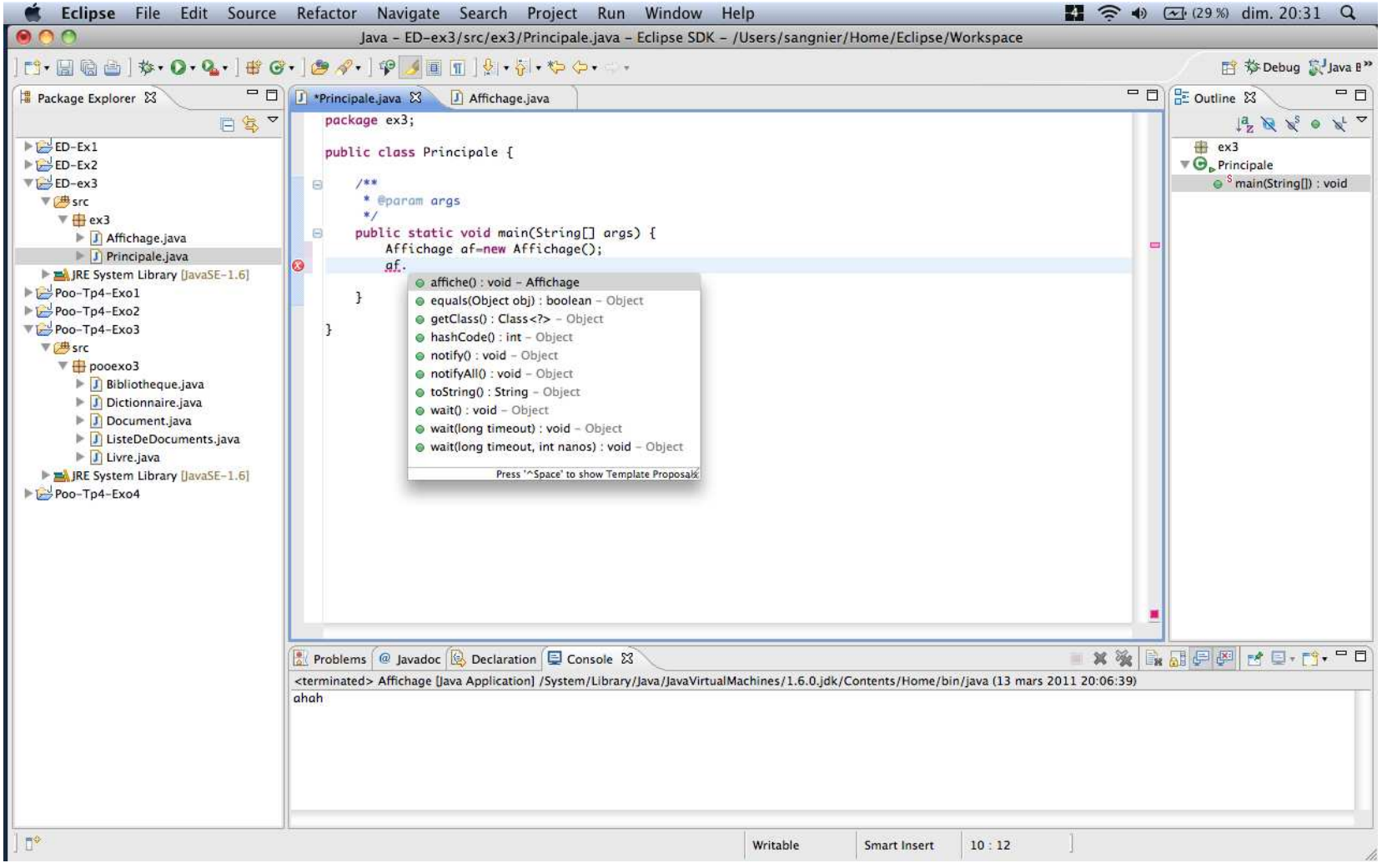
Réouvrir un projet fermé



Plusieurs classes dans un même package

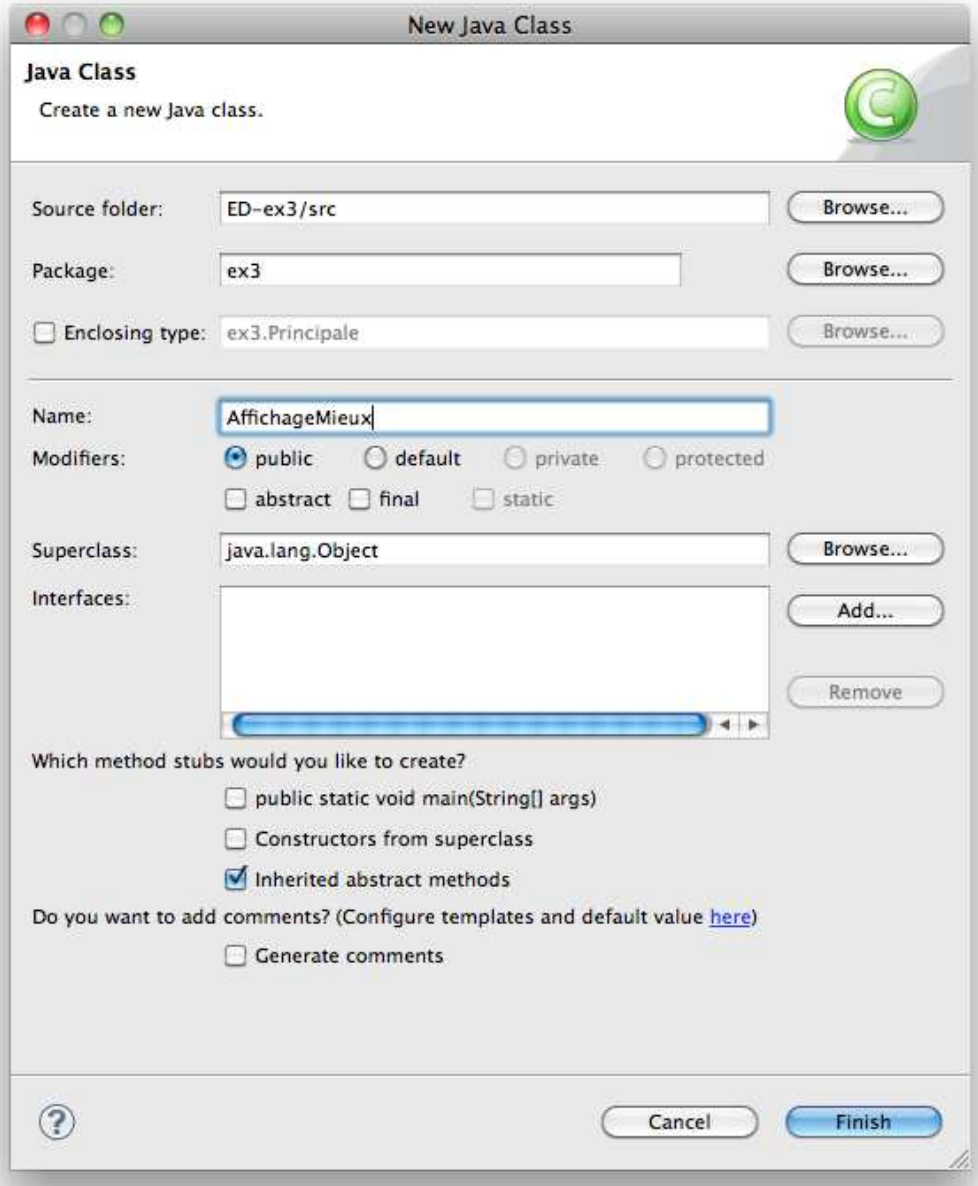


Plusieurs classes dans un même package

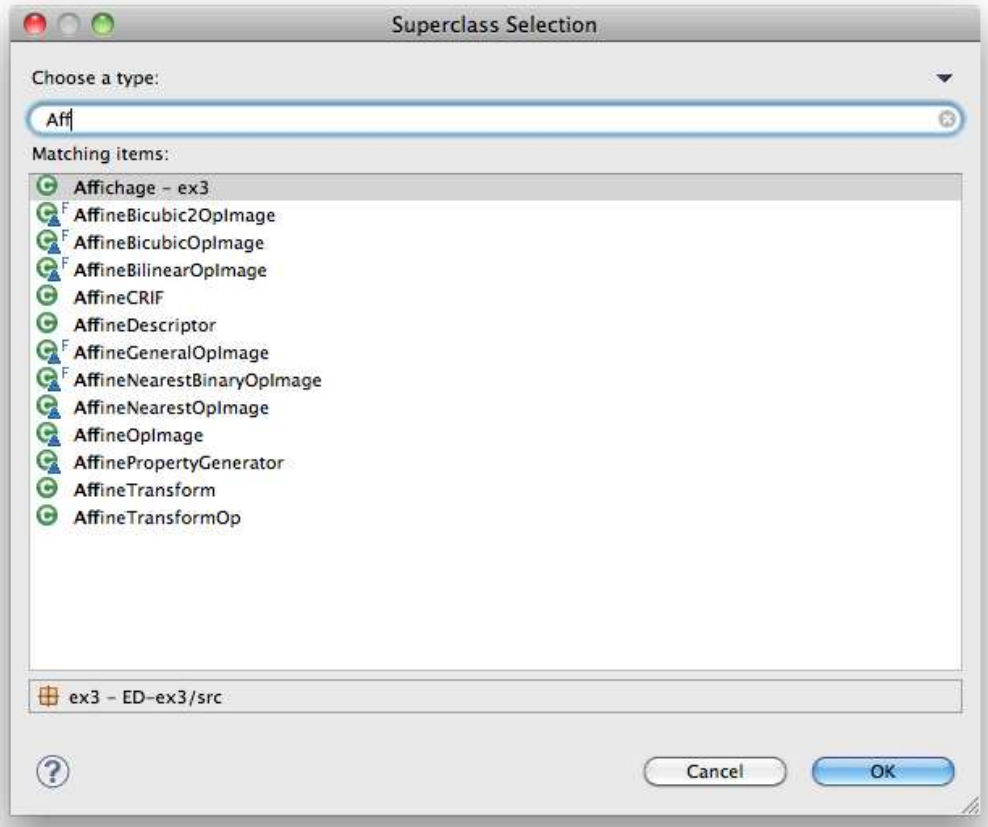


Pour que cela marche, il faut bien sauvegarder la classe Affichage

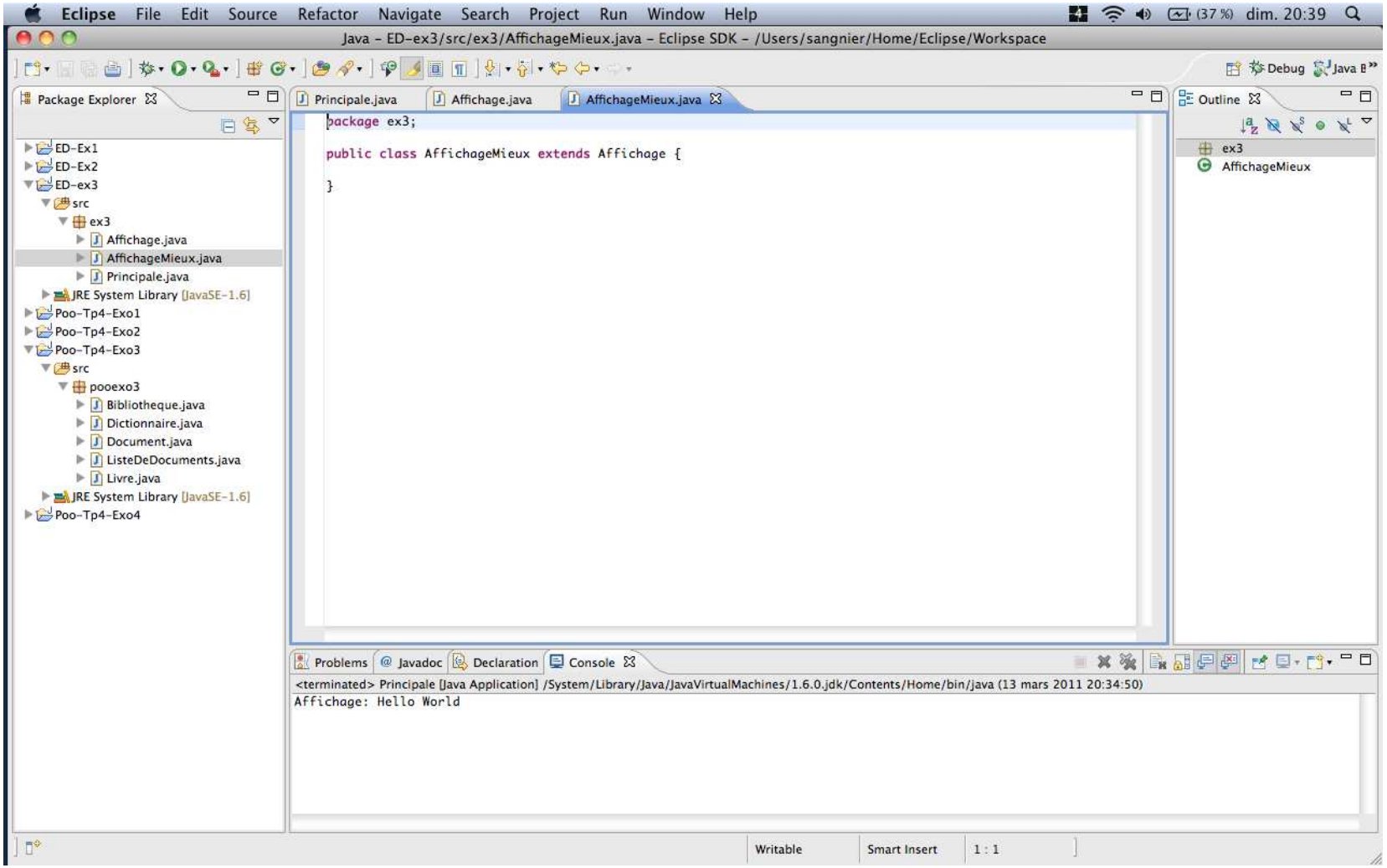
