

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 facile à étendre
- Paramétrisable selon les besoins/goûts du programmeur
- Capable d'automatiser les tâches lourdes du développement
- Utilisable pour son propre développement (*bootstrap*-able)

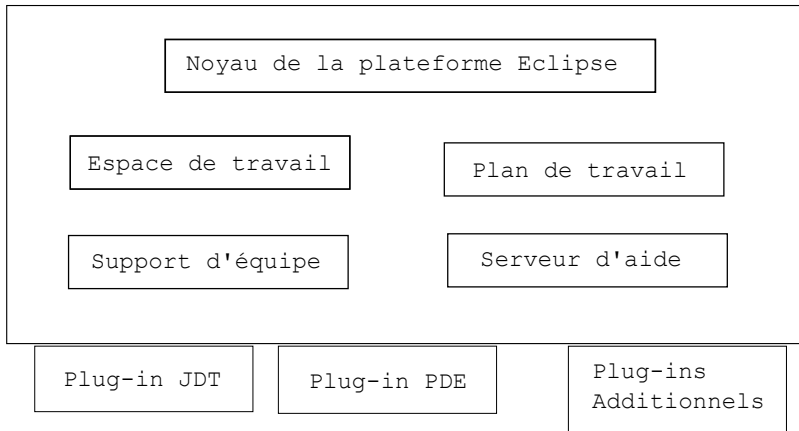
Ressources

- 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

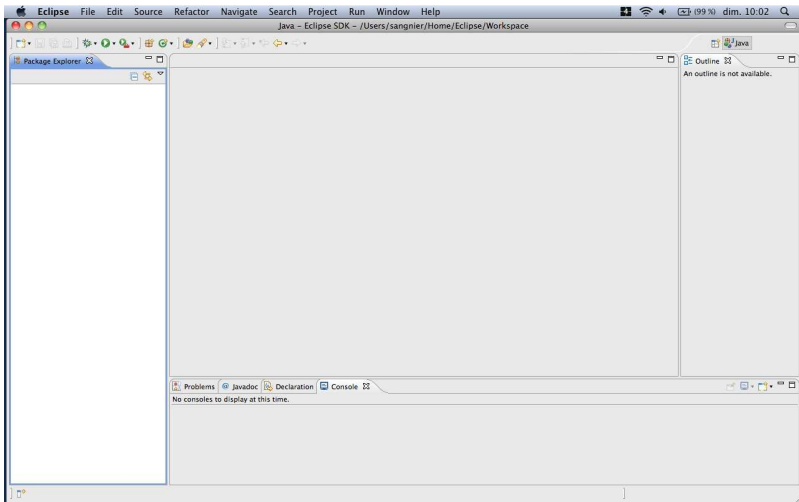


- *JDT : Java Development Toolkit*
- *PDE : Plug-in Development Environment*

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 à 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

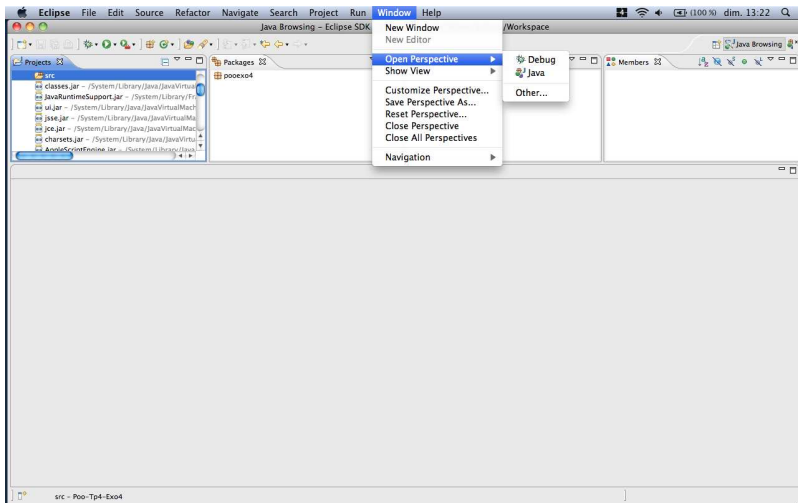
Vues

- 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 IHM)
- Fenêtre d'édition : lieu principal pour le développement de code
- JDT possède un éditeur riche en possibilités

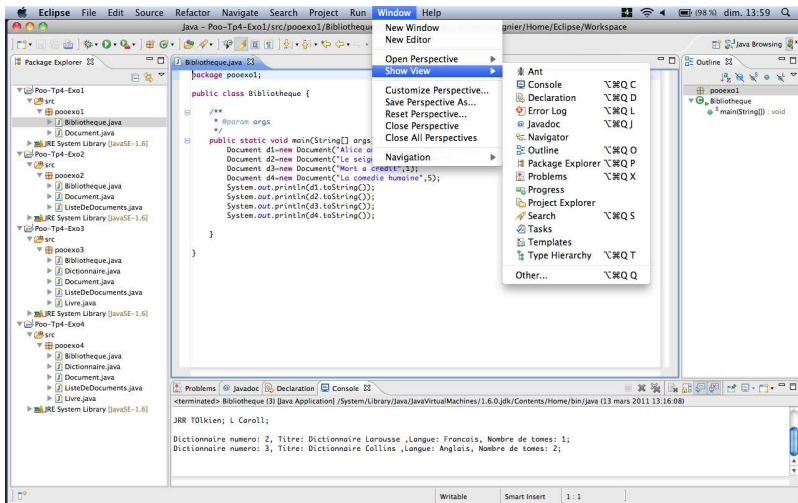
Perspectives

- 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 débogue un programme
- Possibilité de créer ses propres perspectives

Ouvrir une perspective



Ouvrir une vue



The screenshot shows the Eclipse IDE interface. The 'Window' menu is open, and the 'Show View' option is selected. A sub-menu is displayed, listing various views and their keyboard shortcuts:

- Ant
- Console `\t#\tQ C`
- Declaration `\t#\tQ D`
- Error Log `\t#\tQ L`
- Javadoc `\t#\tQ J`
- Navigator
- Outline `\t#\tQ O`
- Package Explorer `\t#\tQ P`
- Problems `\t#\tQ X`
- Progress
- Project Explorer
- Search `\t#\tQ S`
- Tasks
- Templates
- Type Hierarchy `\t#\tQ T`
- Other... `\t#\tQ Q`

The main editor window displays the following Java code:

```
package poeexo1;

public class Bibliotheque {

    /**
     * @param args
     */
    public static void main(String[] args) {
        Document d1=new Document("Alice a écrit",3);
        Document d2=new Document("Le seigneur des anneaux",10);
        Document d3=new Document("Mort a credit",3);
        Document d4=new Document("Le comedie humaine",5);
        System.out.println(d1.toString());
        System.out.println(d2.toString());
        System.out.println(d3.toString());
        System.out.println(d4.toString());
    }
}
```

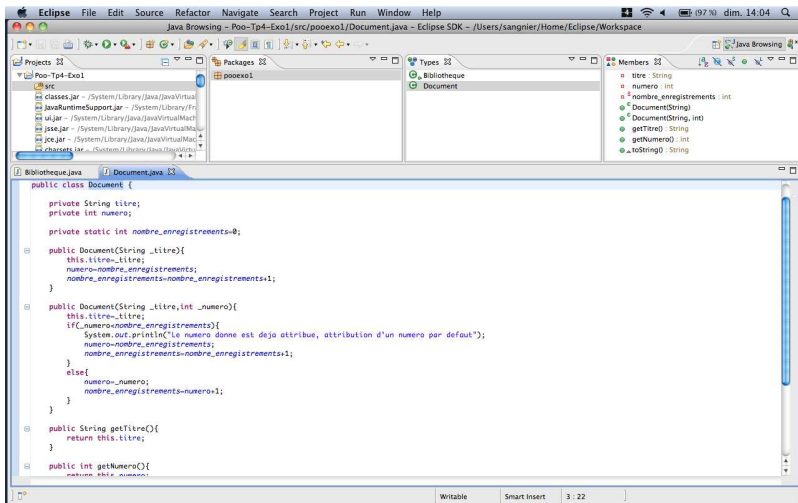
The bottom of the IDE shows the Console output:

```
<terminated> Bibliotheque (3) [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java [13 mars 2011 13:16:08]

JRR T01kten; L Carol1;

Dictionnaire numero: 2, Titre: Dictionnaire Larousse ,Langue: Francais, Nombre de tomes: 1;
Dictionnaire numero: 3, Titre: Dictionnaire Collins ,Langue: Anglais, Nombre de tomes: 2;
```


Perspective Java Browsing



Perspective Débogage

The screenshot shows the Eclipse IDE in a debugging perspective. The main editor displays the source code of the `Document` class:

```
public class Document {  
    private String titre;  
    private int numero;  
  
    private static int nombre_enregistrements=0;  
  
    public Document(String _titre){  
        this.titre=_titre;  
        numero=nombre_enregistrements;  
        nombre_enregistrements++;  
    }  
  
    public Document(String _titre,int _numero){  
        this.titre=_titre;  
        if(_numero<nombre_enregistrements){
```

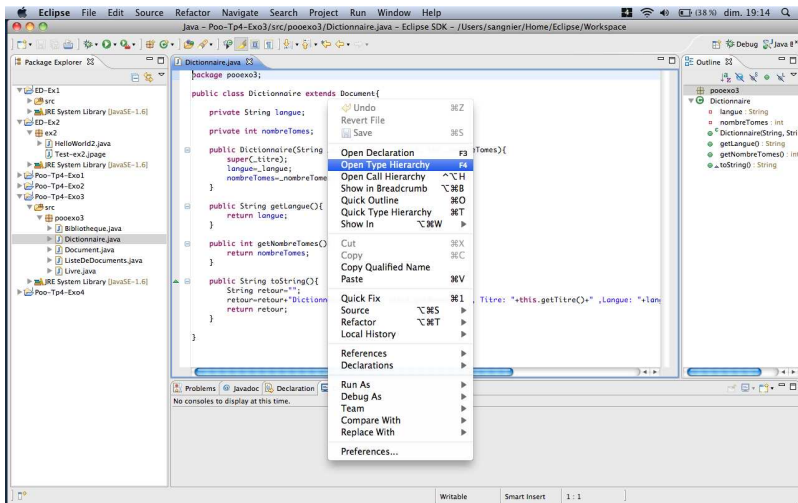
The console window at the bottom shows the output of the application:

```
<terminated> Bibliotheque (3) [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (13 mars 2011 13:16:08)  
dictionnaire numero: 3, titre: dictionnaire collins ,Langue: anglais, nombre de tomes: 2;  
  
JRR Tolkien; L Carroll;  
  
Dictionnaire numero: 2, Titre: Dictionnaire Larousse ,Langue: Francais, Nombre de tomes: 1;  
Dictionnaire numero: 3, Titre: Dictionnaire Collins ,Langue: Anglais, Nombre de tomes: 2;
```