Et pour l'héritage



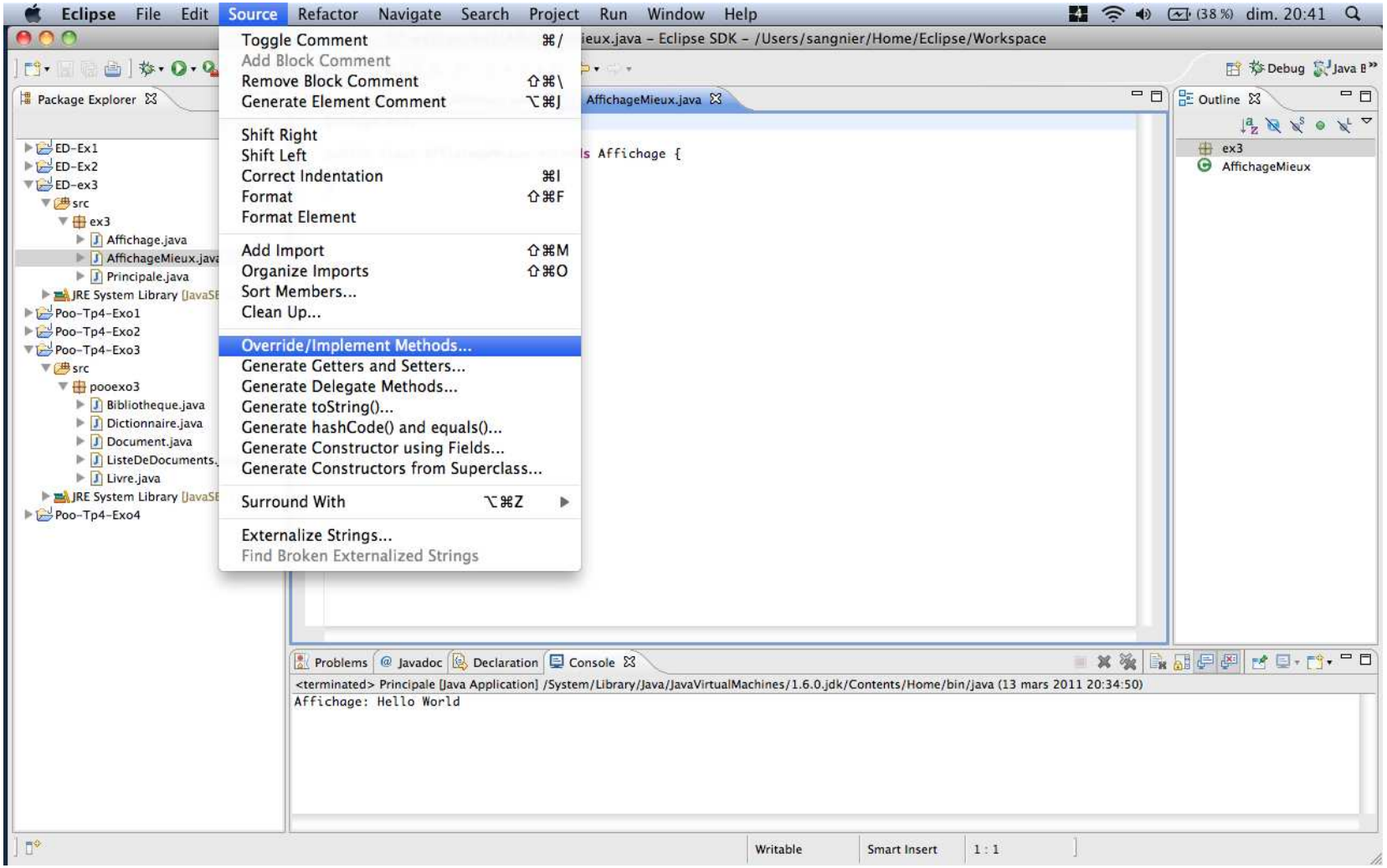
Et pour l'héritage



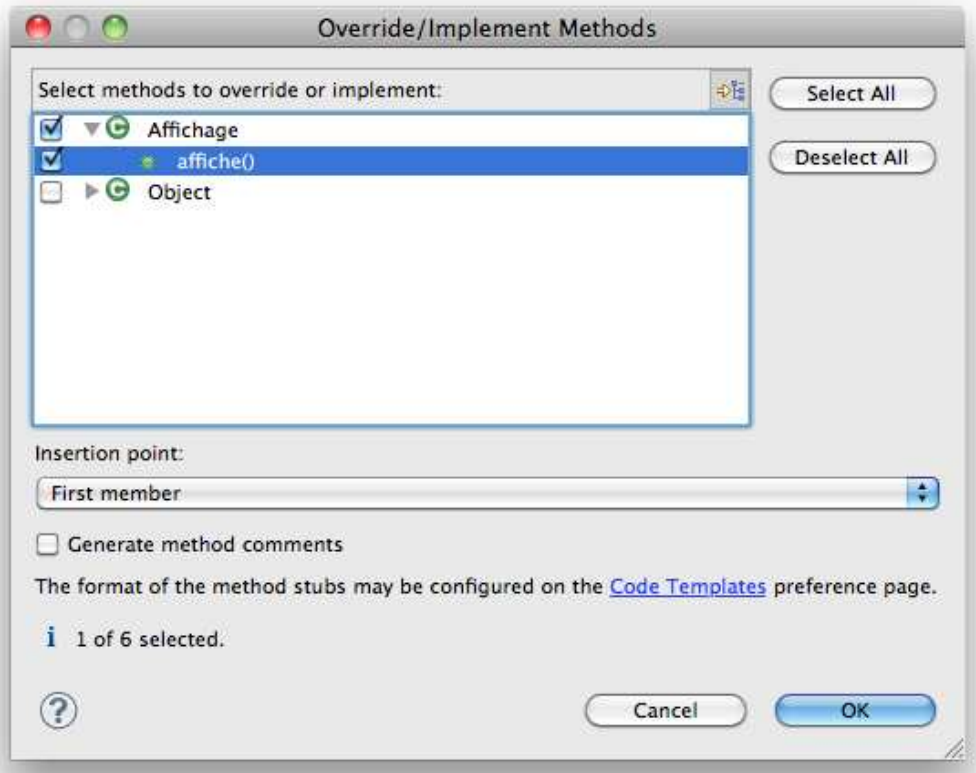
Et pour l'héritage



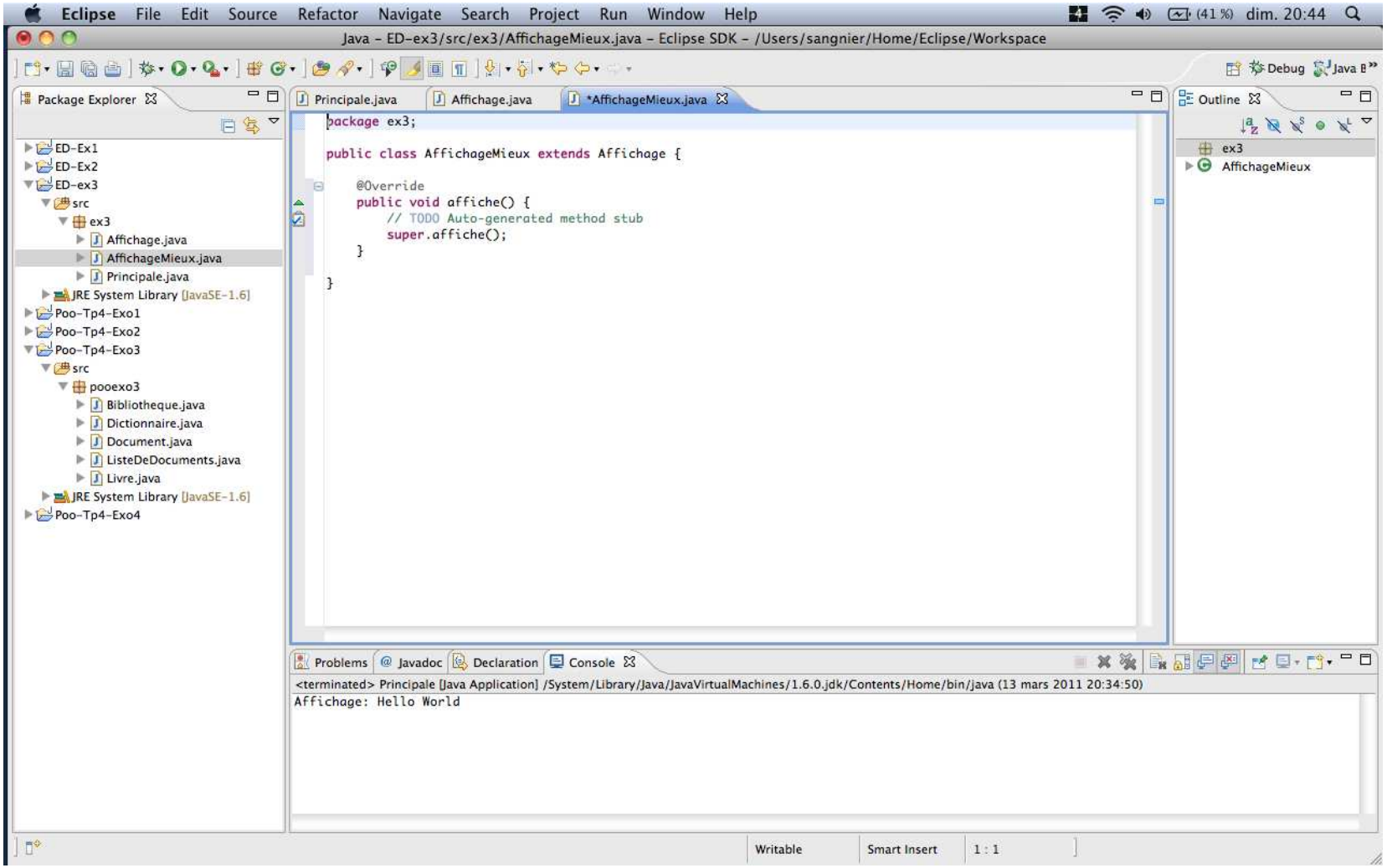
Surcharge de méthodes



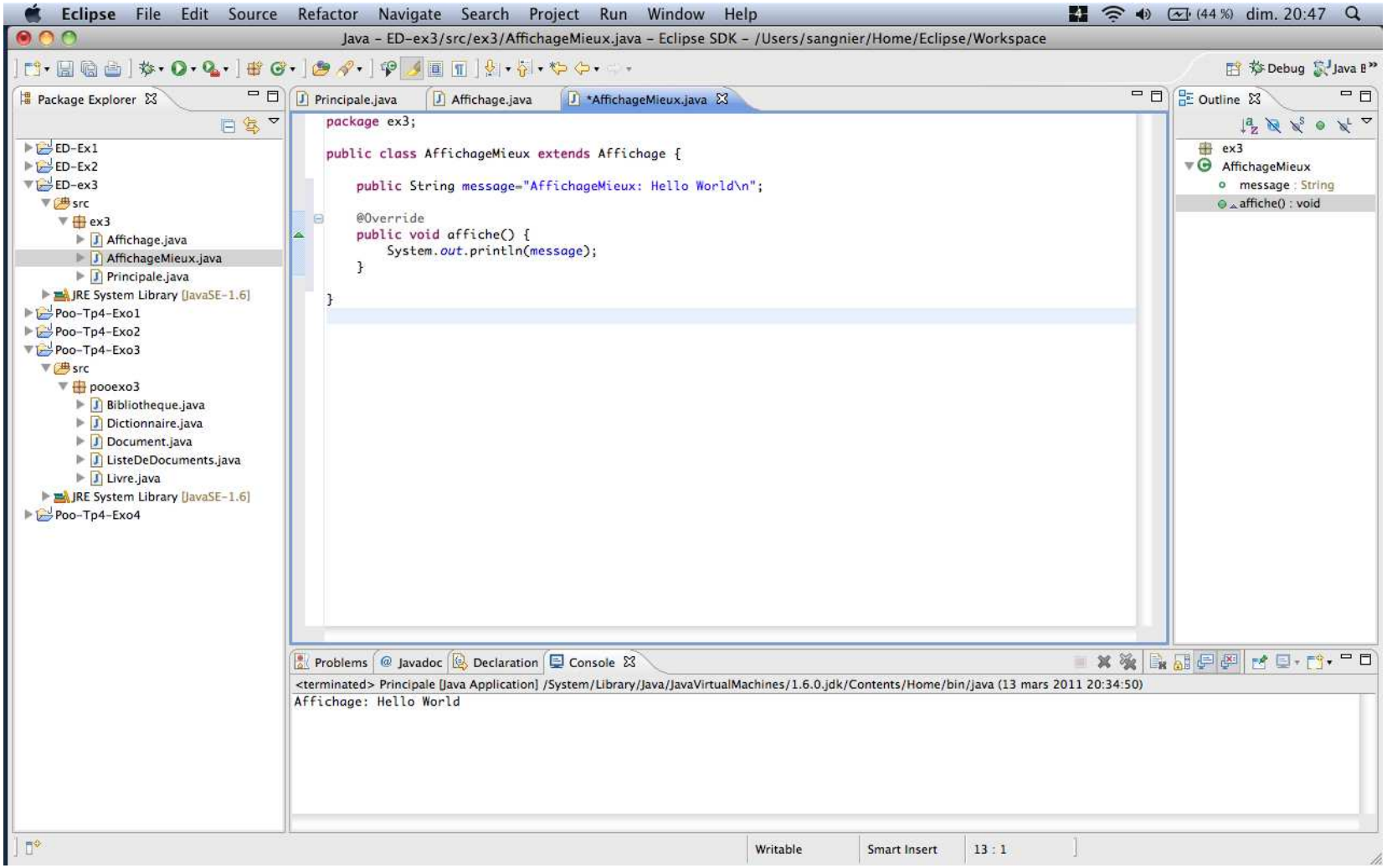
Surcharge de méthodes



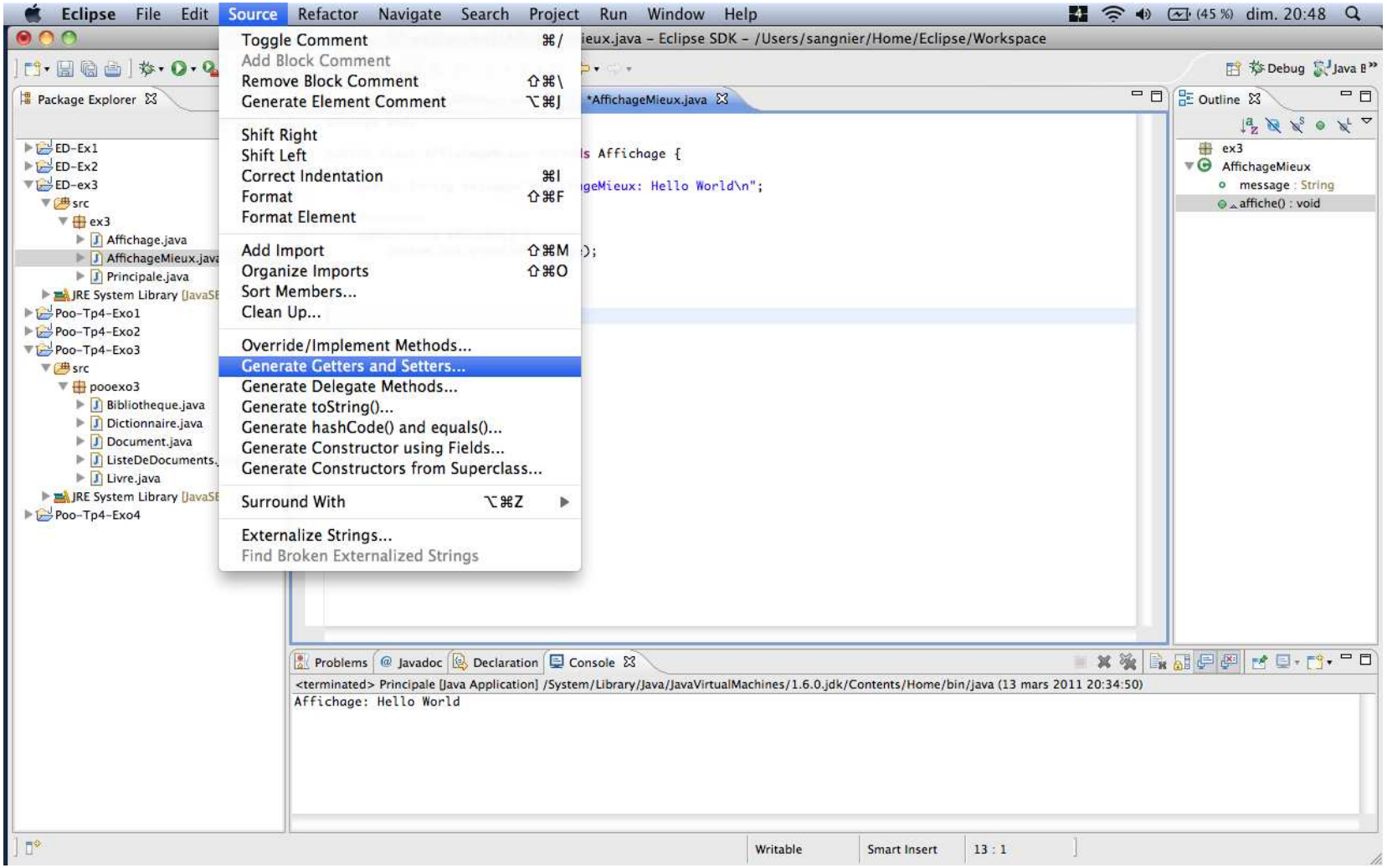
Surcharge de méthodes



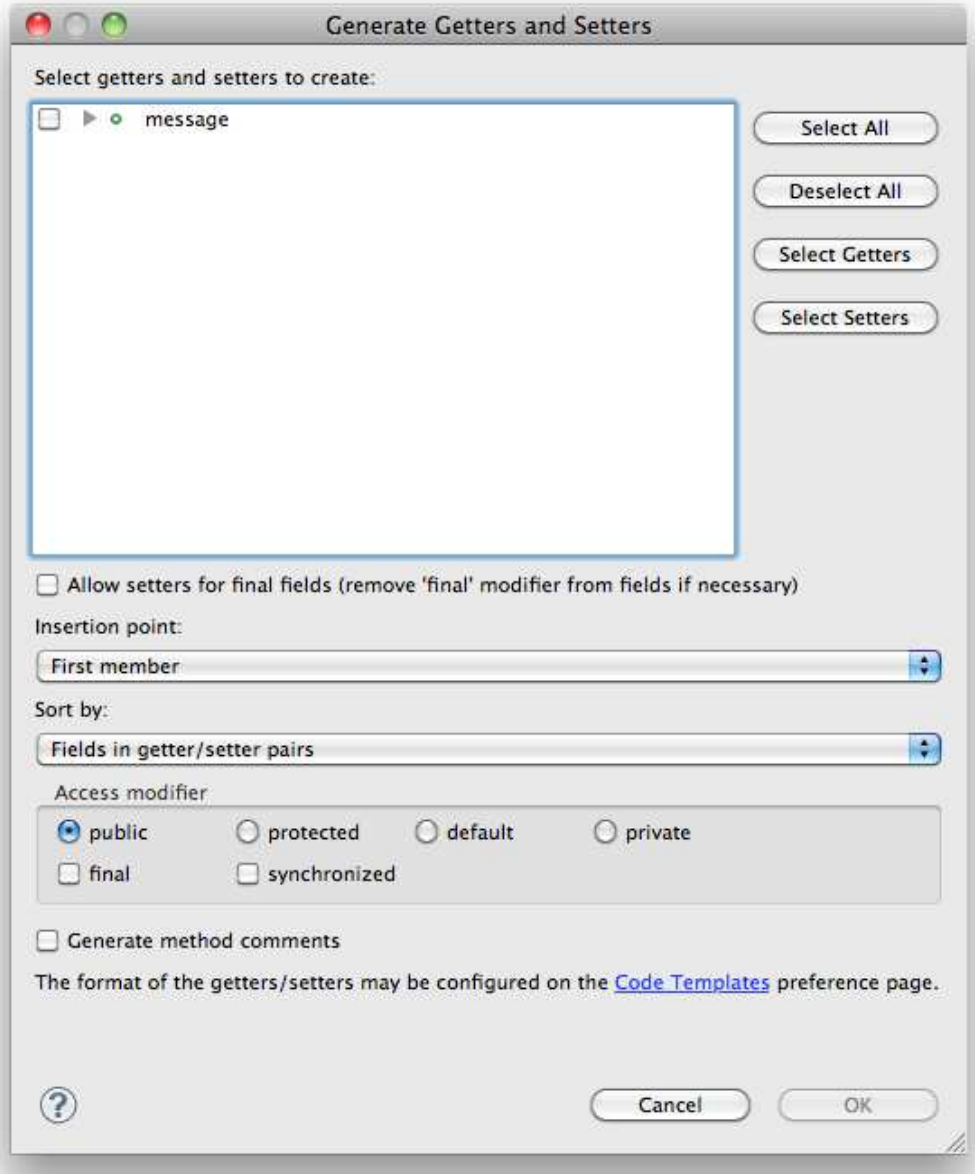
Générer des getters and setters



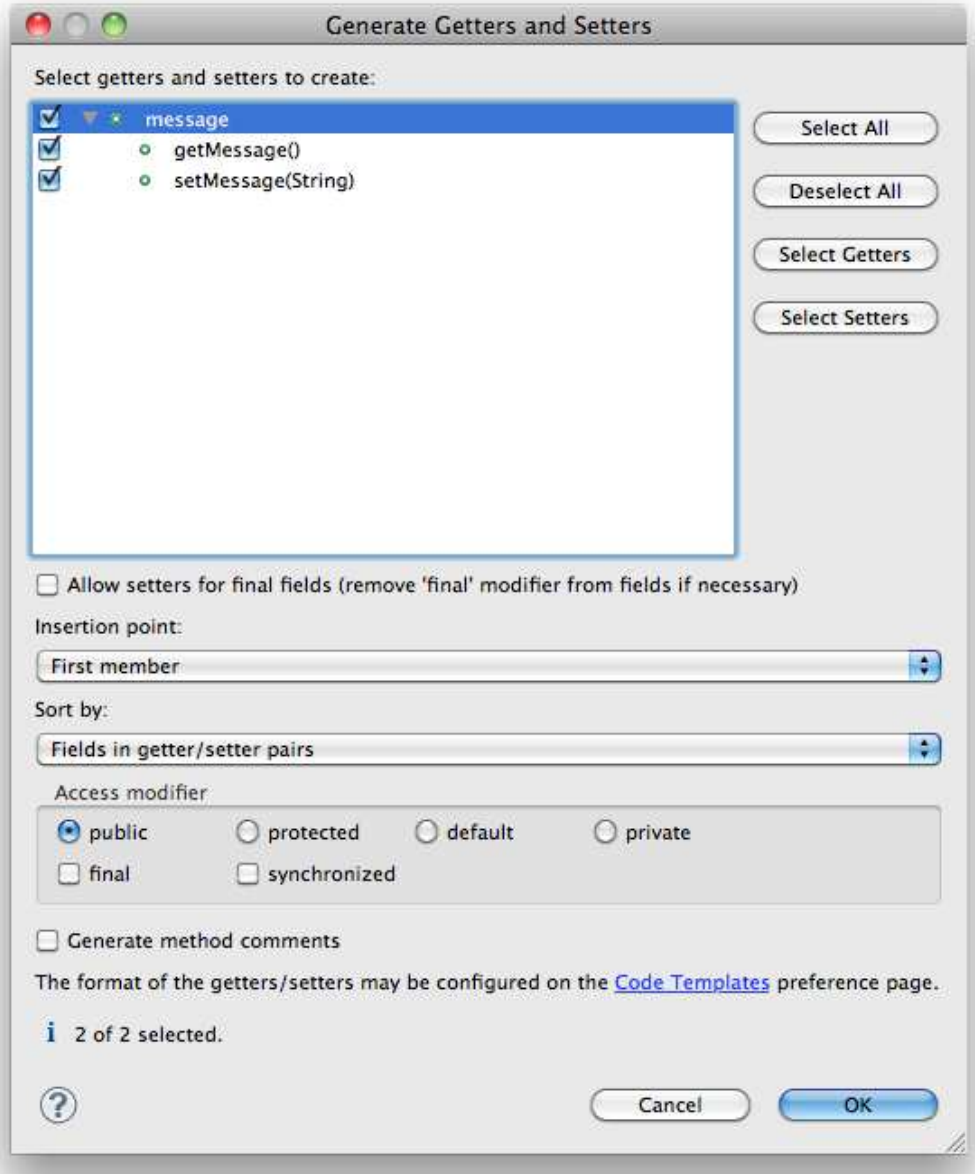
Générer des getters and setters



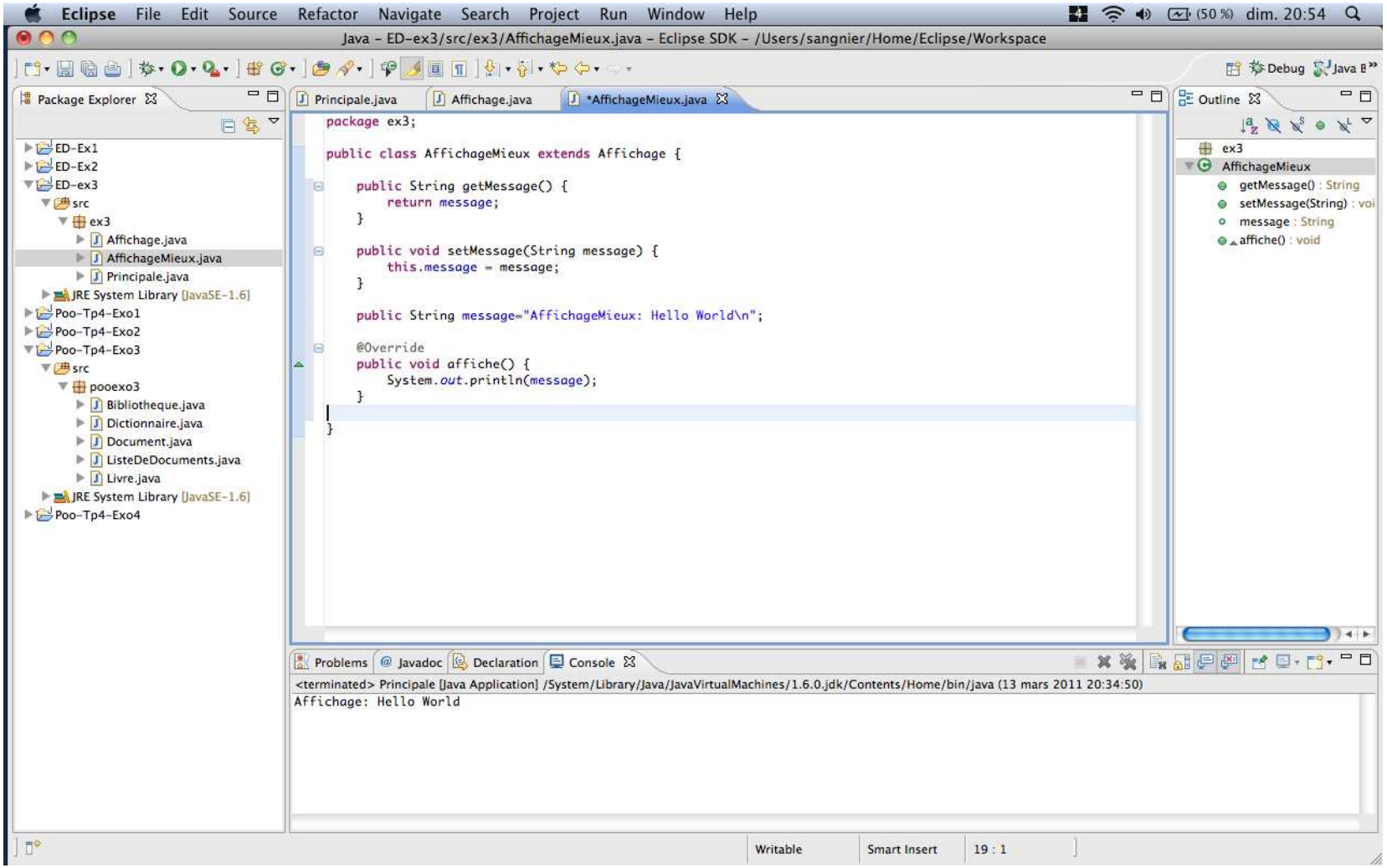
Générer des getters and setters



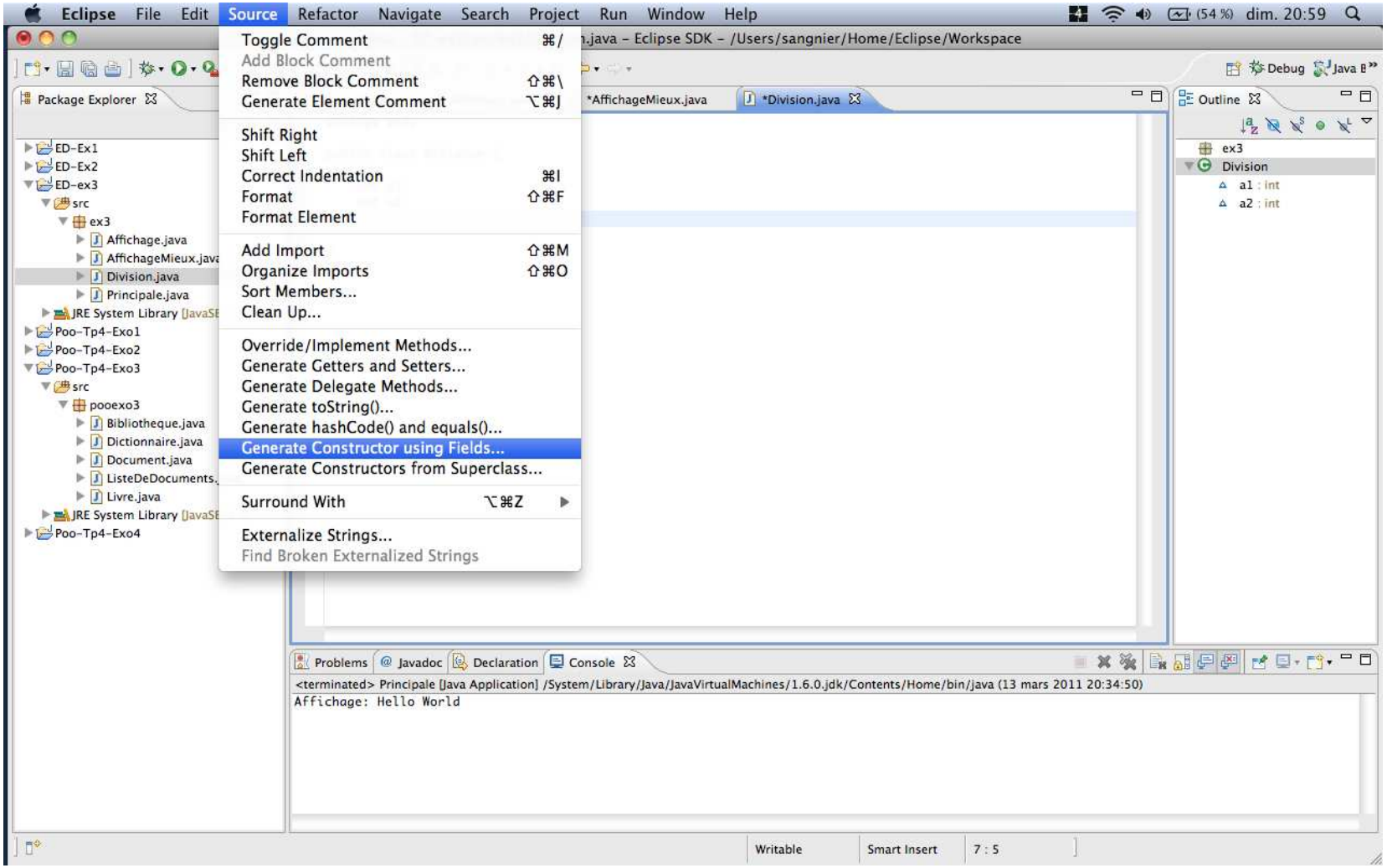
Générer des getters and setters