The Outline view on the right shows the class structure:

- pooxo1
 - Document
 - titre : String
 - numero : int
 - nombre_enregistrements : int
 - Document(String)
 - Document(String, int)
 - getTitre() : String
 - getNumero() : int
 - _toString() : String

Vue Hiérarchie de Types



Vue Hiérarchie de Type

The screenshot displays the Eclipse IDE interface. The main editor shows the source code for the `Dictionnaire` class, which extends `Document`. The code includes private fields for `langue` and `nombreTomes`, and public methods for `getLangue()`, `getNombreTomes()`, and `toString()`.

```
package poeexo3;

public class Dictionnaire extends Document{

    private String langue;

    private int nombreTomes;

    public Dictionnaire(String _titre,String _langue, int _nombreTomes){
        super(_titre);
        langue=_langue;
        nombreTomes=_nombreTomes;
    }

    public String getLangue(){
        return langue;
    }

    public int getNombreTomes(){
        return nombreTomes;
    }

    public String toString(){
        String retour="";
        retour=retour+"Dictionnaire numero: "+this.getNumero()+", Titre: "+this.getTitre()+",langue: "+langue;
        return retour;
    }
}
```

The left sidebar shows the Package Explorer and Type Hierarchy. The Type Hierarchy view displays the class `Dictionnaire` and its members:

- langue : String
- nombreTomes : int
- Dictionnaire(String, String, int)
- getLangue() : String
- getNombreTomes() : int
- _toString() : String

The right sidebar shows the Outline view, which lists the package `poeexo3` and the class `Dictionnaire` with its members:

- langue : String
- nombreTomes : int
- Dictionnaire(String, String, int)
- getLangue() : String
- getNombreTomes() : int
- _toString() : String

The bottom status bar indicates the current file is `poeexo3.Dictionnaire - Poo-Tp4-Exo3/src`.

Perspective Java (II)

The screenshot displays the Eclipse IDE interface. The Package Explorer on the left shows a project named 'Poo-Tp4-Exo1' with several sub-packages: 'src', 'poexo1', 'poexo2', 'poexo3', and 'poexo4'. The 'Bibliothèque.java' file is open in the editor, showing the following code:

```
package poexo1;

public class Bibliothèque {

    /**
     * @param args
     */
    public static void main(String[] args) {
        Document d1=new Document("Alice aux pays des merveilles");
        Document d2=new Document("Le seigneur des anneaux");
        Document d3=new Document("Mort a credit",1);
        Document d4=new Document("Le comedie humaine",5);
        System.out.println(d1.toString());
        System.out.println(d2.toString());
        System.out.println(d3.toString());
        System.out.println(d4.toString());
    }
}
```

The Console at the bottom shows the output of the program:

```
<terminated> Bibliothèque (3) [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java [13 mars 2011 13:16:08]

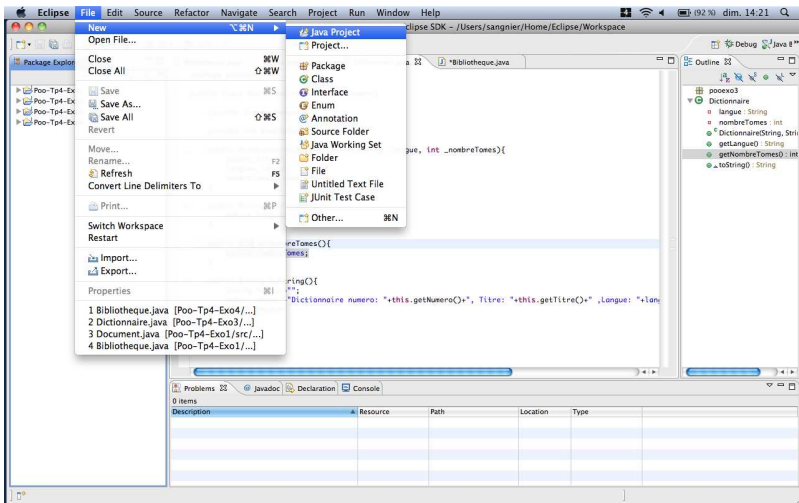
JRR T01kien; L Carol1;

Dictionnaire numero: 2, Titre: Dictionnaire Larousse ,Langue: Francais, Nombre de tomes: 1;
Dictionnaire numero: 3, Titre: Dictionnaire Collins ,Langue: Anglais, Nombre de tomes: 2;
```

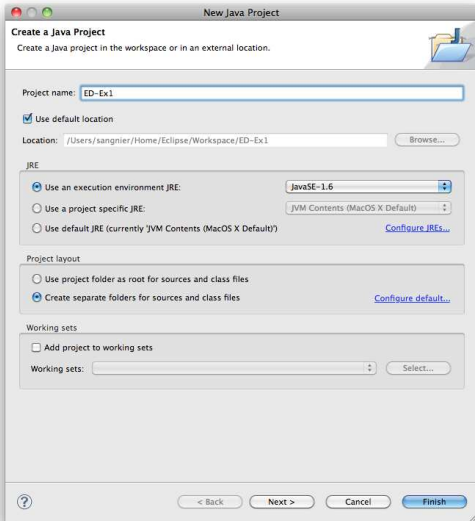
Détails de la perspective Java

- Lorsqu'on ouvre cette perspective, le JDT est lancé
- Perspective 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

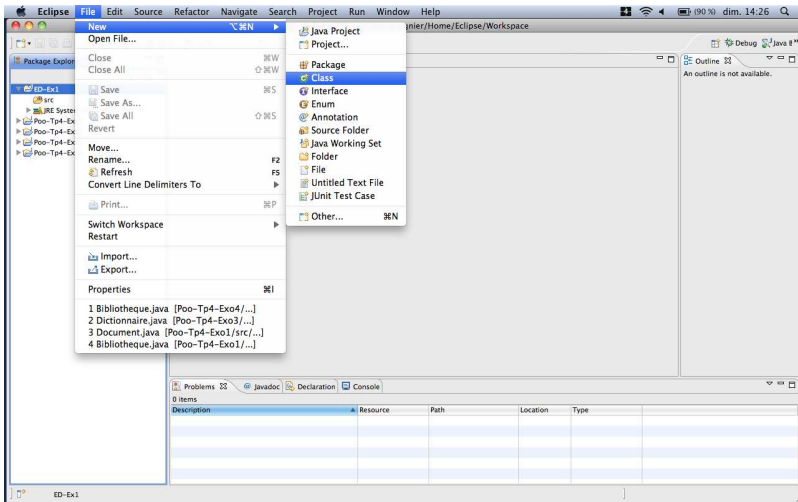
Création d'un nouveau projet



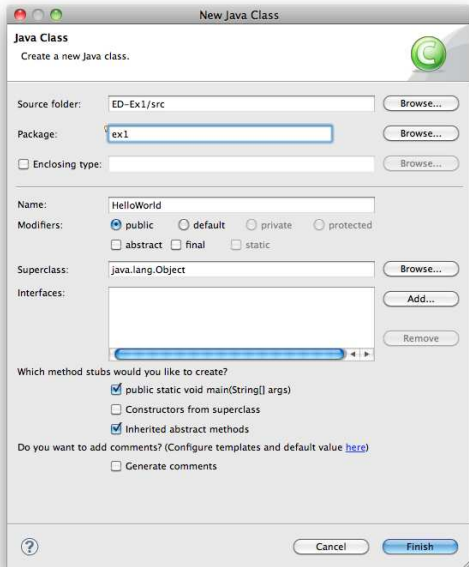
Création d'un nouveau projet



Création d'une nouvelle classe



Création d'une nouvelle classe



Création d'une nouvelle classe

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left displays the project structure: ED-Ex1 > src > ex1 > HelloWorld.java. The main editor window displays the code for HelloWorld.java:

```
package ex1;

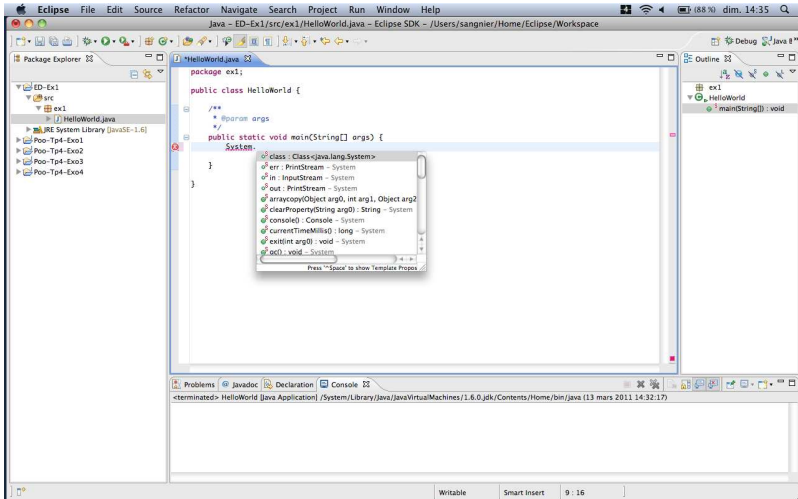
public class HelloWorld {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
    }

}
```

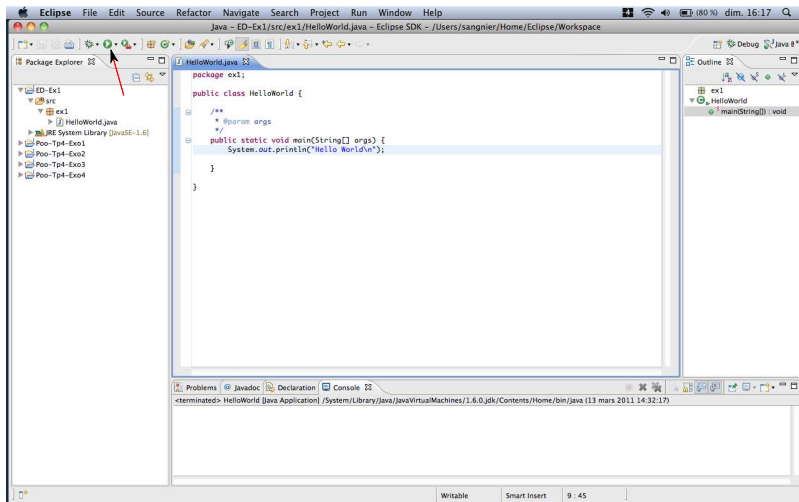
The Outline view on the right shows the class structure with the main method. The bottom status bar indicates 'Writable', 'Smart Insert', and '1:1'.