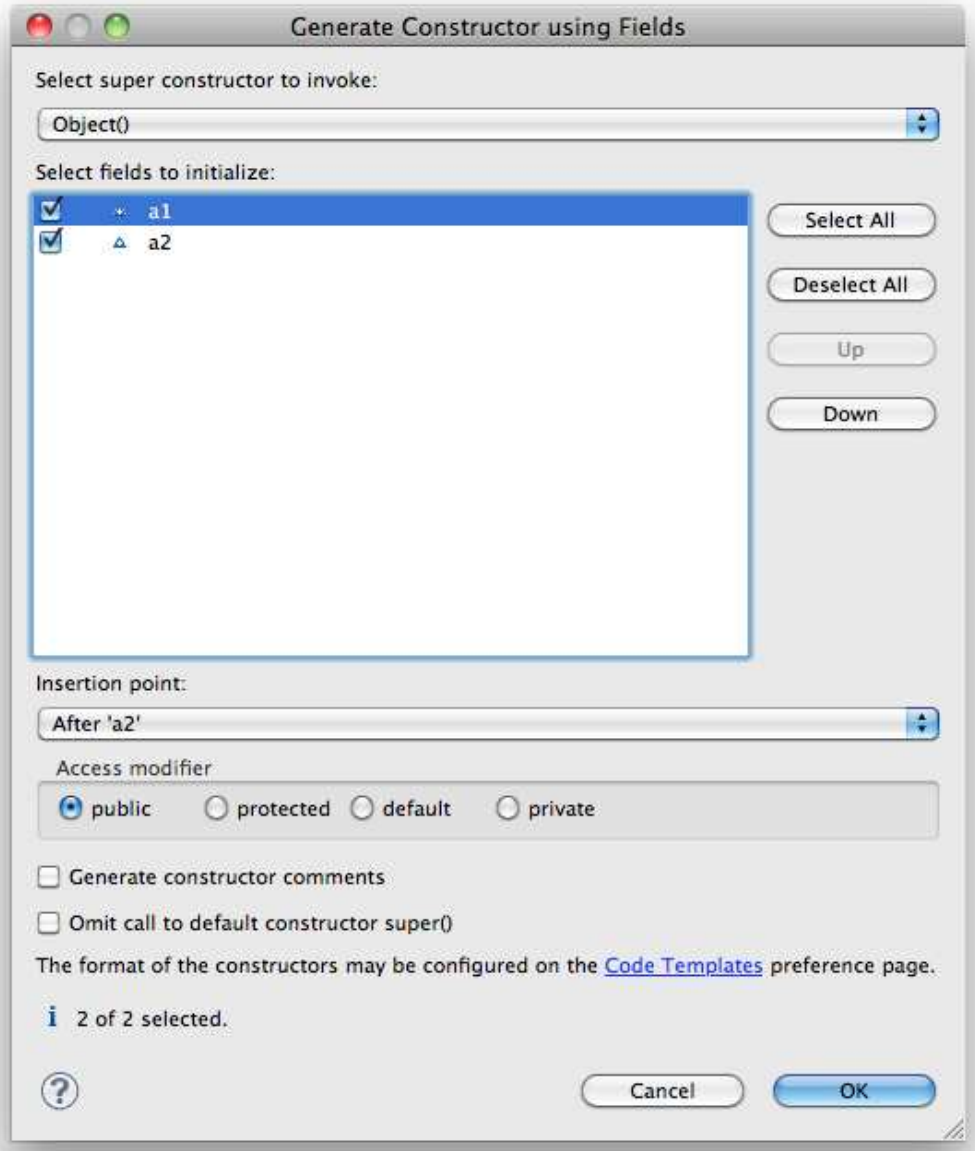
Générer des getters and setters



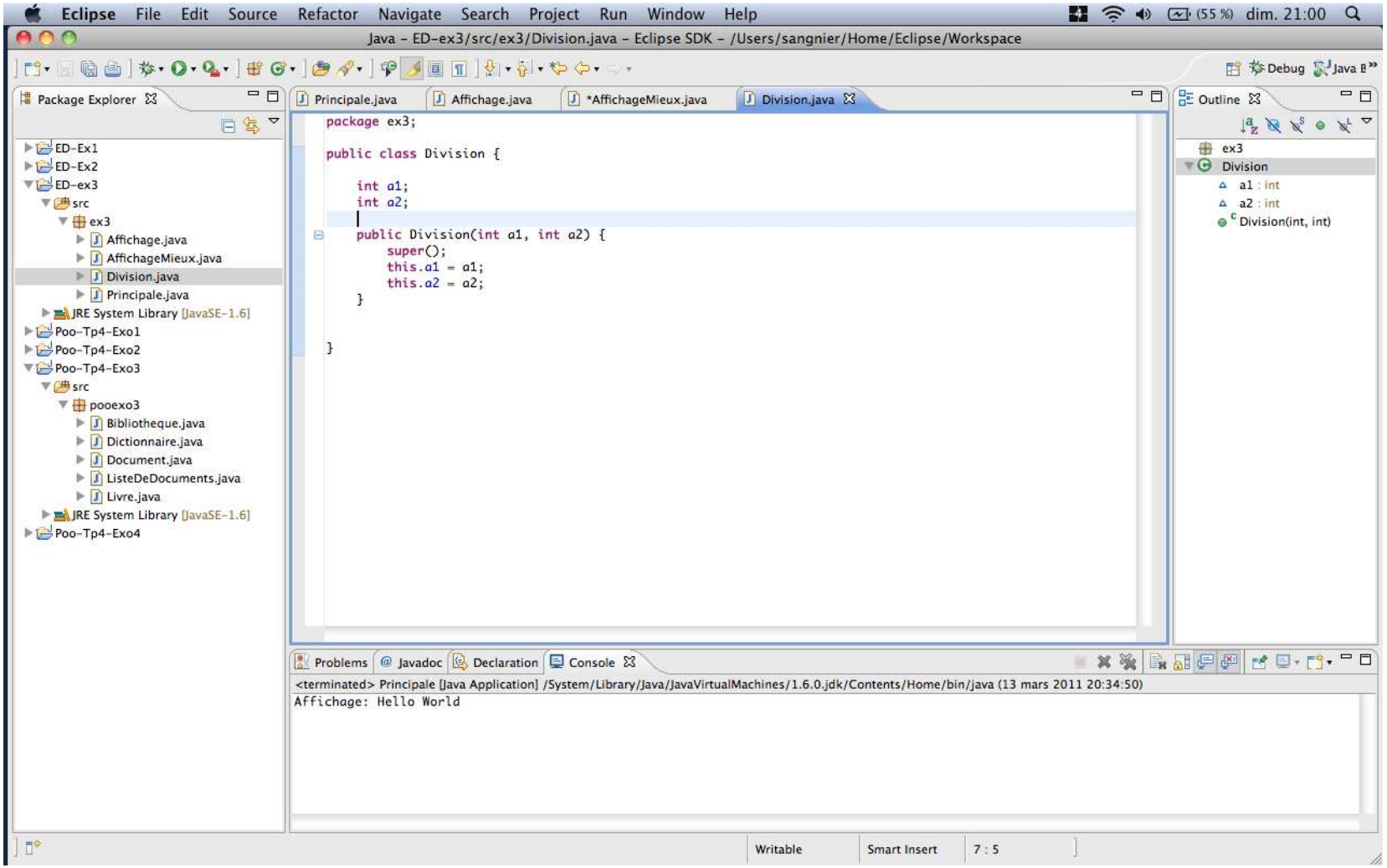
Générer un constructeur



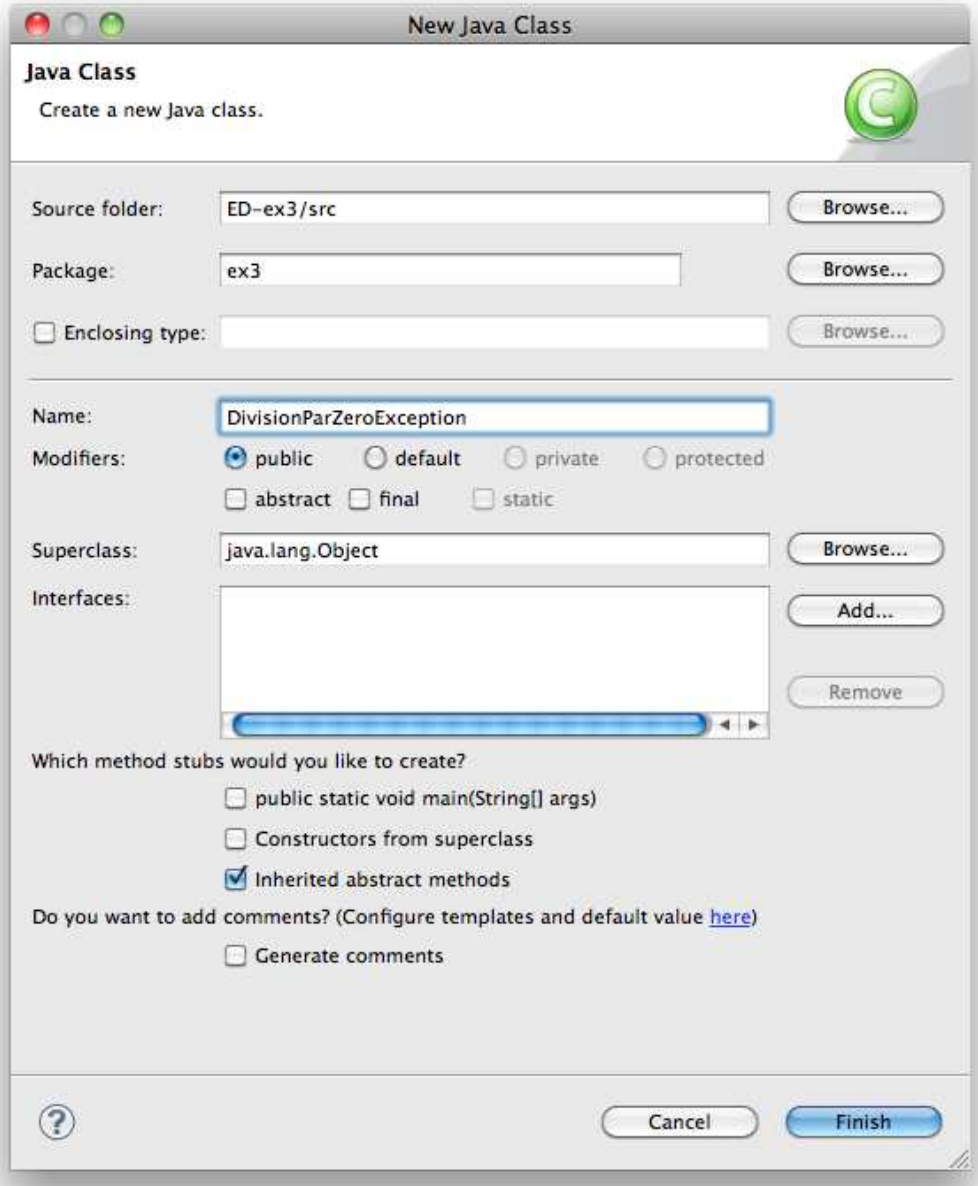
Générer un constructeur



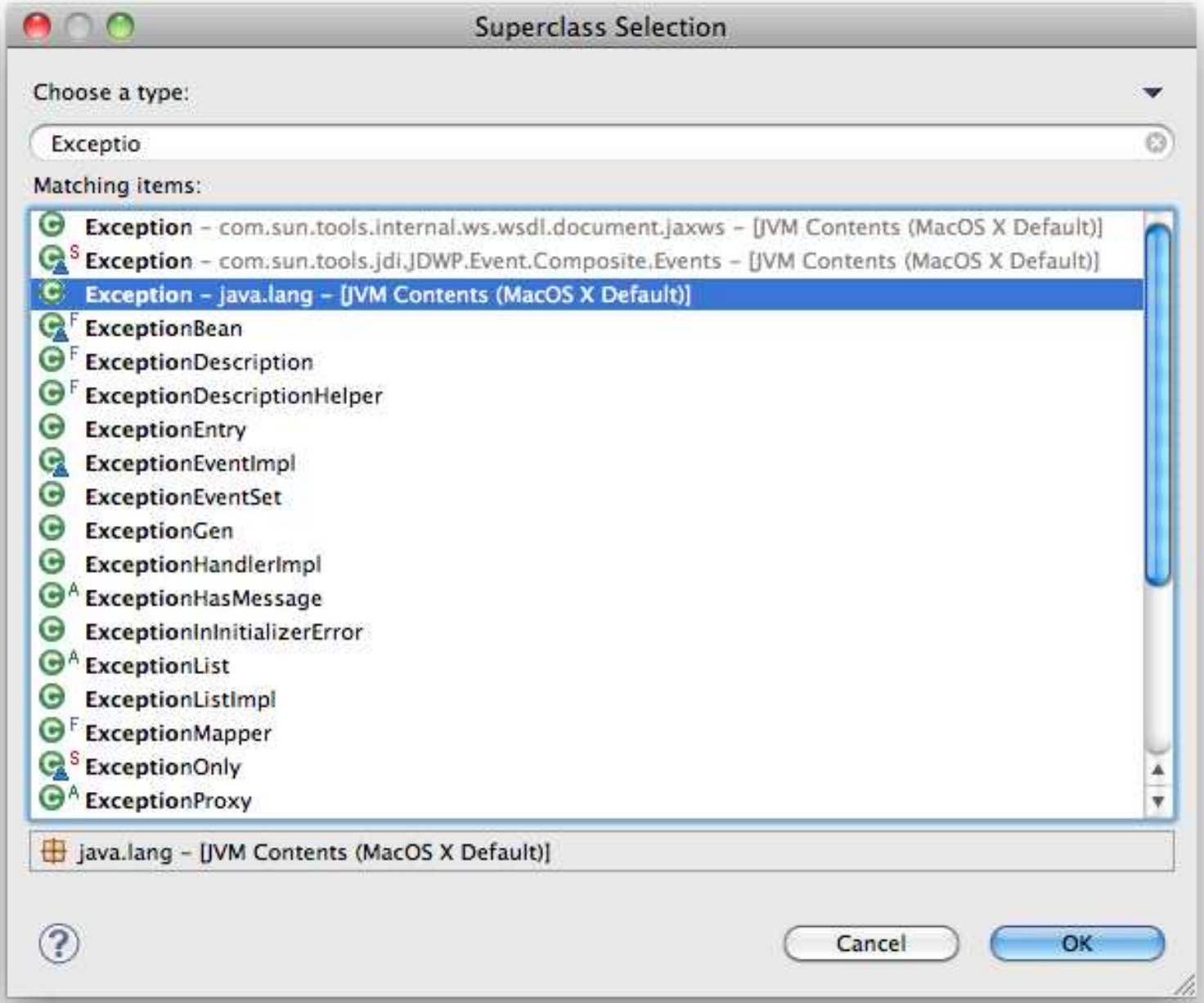
Générer un constructeur



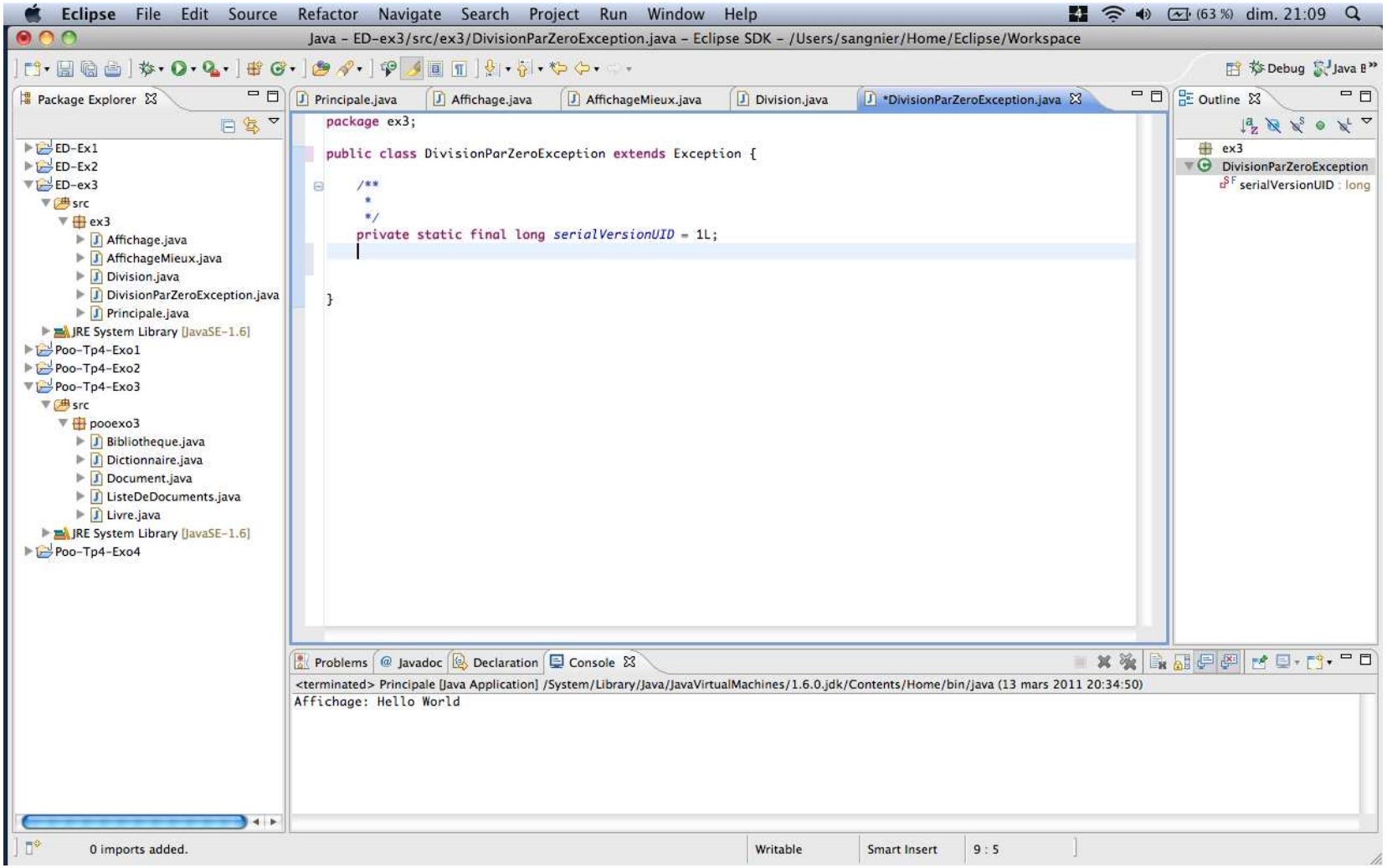
Et pour les exceptions ?