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

```
package ex1;

public class HelloWorld {

    /**
     * @param args
     */
    public static void main(String[] args) {
        System.out.println("Hello World\n");
    }

}
```

Console

```
<terminated> HelloWorld [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (13 mars 2011 16:38:36)
Hello World
```

Quelques remarques

Où se trouve votre programme :

- Dans votre workspace, un répertoire ED-Ex1 a été créé
- Si vous avez coché 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

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left displays a project named 'ED-Ex2' with a sub-project 'HelloWorld2.java'. The main editor window shows the source code for 'HelloWorld2.java' in the 'ex2' package. The code defines a 'HelloWorld2' class with two methods: 'printer()' and 'main()'. The 'printer()' method prints 'Hello World' and the current time. The 'main()' method is a stub that calls 'printer()'. The Outline view on the right shows the class structure. The Console view at the bottom shows the output 'Hello World'.

```
package ex2;

public class HelloWorld2 {

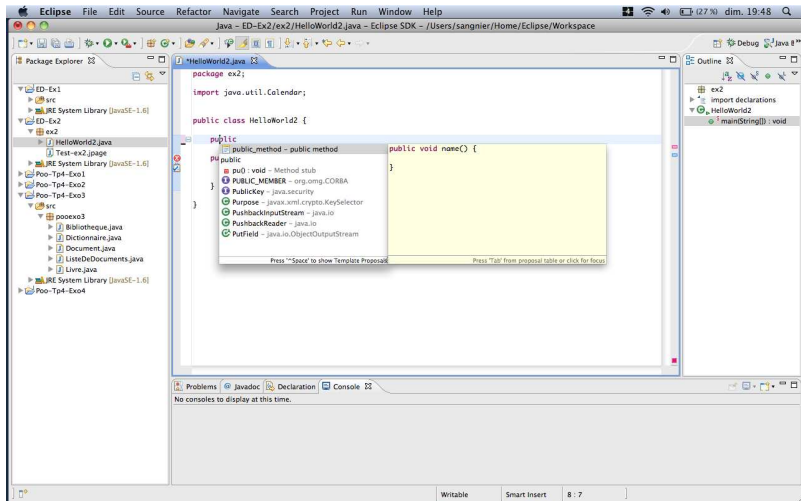
    public void printer(){
        outstring ="Hello World ";
        Calendar rightnow=Calendar.getInstance();
        System.out.println(outstring +rightnow.getTime());
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
    }

}
```

Console Output:
<terminated> HelloWorld [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (13 mars 2011 16:38:36)
Hello World

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

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left shows a project structure with packages 'ex1' and 'ex2', and a class 'HelloWorld2.java' in 'ex2'. The main editor displays the following Java code:

```
package ex2;

public class HelloWorld2 {

    public void printer(){
        outstring ="Hello World ";
    }

    pub

}
```

A quick fix menu is open over the line `outstring ="Hello World ";`. The menu items are:

- Create local variable 'outstring'
- Create field 'outstring'
- Create parameter 'outstring'
- Remove assignment
- Rename in file (R2 R)

The 'Create local variable' option is selected. A tooltip for this option shows the following code snippet:

```
... public void printer()
String outstring = "Hello World ";
Calendar rightnow=Calendar.getInstance();
...
```

The Outline view on the right shows the class structure with 'printer() : void' and 'mainString() : void' listed.

The Console view at the bottom shows the output: `<terminated> HelloWorld [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (13 mars 2011 16:38:36) HelloWorld`

At the bottom of the IDE, a status bar message reads: `outstring cannot be resolved to a variable`. Other status bar elements include 'Writable', 'Smart Insert', and '7 : 18'.

Exemples d'utilisation du correcteur rapide

The screenshot shows the Eclipse IDE interface. The main editor window displays the following code:

```
package ex2;

public class HelloWorld2 {

    public void printer(){
        String putstring = "Hello World ";
        Calendar rightnow=Calendar.getInstance();
    }

    public
}

}
```

A quick fix proposal window is open over the line `Calendar rightnow=Calendar.getInstance();`. The proposal table contains the following items:

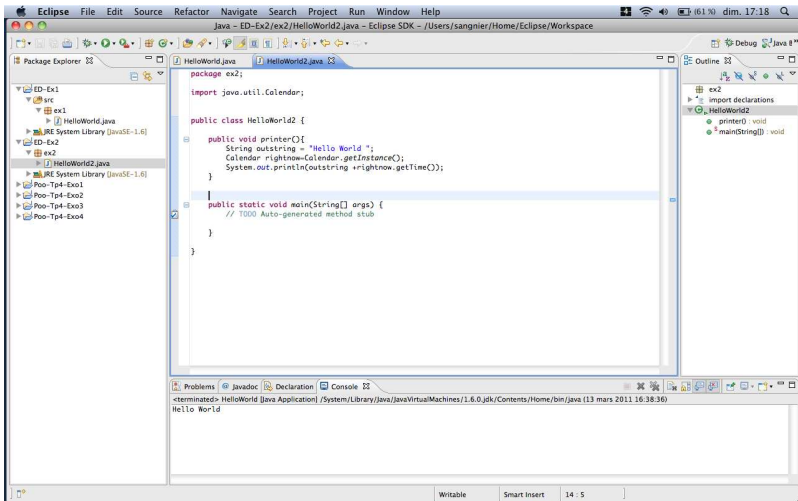
- Create class 'Calendar'
- Create interface 'Calendar'
- Create enum 'Calendar'
- Add type parameter 'Calendar' to 'HelloWorld2'
- Rename in file [src2.R]
- Add type parameter 'Calendar' to 'printer()'
- Fix project setup...

The 'Add type parameter' option is selected. The right-hand pane shows the Outline view with the following structure:

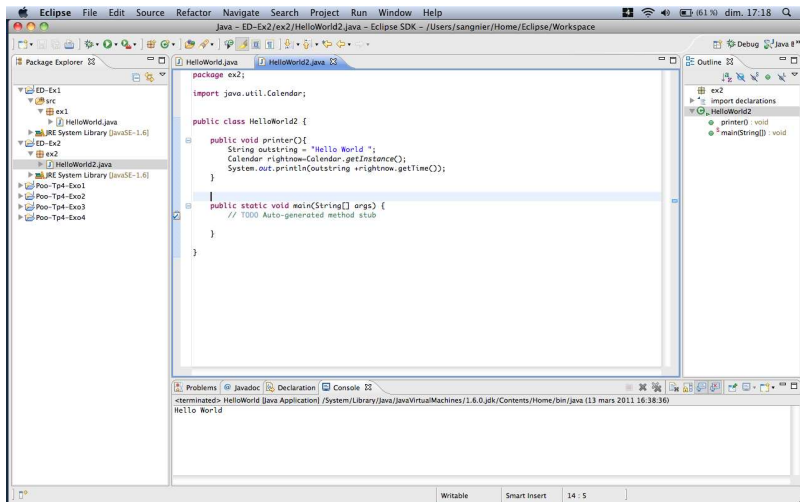
```
ex2
└─ HelloWorld2
   ├── printer() : void
   └─ mainString() : void
```

The bottom status bar displays the message: "Calendar cannot be resolved to a type".

Après correction



Après correction



```
package ex2;

import java.util.Calendar;

public class HelloWorld2 {

    public void printer(){
        String outstring = "Hello World ";
        Calendar rightnow=Calendar.getInstance();
        System.out.println(outstring +rightnow.getTime());
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
    }

}
```

Console Output:

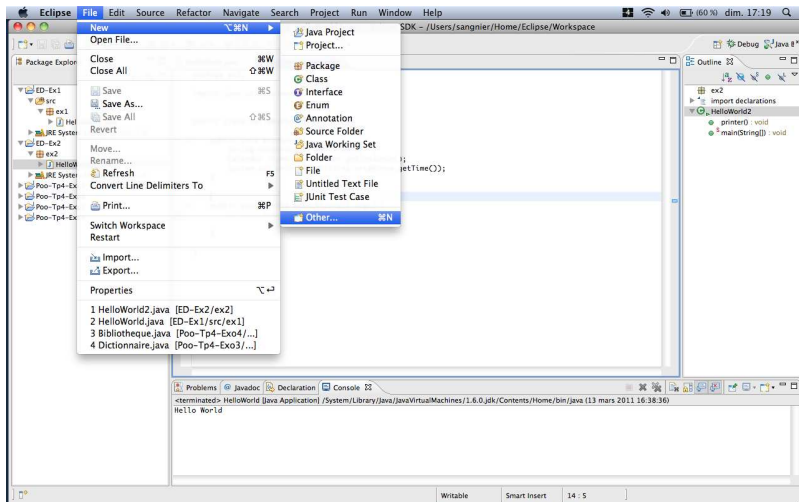
```
<terminated> HelloWorld [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (13 mars 2011 16:38:36)
Hello World
```

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

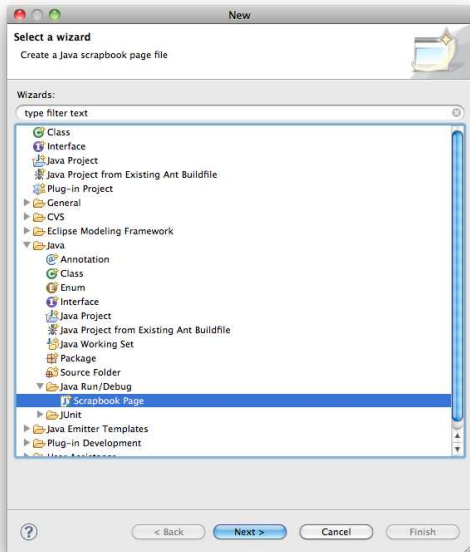
Les testeurs de code

- 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

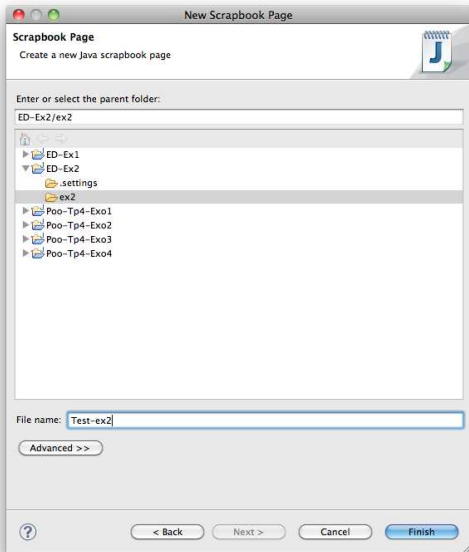
Ouvrir une page de testeur de code



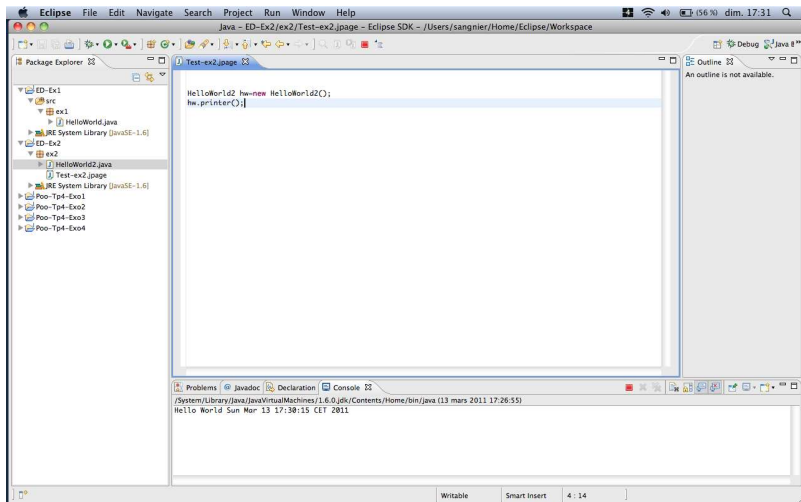
Ouvrir une page de testeur de code



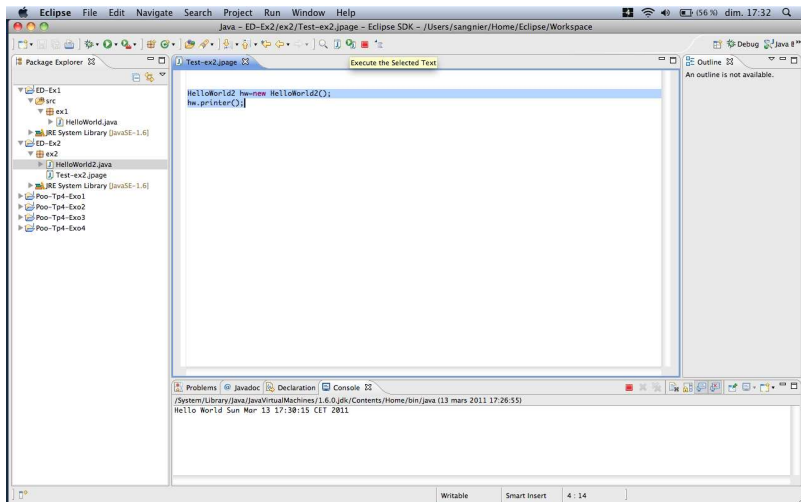
Ouvrir une page de testeur de code



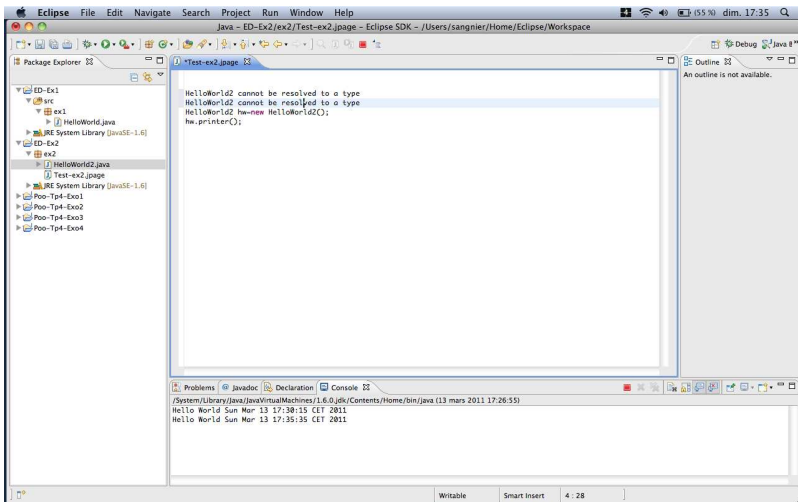
Ouvrir une page de testeur de code



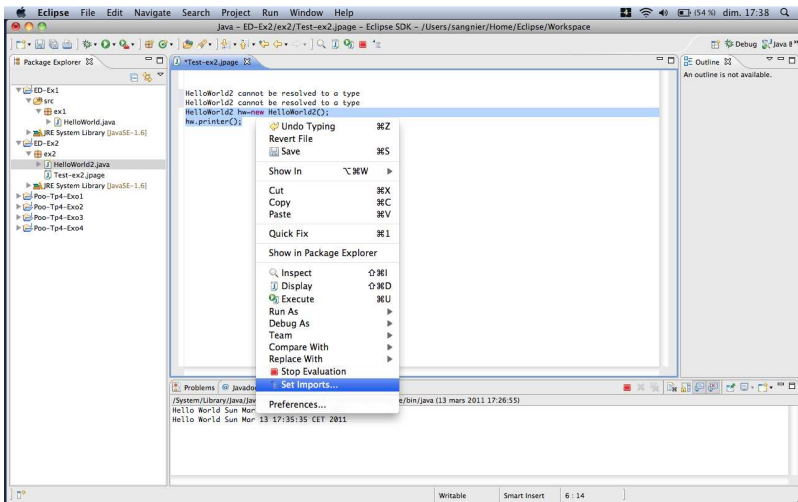
Exécuter le test sélectionné



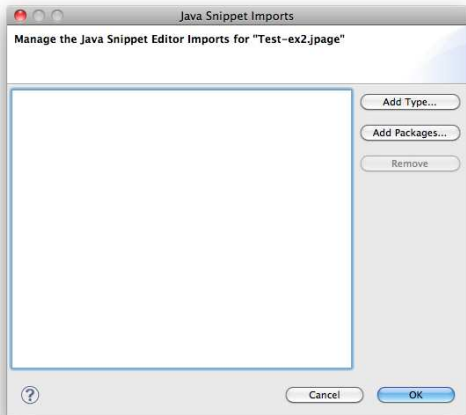
Les erreurs sont écrites dans le fichier