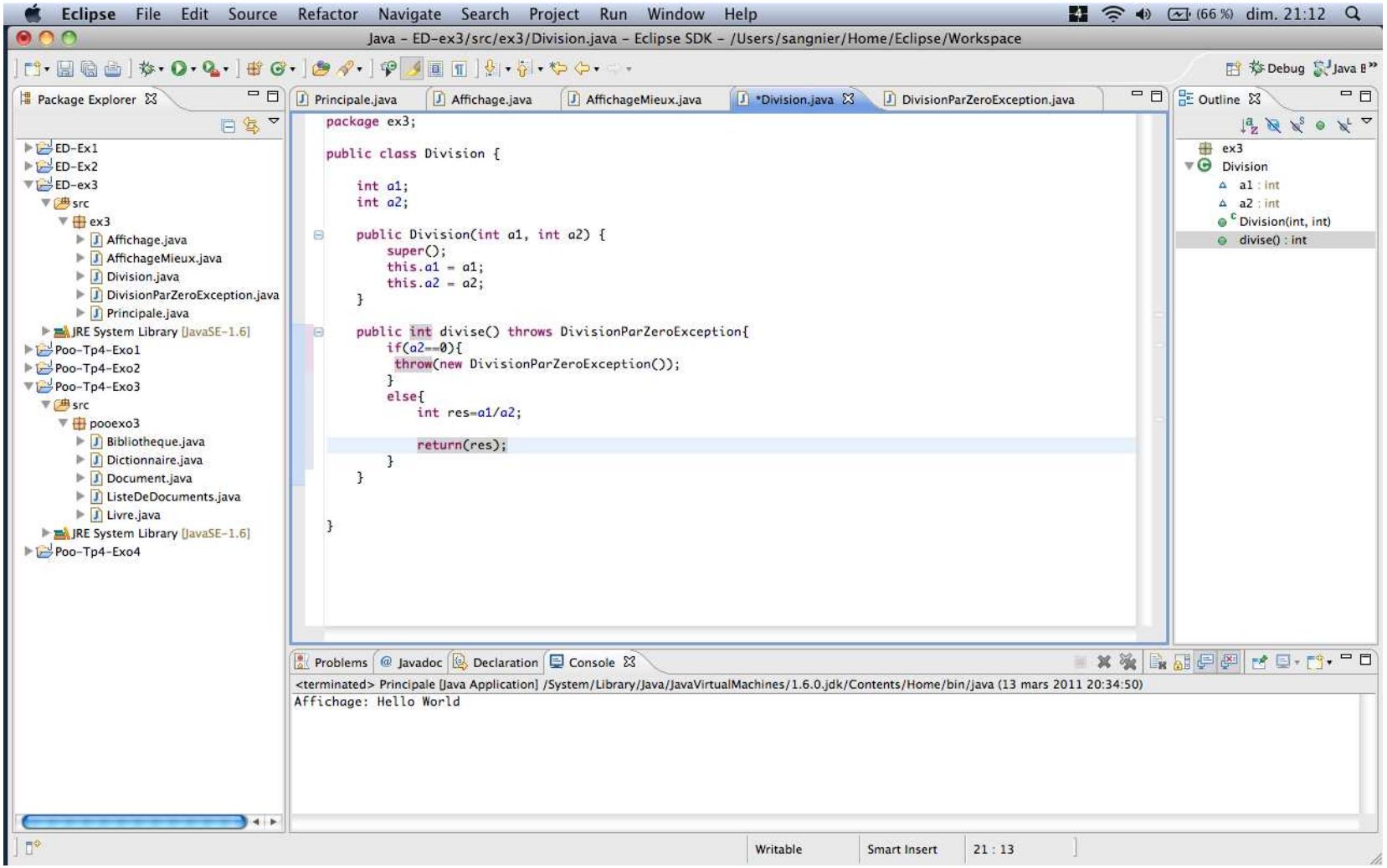
Et pour les exceptions ?



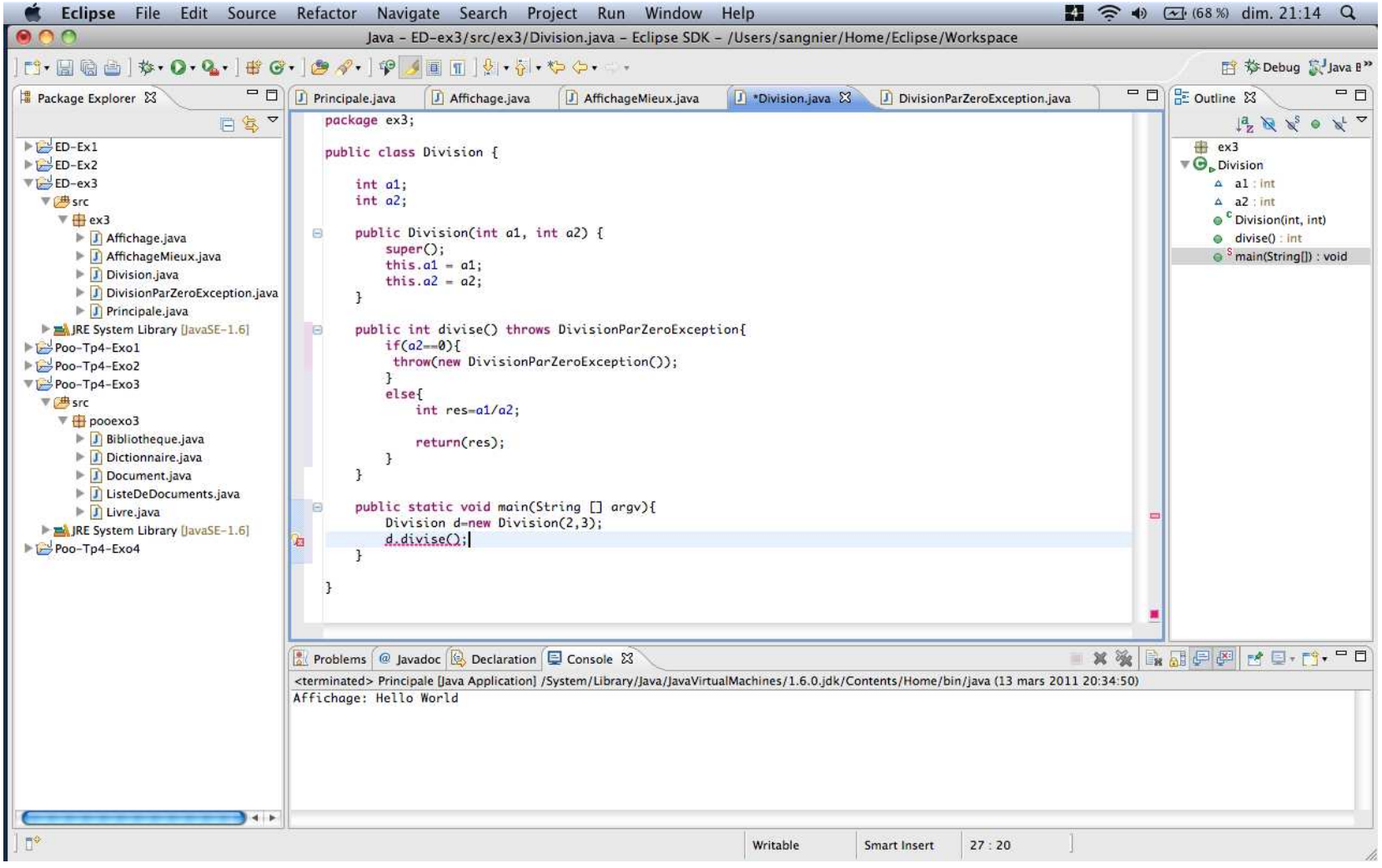
Et pour les exceptions ?



Et pour les exceptions ?

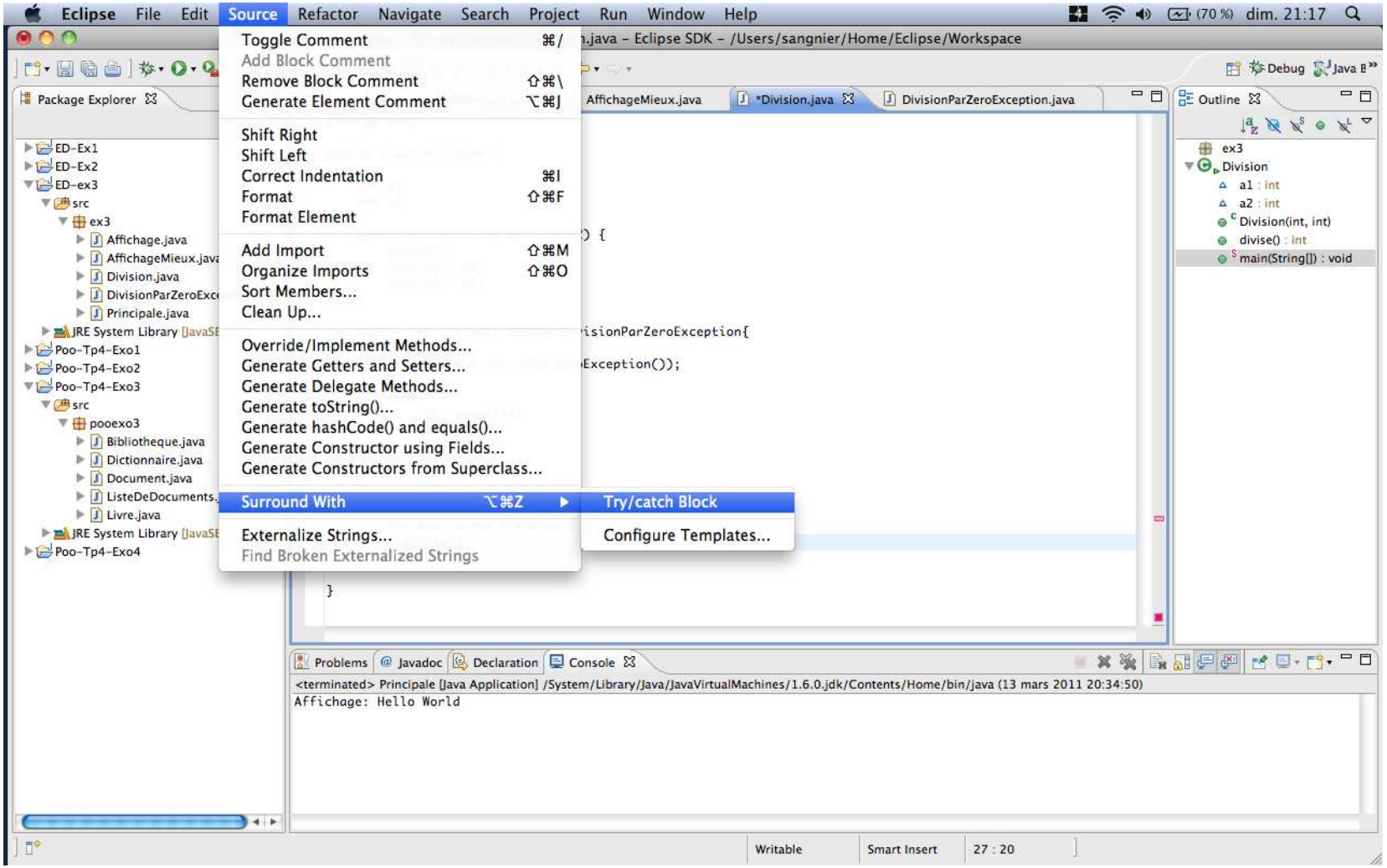


Et pour les exceptions ?

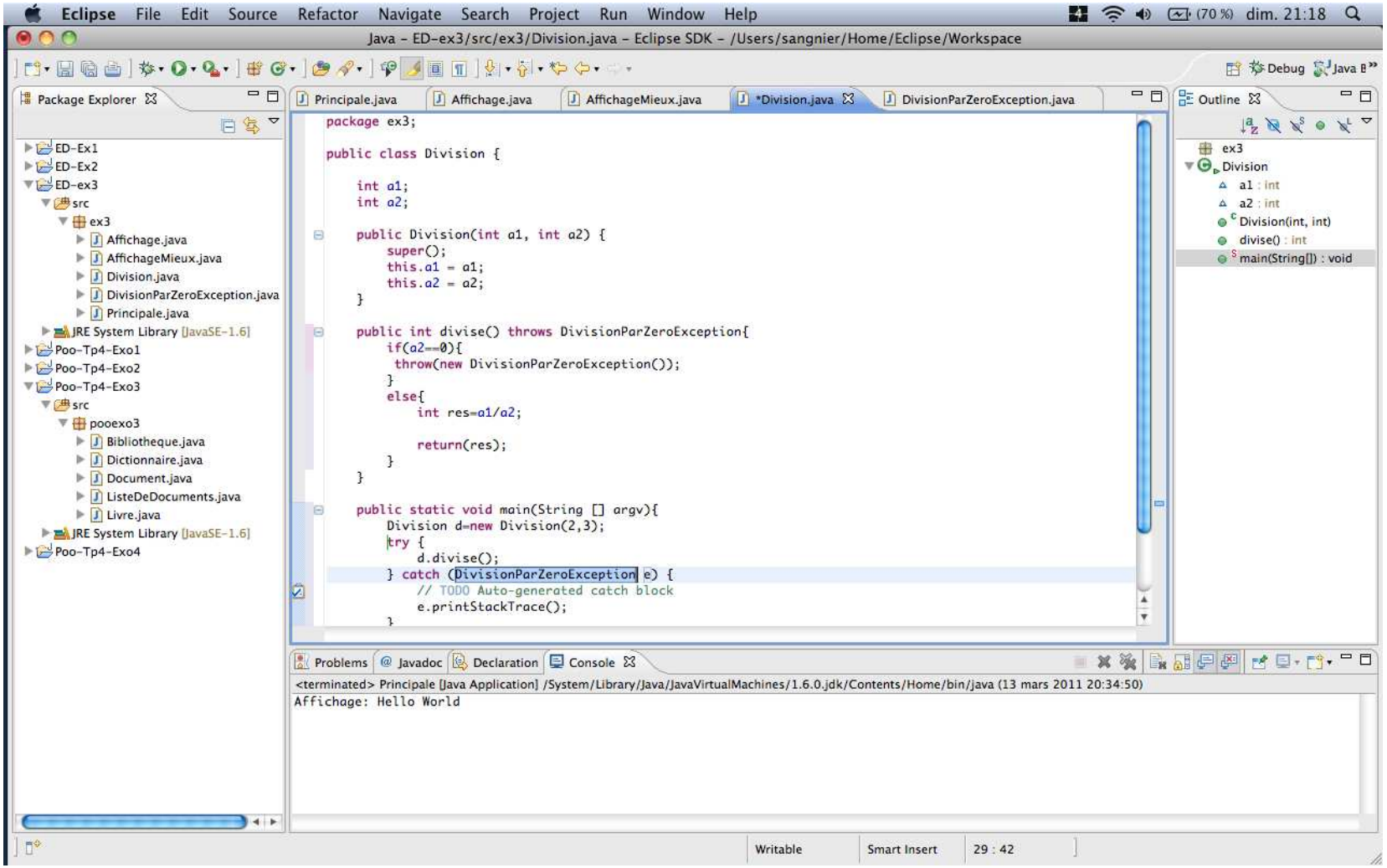


Erreur : car on ne *catch* pas l'exception

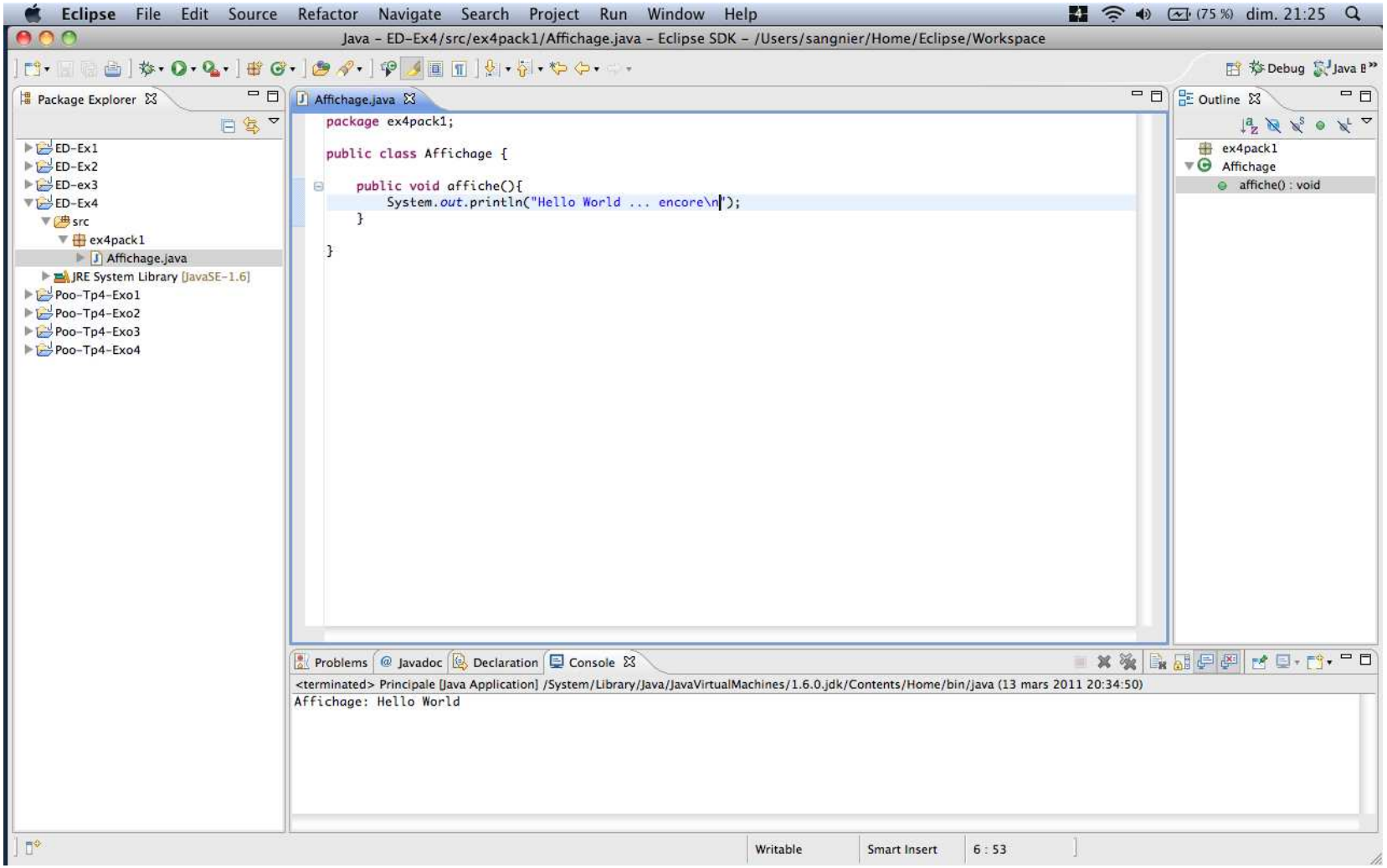
Insertion automatique de try ... catch



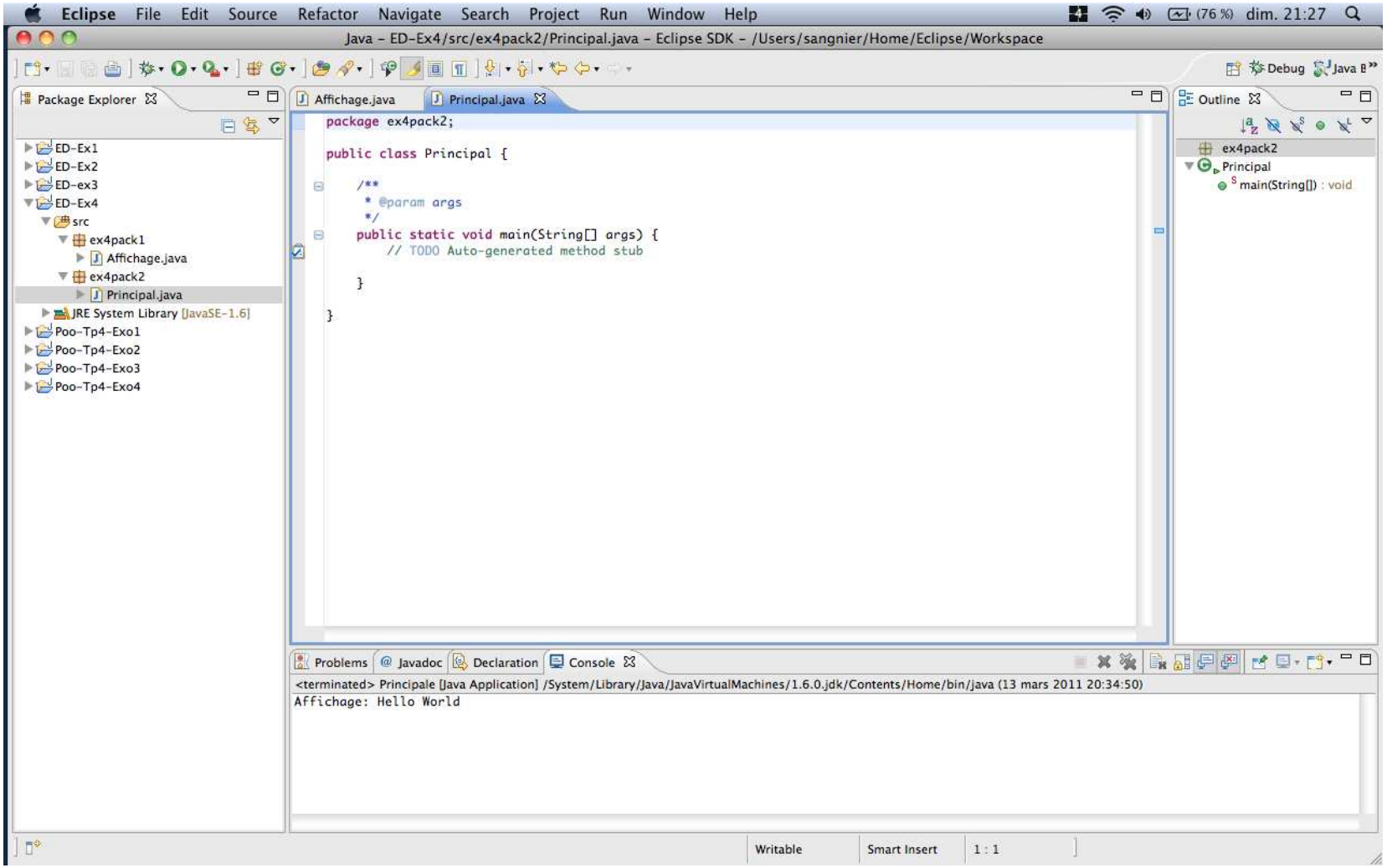
Insertion automatique de try ... catch



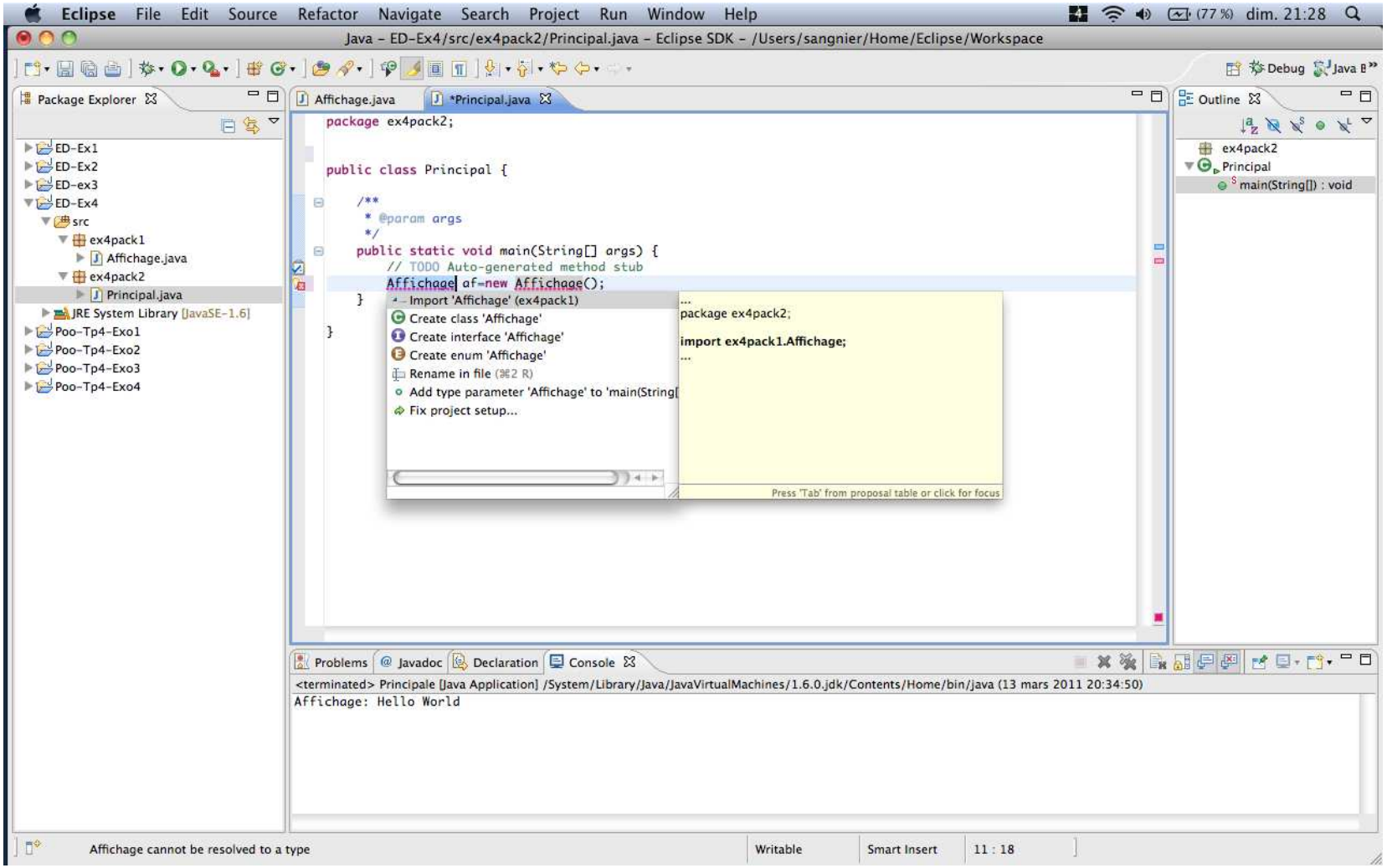
Gérer plusieurs packages



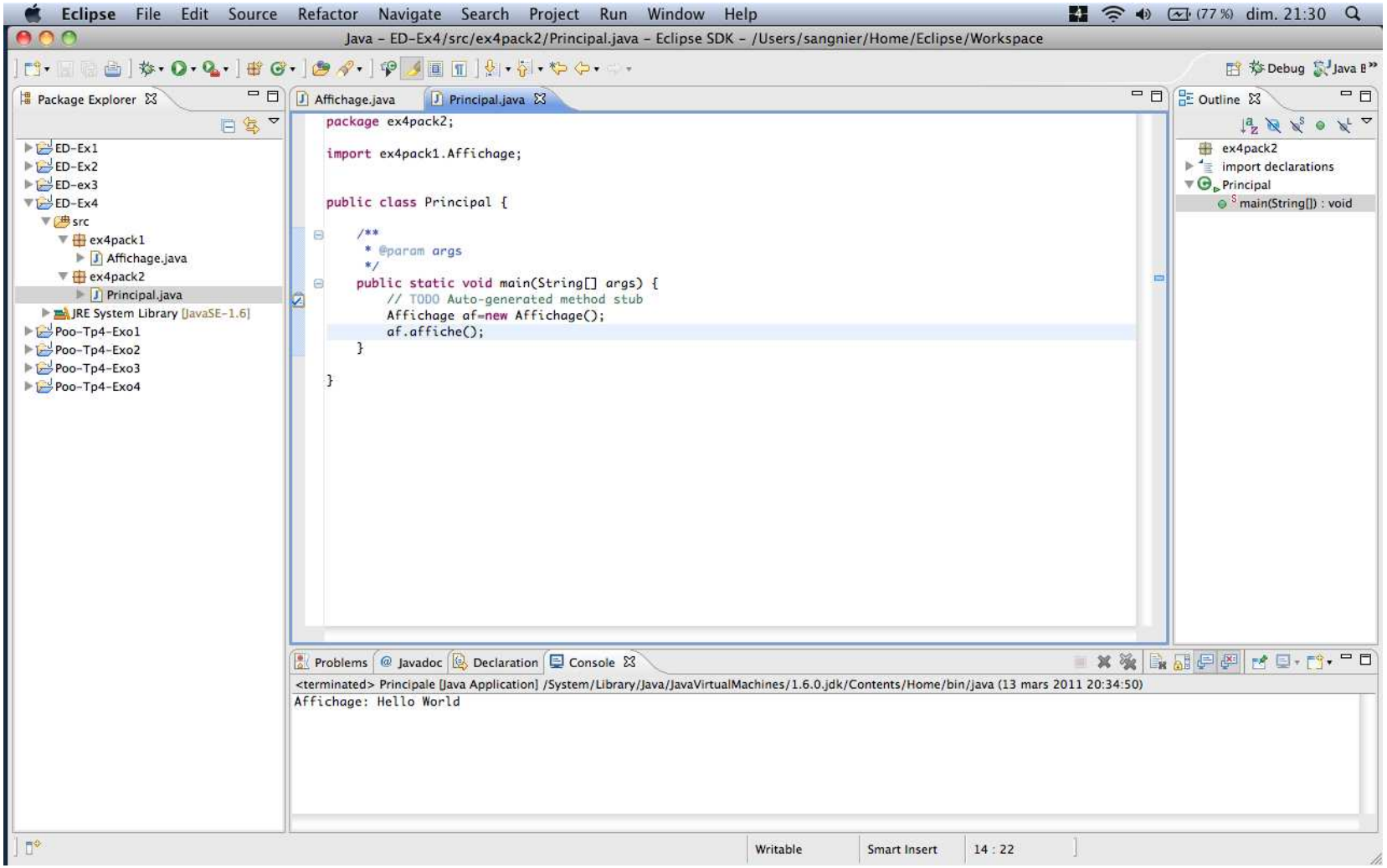
Gérer plusieurs packages



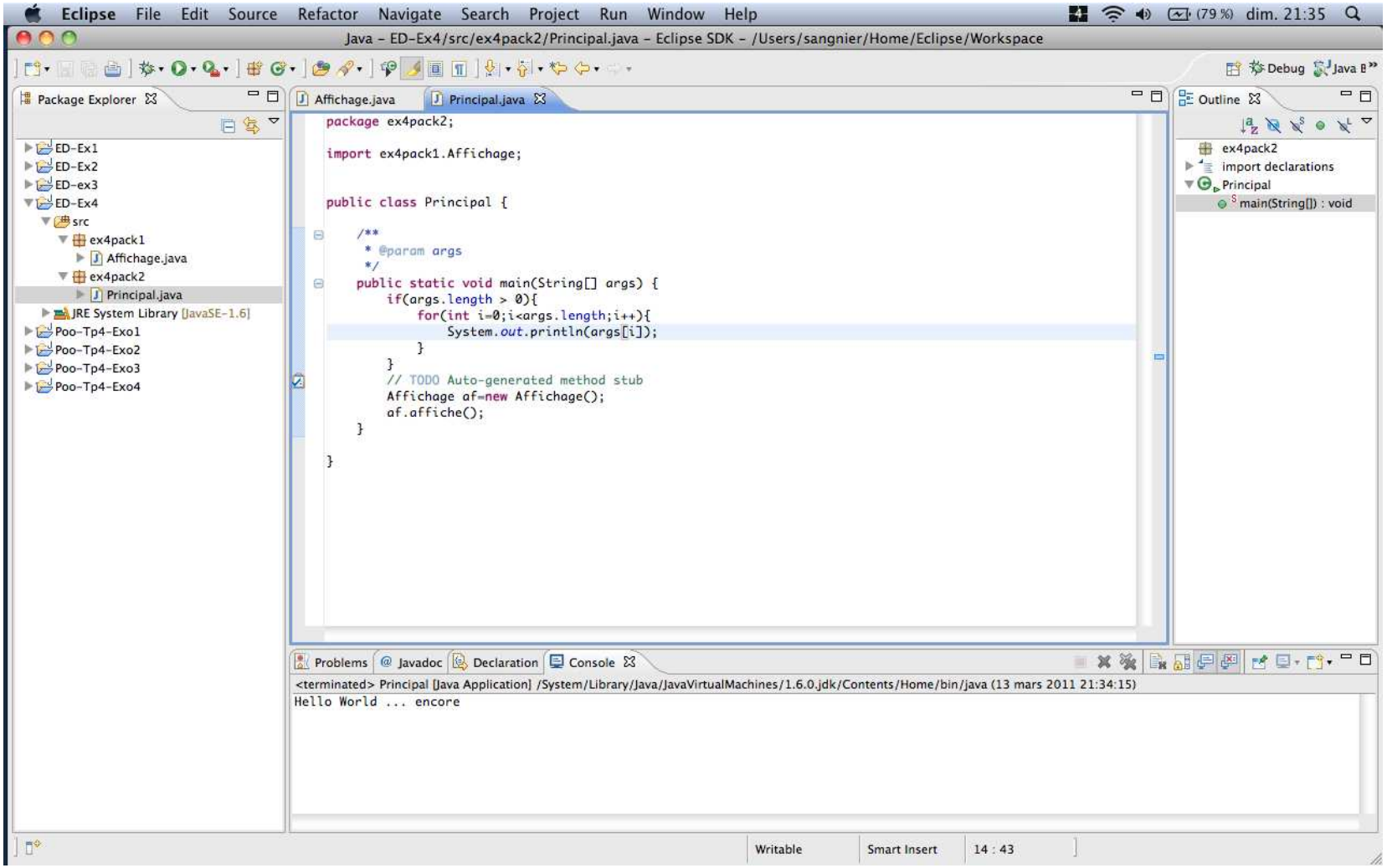
Gérer plusieurs packages



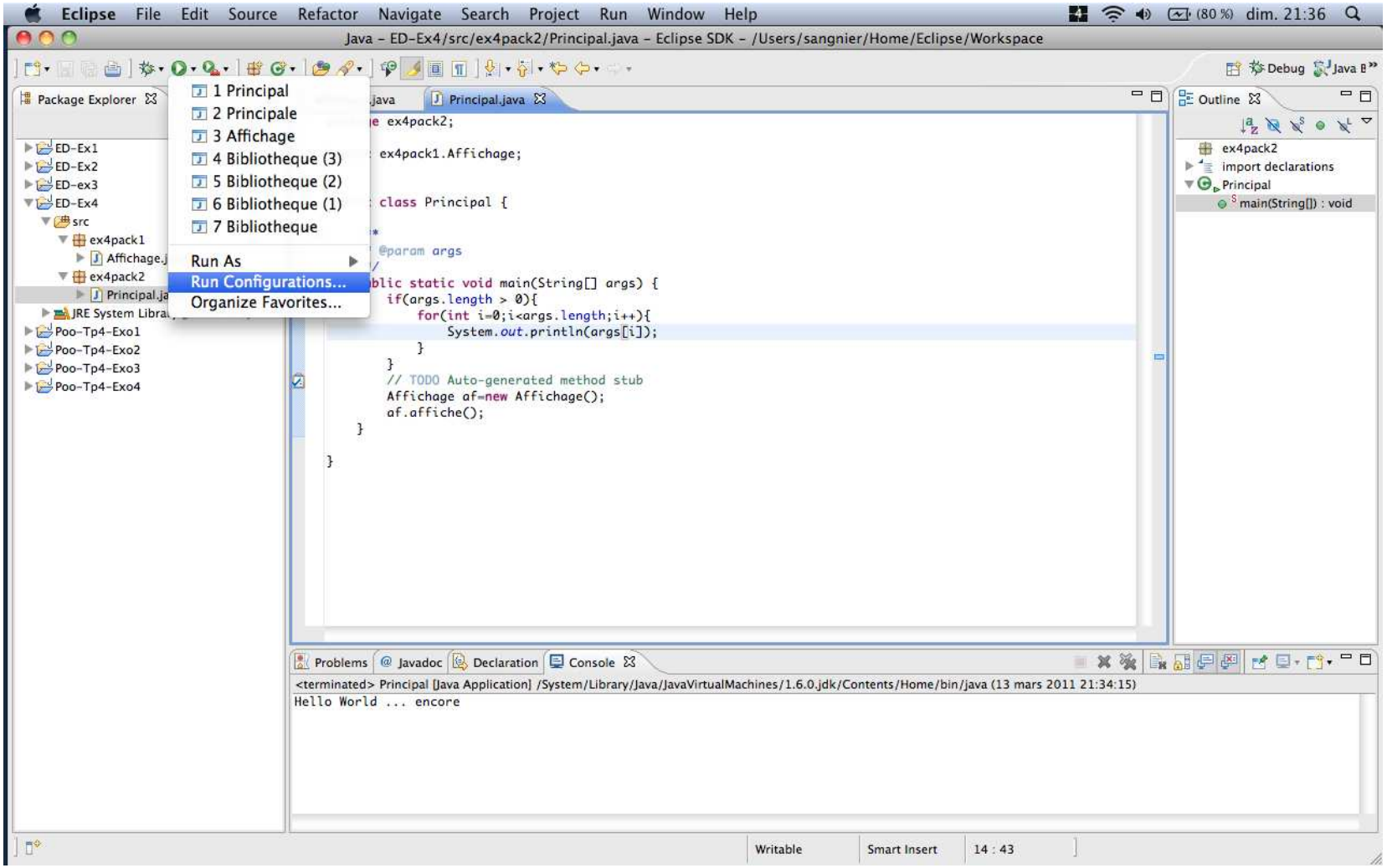
Gérer plusieurs packages



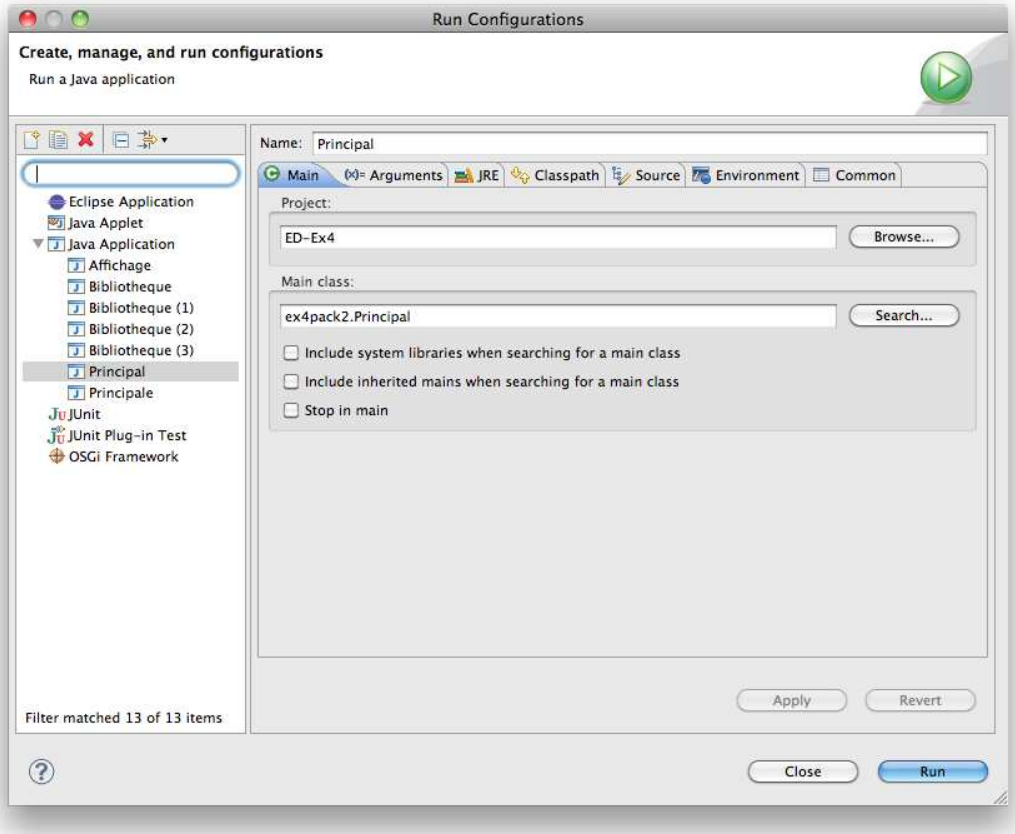
Donner des arguments au main



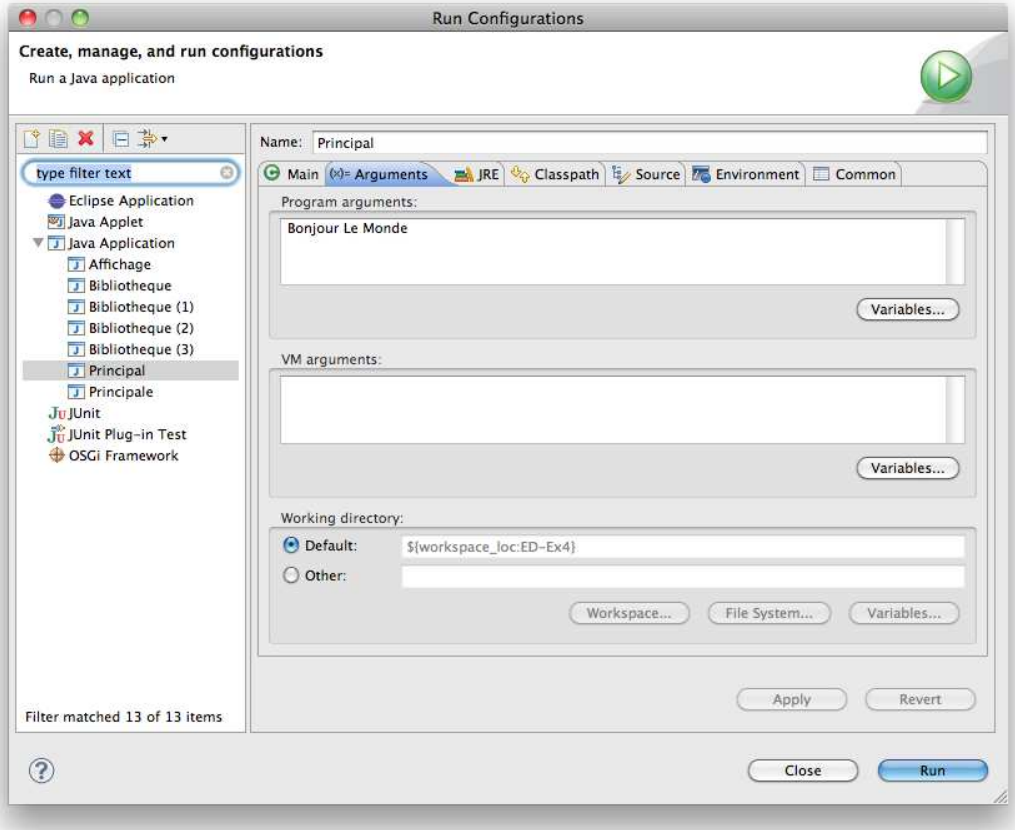
Donner des arguments au main