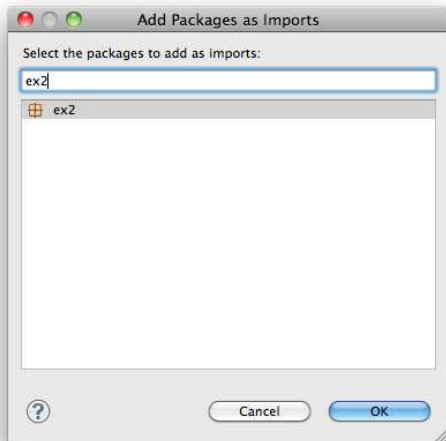
Faire les imports corrects



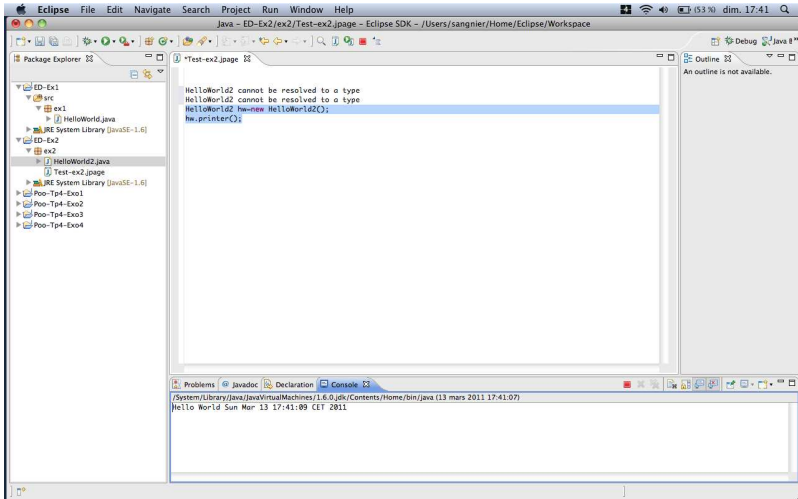
Faire les imports corrects



Faire les imports corrects

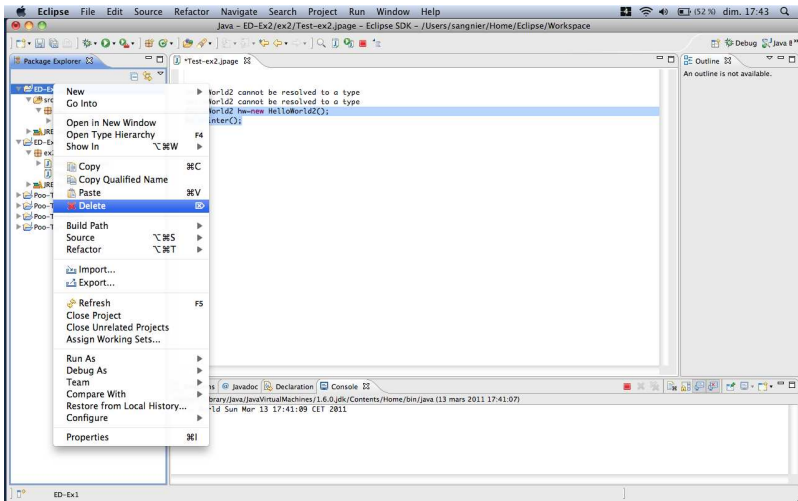


Résultat du test dans la console

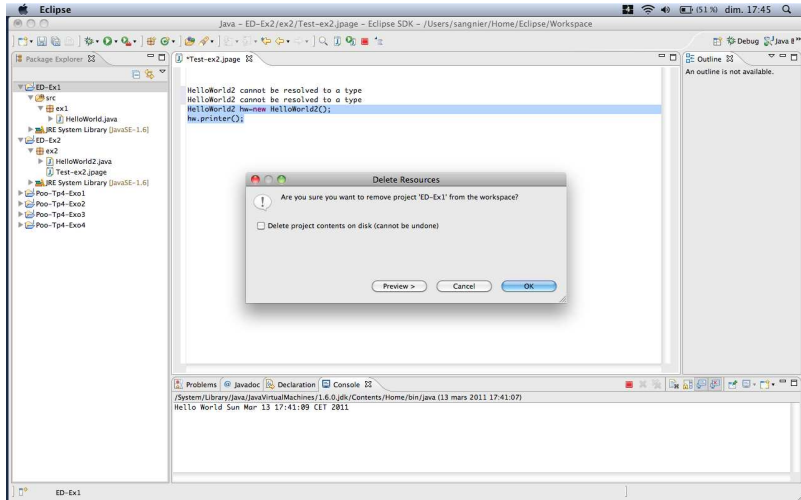


Appuyer sur le carré rouge au dessus de la console pour arrêter la phase d'exécution de la page de testeur de code

Alléger la vue package

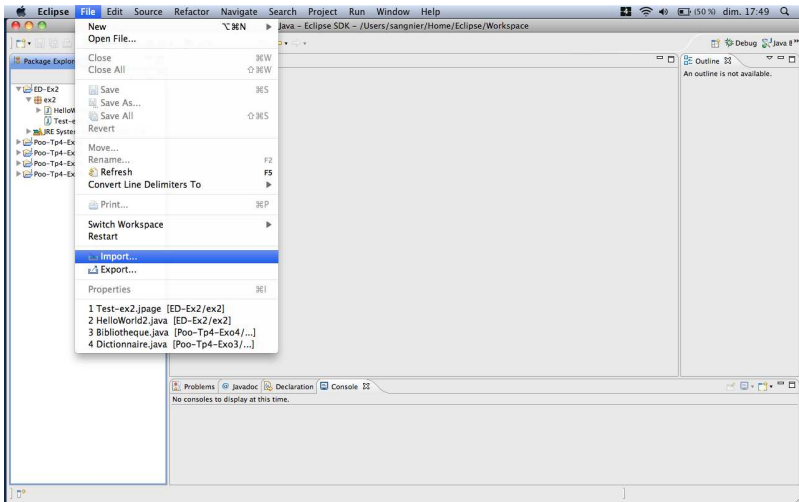


Alléger la vue package

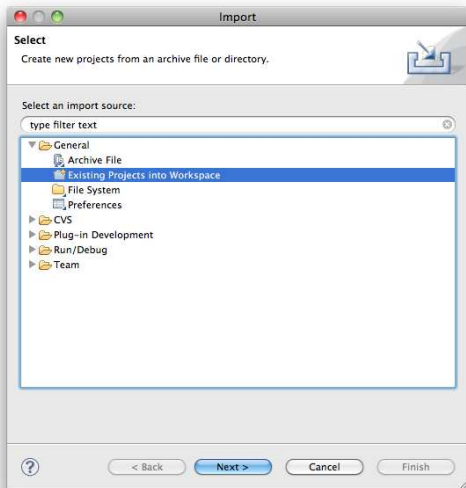


Attention : Ne pas effacer votre projet du disque si vous voulez encore le garder

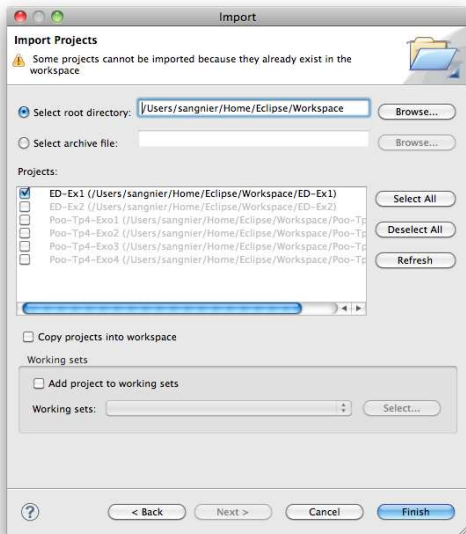
Réouvrir un projet fermé



Réouvrir un projet fermé



Réouvrir un projet fermé



Plusieurs classes dans un même package

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left displays a project structure with a package named 'ex3' containing two classes: 'Affichage.java' and 'Principale.java'. The main editor window shows the code for 'Affichage.java', which is defined as follows:

```
package ex3;

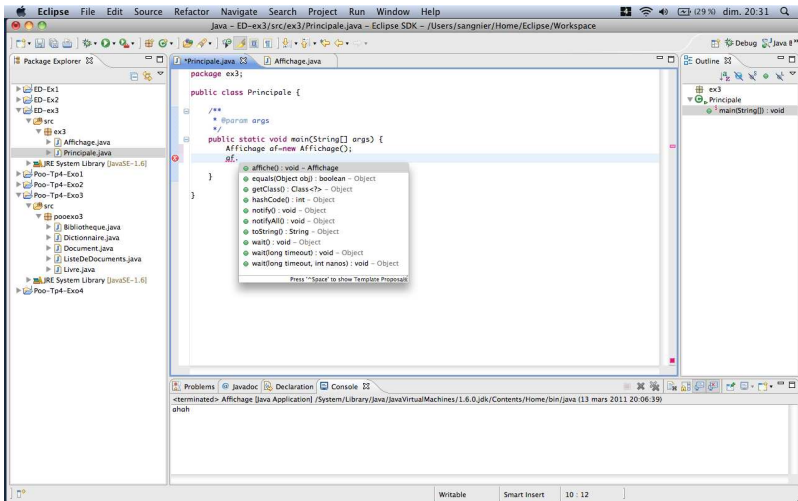
public class Affichage {

    public void affiche() {
        System.out.println("Affichage: Hello World\n");
    }

}
```

The Outline view on the right shows the package 'ex3' and the class 'Affichage' with the method 'affiche() : void'. The Console view at the bottom shows the output of the application: '<terminated> Principale (Java Application) /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (13 mars 2011 20:33:32) Affichage: Hello World'. The status bar at the bottom indicates '0 imports added.', 'Writable', 'Smart Insert', and '6 / 32'.

Plusieurs classes dans un même package



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

Et pour l'héritage

Java Class
Create a new Java class.

Source folder:

Package:

Enclosing type:

Name:

Modifiers: public default private protected
 abstract final static

Superclass:

Interfaces:

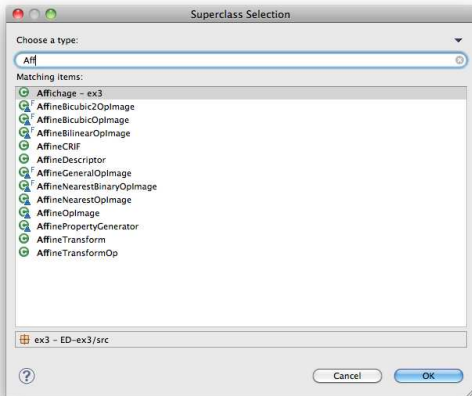
Which method stubs would you like to create?

public static void main(String[] args)
 Constructors from superclass
 Inherited abstract methods

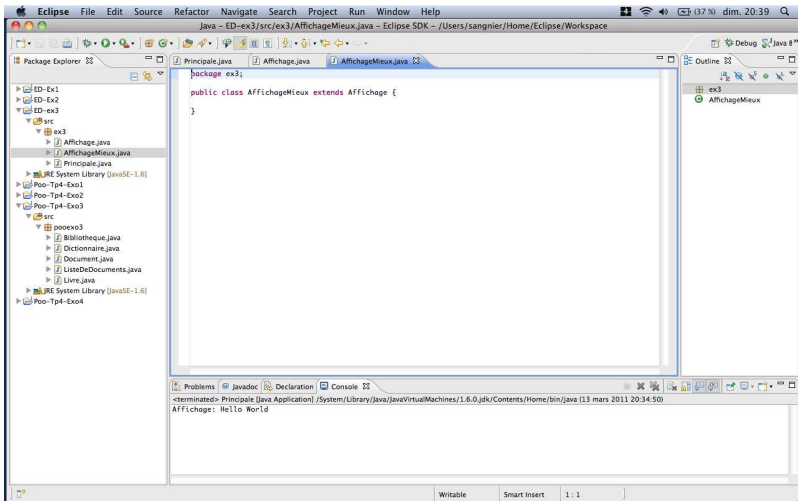
Do you want to add comments? (Configure templates and default value [here](#))

Generate comments

Et pour l'héritage



Et pour l'héritage



The screenshot shows the Eclipse IDE interface. The Package Explorer on the left displays a project structure with packages ED-Ex1, ED-Ex2, ED-Ex3, and Poo-Tp4-Exo3. The main editor window shows the source code for AffichageMieux.java, which extends Affichage. The Outline view on the right shows the class hierarchy. The Console view at the bottom shows the output of the application: "Affichage: Hello World".

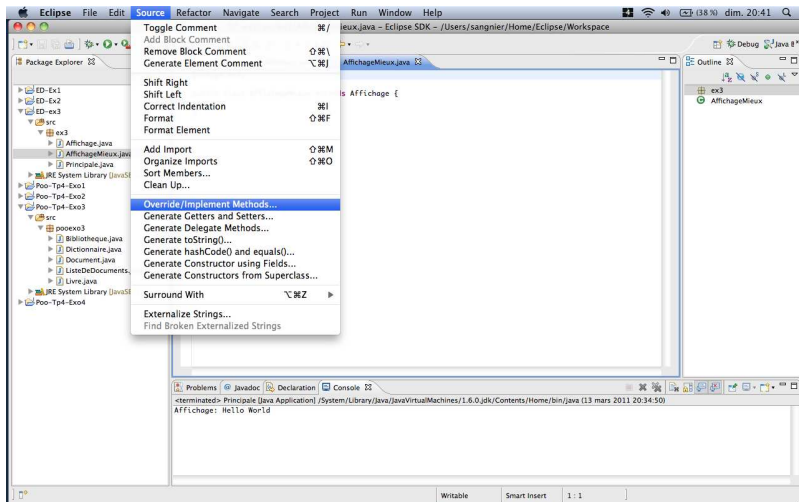
```
package ex3;

public class AffichageMieux extends Affichage {
}

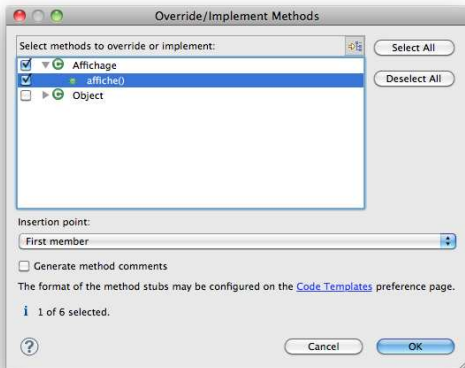

```

<terminated> Principale [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0_jdk/Contents/Home/bin/java (13 mars 2011 20:34:50)
Affichage: Hello World

Surcharge de méthodes



Surcharge de méthodes



Surcharge de méthodes

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left displays a project structure with packages ED-Ex1, ED-Ex2, ED-Ex3, and src. Under src, there is a package ex3 containing Affichage.java, AffichageMieux.java, and Principale.java. The main editor window shows the code for AffichageMieux.java:

```
package ex3;

public class AffichageMieux extends Affichage {

    @Override
    public void affiche() {
        // TODO Auto-generated method stub
        super.affiche();
    }

}
```

The Outline view on the right shows the package ex3 and the class AffichageMieux. The Console view at the bottom shows the output: <terminated> Principale [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (13 mars 2011 20:34:50) Affichage: Hello World. The status bar at the bottom indicates the editor is in Writeable mode, Smart Insert is active, and the cursor is at line 1, column 1.

Générer des getters and setters

The screenshot displays the Eclipse IDE interface. The Package Explorer on the left shows a project structure with a package named 'ex3' containing files 'Affichage.java', 'AffichageMieux.java', and 'Principale.java'. The main editor window shows the source code for 'AffichageMieux.java' in the 'ex3' package. The code defines a class 'AffichageMieux' that extends 'Affichage'. It has a public String attribute 'message' initialized to 'AffichageMieux: Hello World\n'. An '@Override' annotation is present above a public void method 'affiche()' which prints the 'message' attribute to the console.

```
package ex3;

public class AffichageMieux extends Affichage {

    public String message="AffichageMieux: Hello World\n";

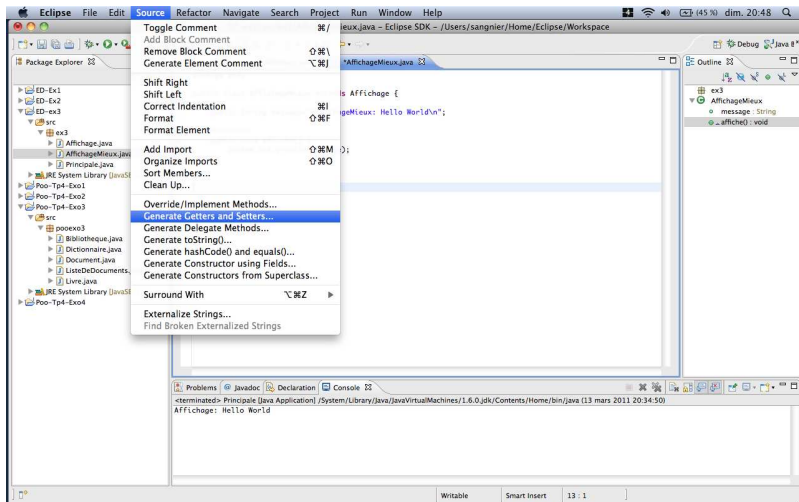
    @Override
    public void affiche() {
        System.out.println(message);
    }

}
```

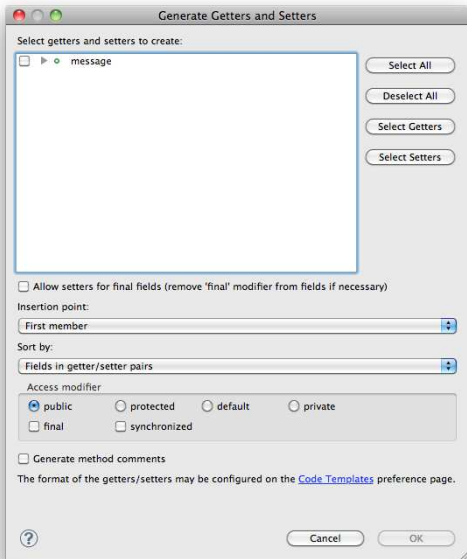
The Outline view on the right shows the class structure with 'AffichageMieux' expanded to show the 'message' attribute and the 'affiche()' method.

The Console view at the bottom shows the output of the application: '<terminated> Principale [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (13 mars 2011 20:34:50) Affichage: Hello World'.

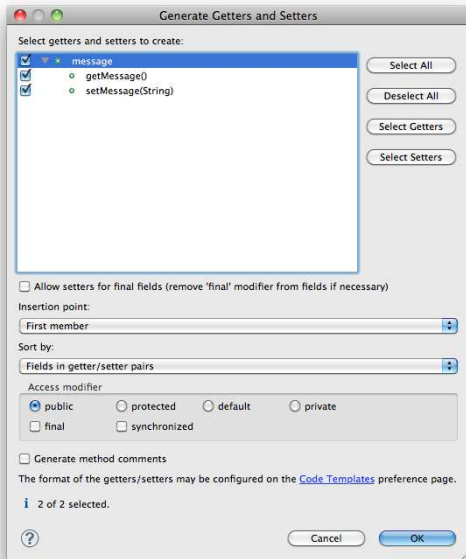
Générer des getters and setters



Générer des getters and setters



Générer des getters and setters



Générer des getters and setters

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left shows a project structure with a package named 'ex3' containing 'AffichageMieux.java' and 'Principale.java'. The main editor displays the code for 'AffichageMieux.java'.

```
package ex3;

public class AffichageMieux extends Affichage {

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public String message="AffichageMieux: Hello World\n";

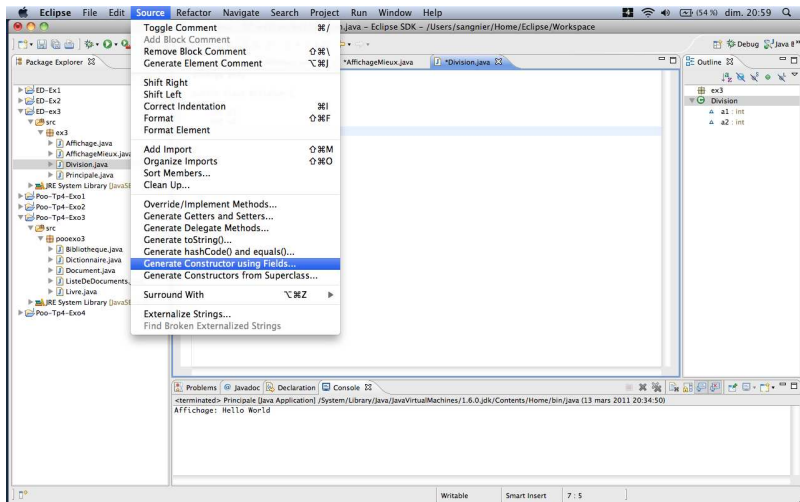
    @Override
    public void affiche() {
        System.out.println(message);
    }
}
```

The Outline view on the right shows the class structure with methods: getMessage(): String, setMessage(String): void, message: String, and _affiche(): void.