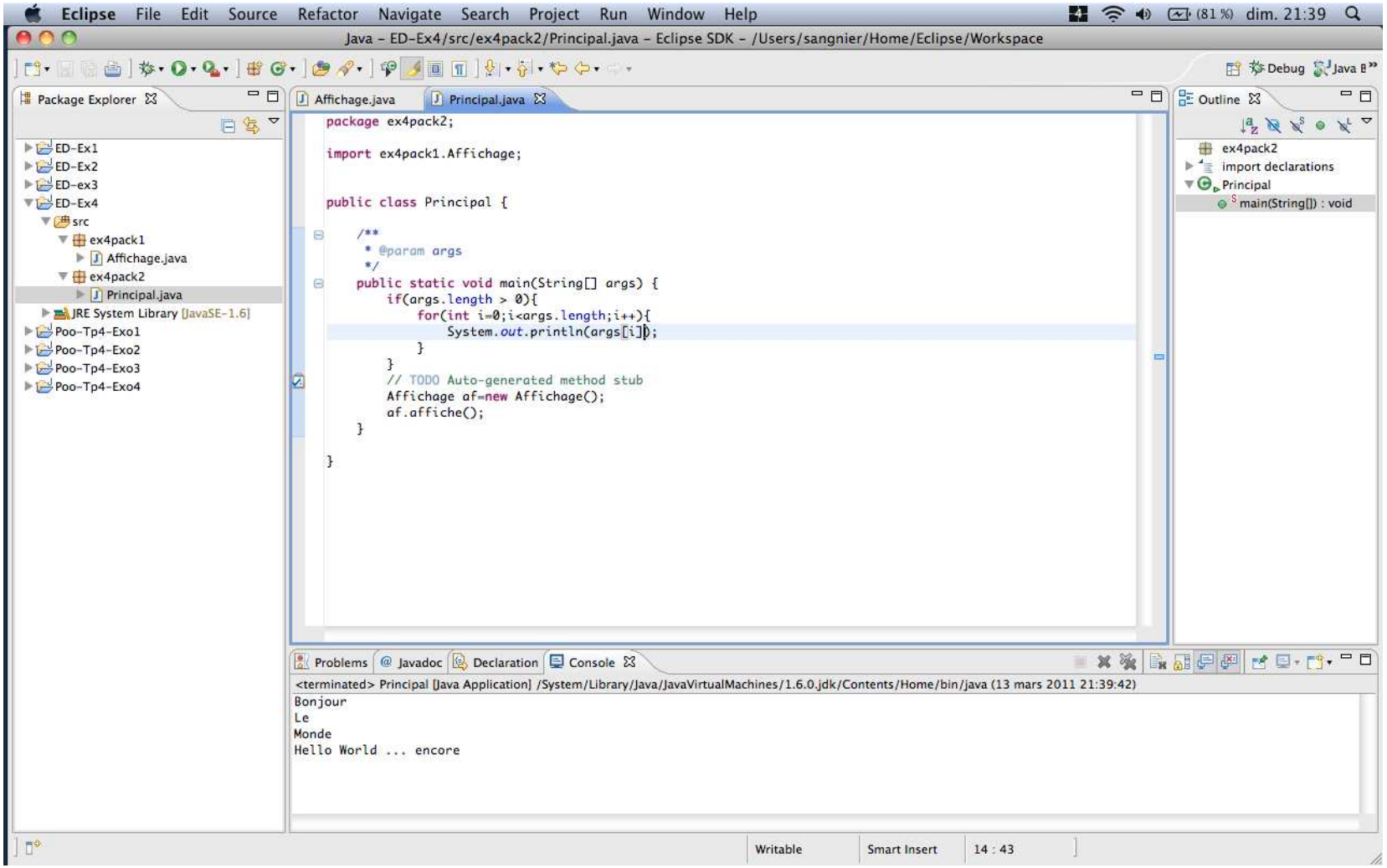
Donner des arguments au main



Donner des arguments au main



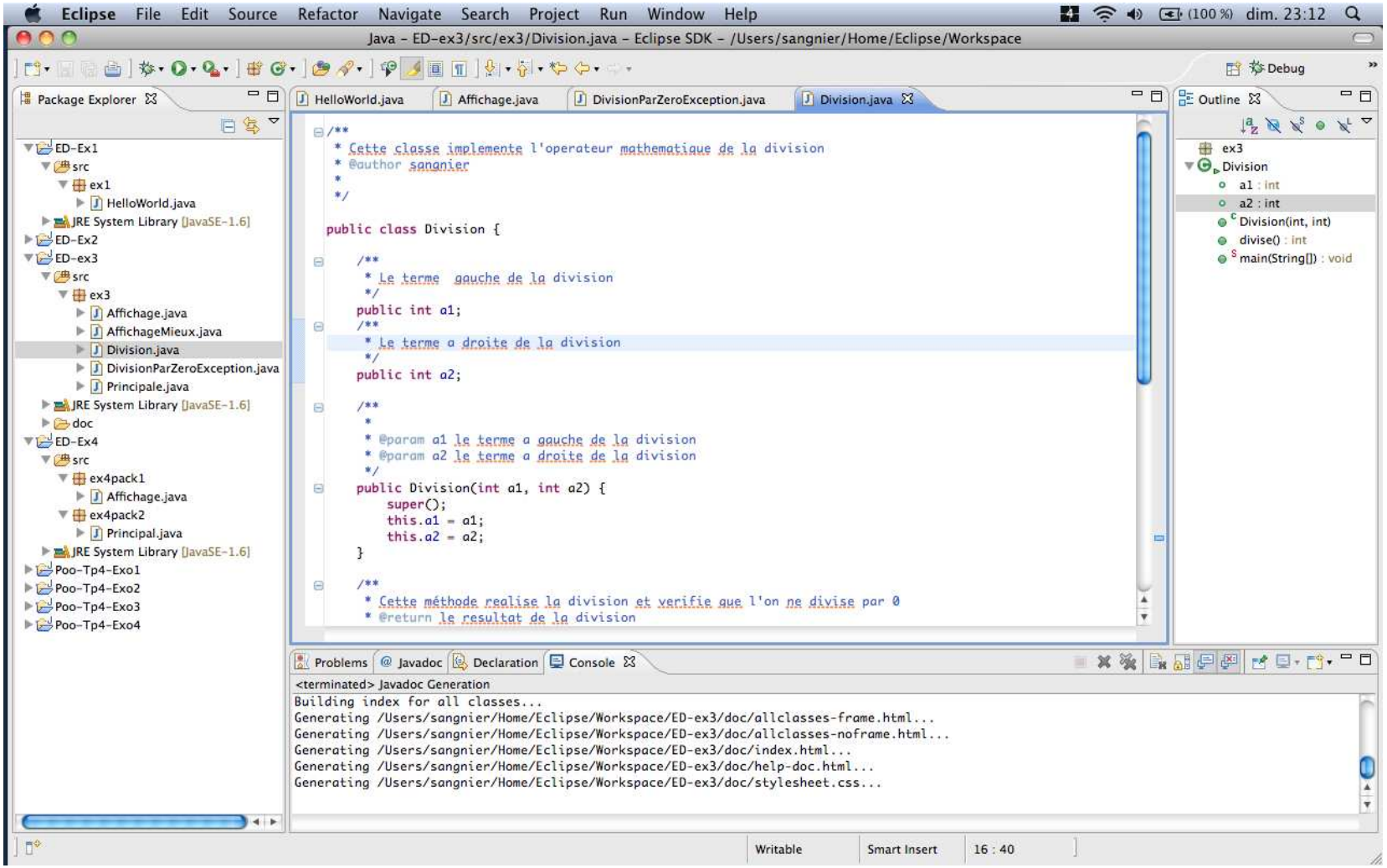
Donner des arguments au main



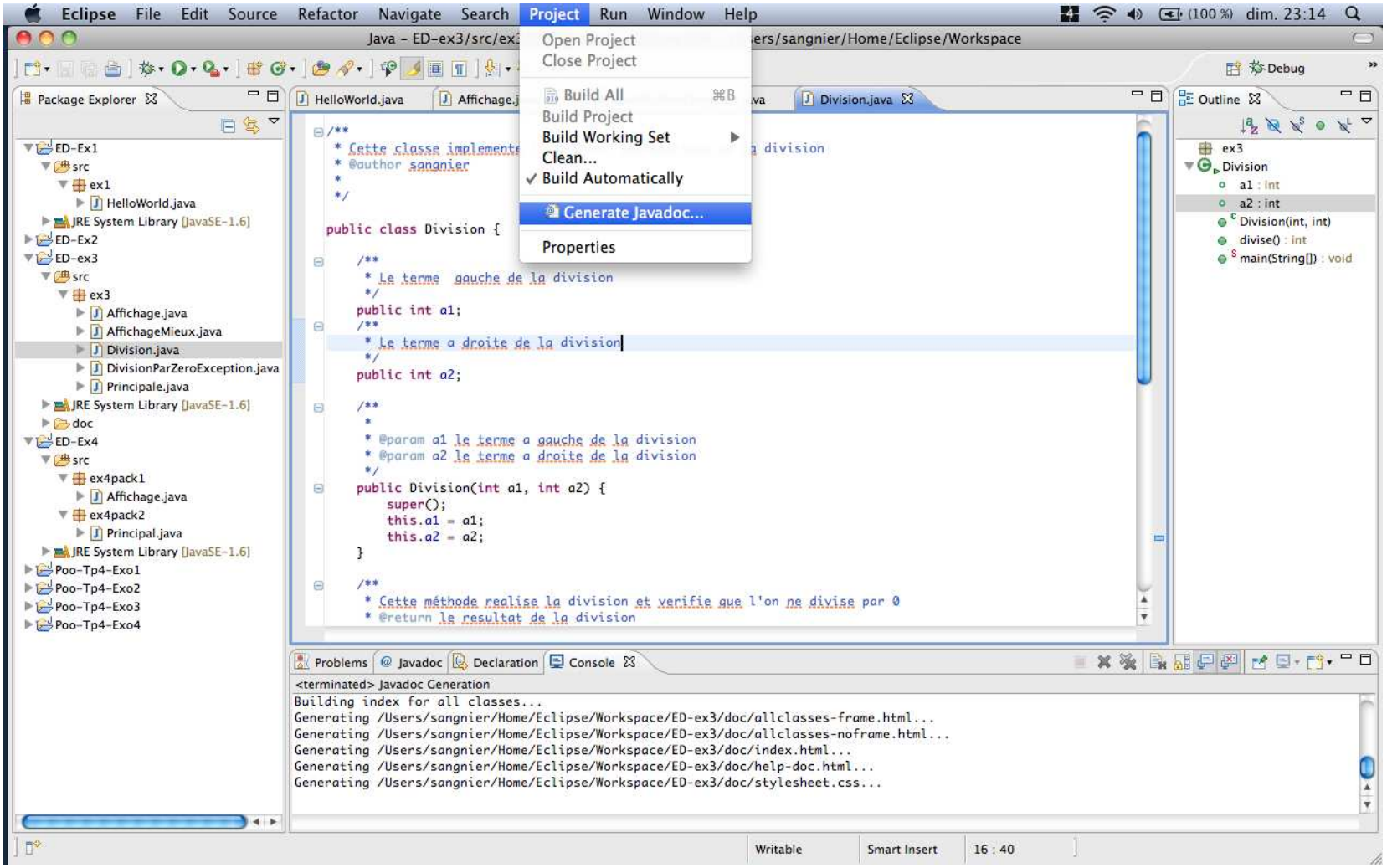
Rappel sur Javadoc

- Permet de générer automatiquement une documentation au format HTML
- Fonctionne par annotation du programme à l'aide *tags* spéciaux
- Les commentaires Javadoc commencent par /** et finissent par */
- Les tags :
  - @param : Définit un paramètre de méthode. Requis pour chaque paramètre.
  - @return : Documente la valeur de retour. Ce tag ne devrait pas être employé pour des constructeurs ou des méthodes définis avec un type de retour void.