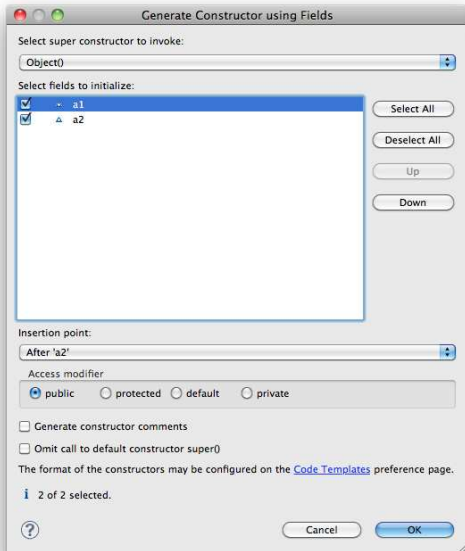
The Console view at the bottom shows the output of the application:

```
<terminated> Principale [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (13 mars 2011 20:34:50)
Affichage: Hello World
```

Générer un constructeur



Générer un constructeur



Générer un constructeur

The screenshot shows the Eclipse IDE with the following components:

- Package Explorer (Left):** Shows a project structure with packages `ED-Ex1`, `ED-Ex2`, `ED-ex3`, and `src`. Under `src`, there is a package `ex3` containing `Affichage.java`, `AffichageMieux.java`, `Division.java`, and `Principale.java`. There are also system libraries and other projects like `Poo-Tp4-Exo1` through `Exo4`.
- Editor (Center):** Displays the code for `Division.java` in the `ex3` package:

```
package ex3;

public class Division {

    int a1;
    int a2;

    public Division(int a1, int a2) {
        super();
        this.a1 = a1;
        this.a2 = a2;
    }

}
```
- Outline (Right):** Shows the class structure for `ex3`, including `Division` with attributes `a1: int`, `a2: int`, and a constructor `Division(int, int)`.
- Console (Bottom):** Shows the output of the `Affichage` class: `<terminated> Principale [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (13 mars 2011 20:34:50) Affichage: Hello World`.

Et pour les exceptions ?

Java Class
Create a new Java class.

Source folder:

Package:

Enclosing type:

Name:

Modifiers: public default private protected
 abstract final static

Superclass:

Interfaces:

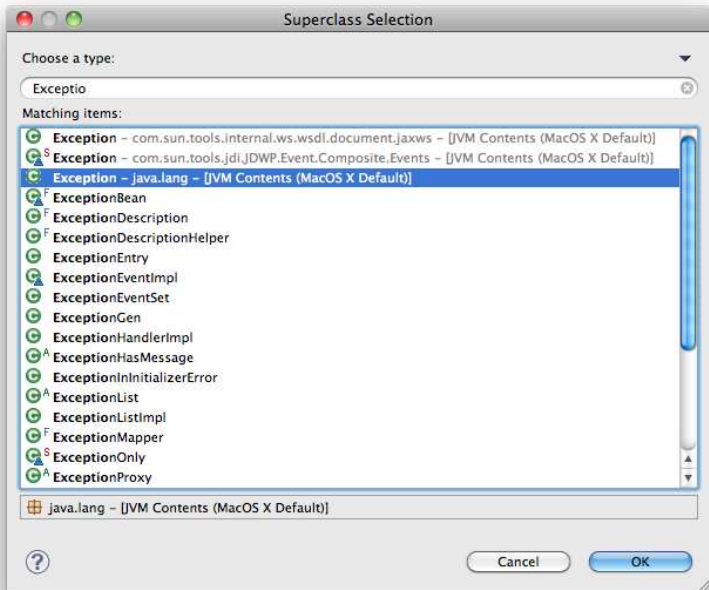
Which method stubs would you like to create?

public static void main(String[] args)
 Constructors from superclass
 Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

Generate comments

Et pour les exceptions ?



Et pour les exceptions ?

The screenshot displays the Eclipse IDE interface. The main editor window shows the following Java code:

```
package ex3;

public class DivisionParZeroException extends Exception {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

}

}
```

The Package Explorer on the left shows the project structure:

- ED-Ex1
- ED-Ex2
- ED-Ex3
 - src
 - ex3
 - Affichage.java
 - AffichageMieux.java
 - Division.java
 - DivisionParZeroException.java
 - Principale.java
- JRE System Library [javaSE-1.6]
- Poo-Tp4-Exo1
- Poo-Tp4-Exo2
- Poo-Tp4-Exo3
 - src
 - poeexo3
 - Bibliothèque.java
 - Dictionnaire.java
 - Document.java
 - ListeDeDocuments.java
 - Livre.java
- JRE System Library [javaSE-1.6]
- Poo-Tp4-Exo4

The Console at the bottom shows the output:

```
<terminated> Principale [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (13 mars 2011 20:34:50)
Affichage: Hello World
```

At the bottom of the IDE, the status bar indicates: 0 imports added. Writable Smart Insert 9 : 5

Et pour les exceptions ?

The screenshot shows the Eclipse IDE interface. The main editor displays the following Java code:

```
package ex3;

public class Division {

    int a1;
    int a2;

    public Division(int a1, int a2) {
        super();
        this.a1 = a1;
        this.a2 = a2;
    }

    public int divide() throws DivisionParZeroException{
        if(a2==0){
            throw(new DivisionParZeroException());
        }
        else{
            int res=a1/a2;

            return(res);
        }
    }
}
```

The Package Explorer on the left shows the project structure:

- ED-Ex1
- ED-Ex2
- ED-ex3
 - src
 - ex3
 - Affichage.java
 - AffichageMieux.java
 - Division.java
 - DivisionParZeroException.java
 - Principale.java
- JRE System Library [javaSE-1.6]
- Poo-Tp4-Exo1
- Poo-Tp4-Exo2
- Poo-Tp4-Exo3
 - src
 - po0ex03
 - Biographie.java
 - Dictionnaire.java
 - Document.java
 - ListeDeDocuments.java
 - Livre.java
- JRE System Library [javaSE-1.6]
- Poo-Tp4-Exo4

The Outline view on the right shows the class structure:

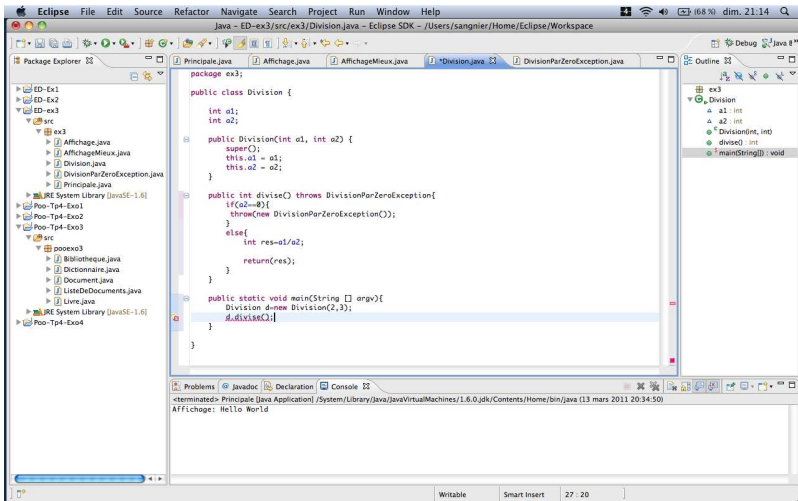
- ex3
 - Division
 - a1 : int
 - a2 : int
 - Division(int, int)
 - divide() : int

The Console view at the bottom shows the output of the application:

```
<terminated> Principale [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (13 mars 2011 20:34:50)
Affichage: Hello World
```

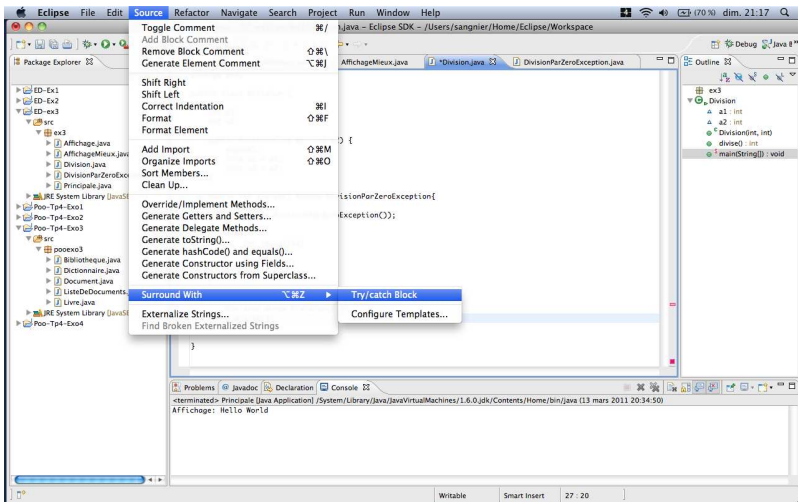
The status bar at the bottom indicates: Writable, Smart Insert, 21 / 13.

Et pour les exceptions ?