  - @throws : Documente une exception lancée par une méthode.
  - @author : Nom du développeur.
  - @version : Donne la version d'une classe ou d'une méthode.
  - @see : Documente une association à une autre méthode ou classe.
  - @since : Précise à quelle version de la SDK/JDK une méthode a été ajoutée à la classe.
  - @deprecated : Marque la méthode comme dépréciée. Certains IDEs créent un avertissement à la compilation si la méthode est appelée.

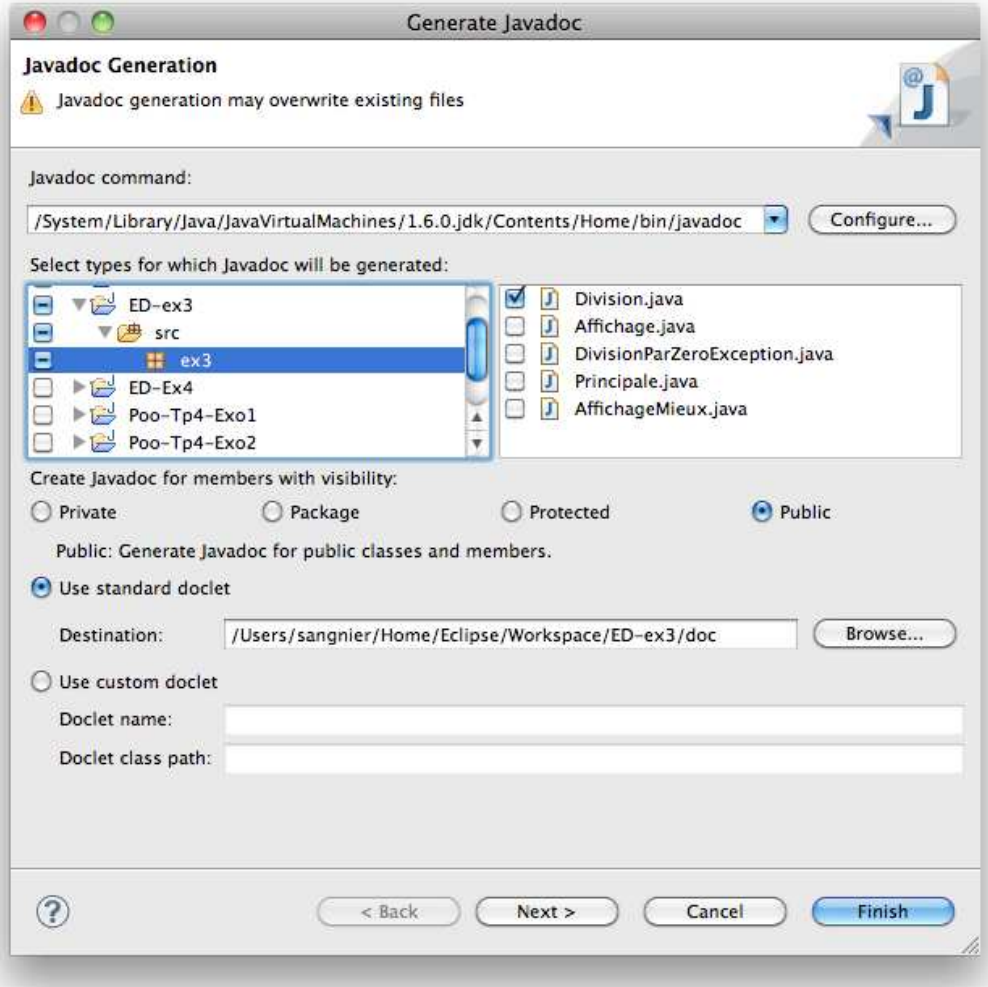
Générer la javadoc avec Eclipse



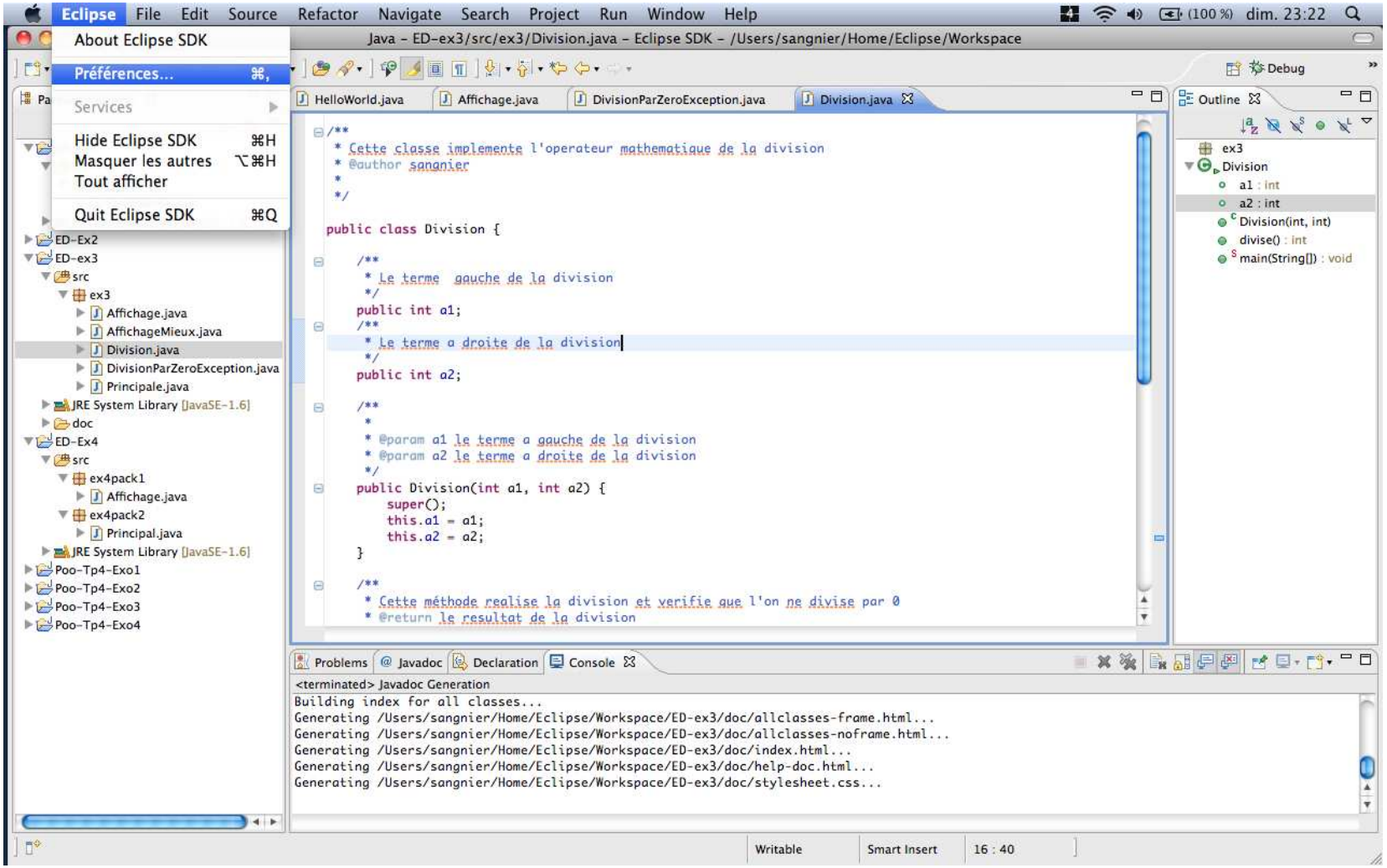
Générer la javadoc avec Eclipse



Générer la javadoc avec Eclipse



Personnaliser Eclipse



Personnaliser Eclipse