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

Insertion automatique de try ... catch



Insertion automatique de try ... catch

The screenshot shows the Eclipse IDE interface. The main editor displays the following Java code:

```
package ex3;

public class Division {

    int a1;
    int a2;

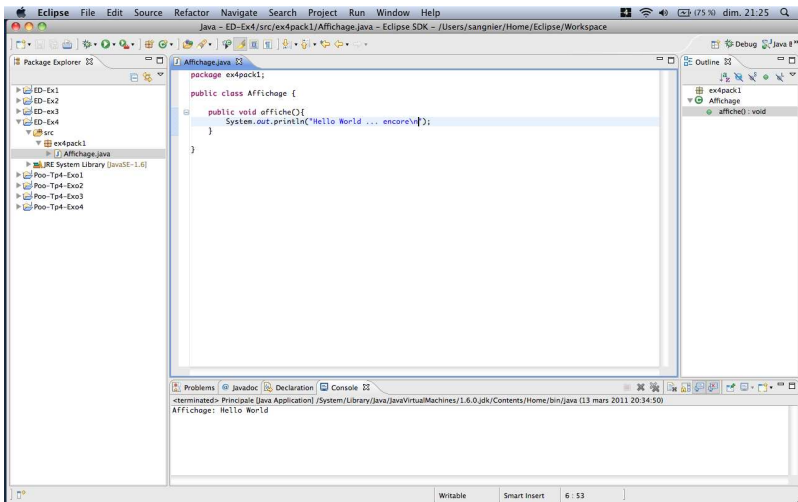
    public Division(int a1, int a2) {
        super();
        this.a1 = a1;
        this.a2 = a2;
    }

    public int divise() throws DivisionParZeroException{
        if(a2==0){
            throw(new DivisionParZeroException());
        }
        else{
            int res=a1/a2;
            return(res);
        }
    }

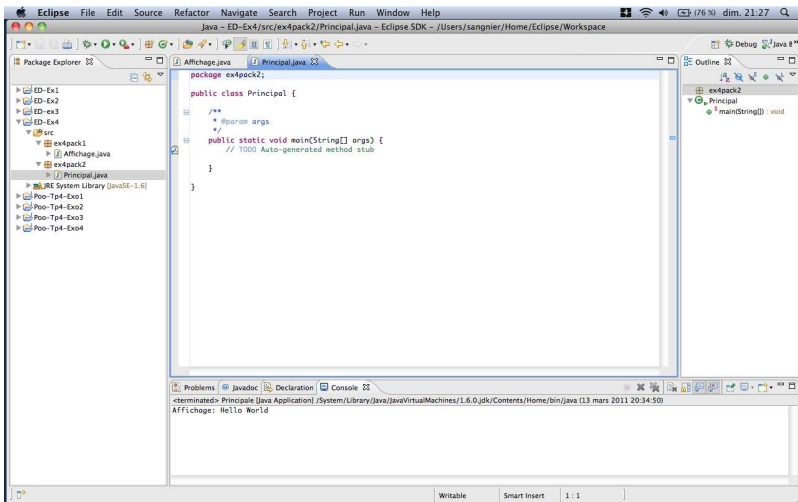
    public static void main(String [] argv){
        Division d=new Division(2,3);
        try {
            d.divise();
        } catch (DivisionParZeroException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

The IDE interface includes a Package Explorer on the left showing the project structure, an Outline on the right, and a Console at the bottom displaying the output: "Affichage: Hello World". The status bar at the bottom indicates "Writable", "Smart Insert", and "29 / 42".

Gérer plusieurs packages



Gérer plusieurs packages



Gérer plusieurs packages

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left displays a project structure with packages `ex4pack1` and `ex4pack2`. The main editor shows the source code of `Principal.java` in the `ex4pack2` package. The code includes a `main` method that calls `Affichage`. A code completion popup is visible, suggesting options like `Create class 'Affichage'`, `Create interface 'Affichage'`, and `import ex4pack1.Affichage;`. The Outline view on the right shows the `Principal` class with a `main(String[])` method. The Console at the bottom shows the output: `Affichage: Hello World`. A status bar at the bottom left indicates the error: `Affichage cannot be resolved to a type`.

```
package ex4pack2;

public class Principal {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Affichage af = new Affichage();
    }
}
```

Code completion popup options:

- Create class 'Affichage'
- Create interface 'Affichage'
- Create enum 'Affichage'
- Rename in file (R2 R)
- Add type parameter 'Affichage' to 'main(String)
- Fix project setup...

Import suggestion:

```
import ex4pack1.Affichage;
```

Console output:

```
<terminated> Principale [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0_jdk/Contents/Home/bin/java (13 mars 2011 20:34:50)
Affichage: Hello World
```

Status bar: Affichage cannot be resolved to a type | Writable | Smart Insert | 11 : 18

Gérer plusieurs packages

The screenshot displays the Eclipse IDE interface. The Package Explorer on the left shows a project structure with packages `ex4pack1` and `ex4pack2`. The main editor window shows the source code for `Principal.java` in the `ex4pack2` package, which imports `ex4pack1.Affichage` and contains a `main` method that calls `af.affiche()`. The Outline view on the right shows the class structure. The Console at the bottom displays the output: `<terminated> Principale [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (13 mars 2011 20:34:50) Affichage: Hello World`.

```
package ex4pack2;

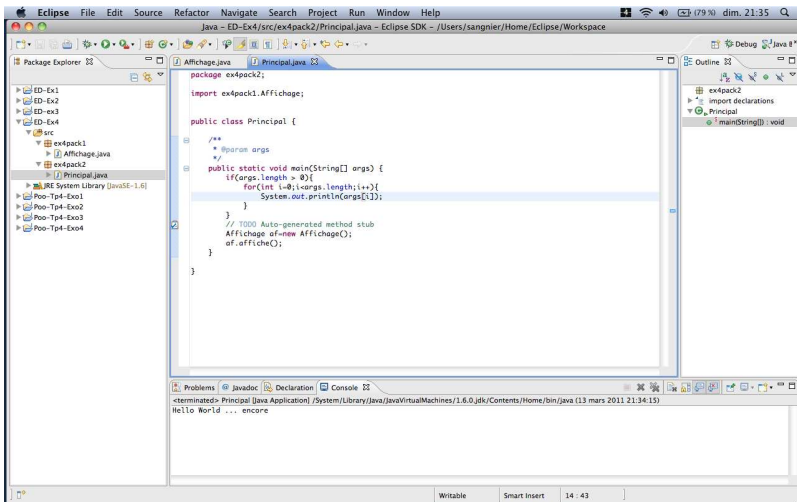
import ex4pack1.Affichage;

public class Principal {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Affichage af = new Affichage();
        af.affiche();
    }
}
```

<terminated> Principale [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (13 mars 2011 20:34:50)
Affichage: Hello World

Donner des arguments au main



The screenshot shows the Eclipse IDE interface. The Package Explorer on the left shows a project structure with packages ex4pack1 and ex4pack2. The main editor displays the code for Principal.java in the ex4pack2 package. The code includes an import statement for Affichage, a class definition for Principal, and a main method that iterates over command-line arguments and prints each one. The console at the bottom shows the output of the program: "Hello World ... encore".

```
package ex4pack2;

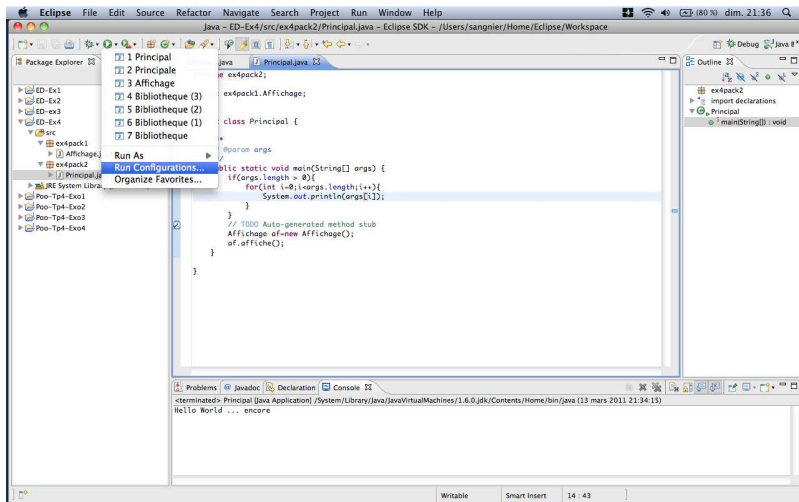
import ex4pack1.Affichage;

public class Principal {

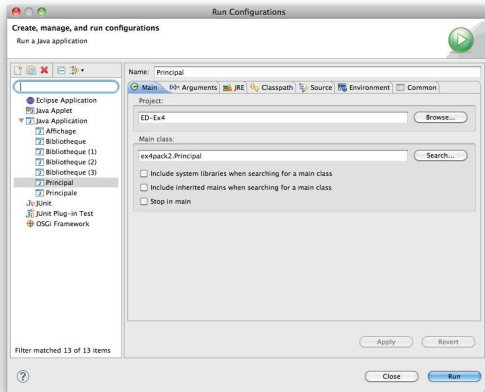
    /**
     * @param args
     */
    public static void main(String[] args) {
        if(args.length > 0){
            for(int i=0;i<args.length;i++){
                System.out.println(args[i]);
            }
        }
        // TODO Auto-generated method stub
        Affichage af=new Affichage();
        af.affiche();
    }
}
```

Console Output:
<terminated> Principal [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (13 mars 2011 21:34:15)
Hello World ... encore

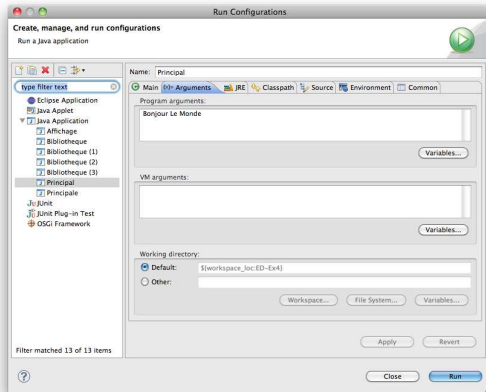
Donner des arguments au main



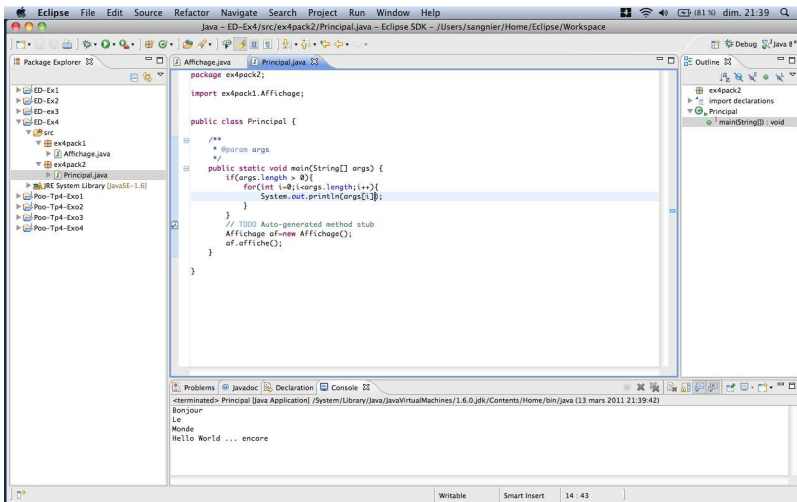
Donner des arguments au main



Donner des arguments au main



Donner des arguments au main



The screenshot displays the Eclipse IDE interface. The Package Explorer on the left shows a project structure with packages ex4pack1 and ex4pack2, and a class Principal.java. The main editor window shows the following Java code:

```
package ex4pack2;

import ex4pack1.Affichage;

public class Principal {

    /**
     * @param args
     */
    public static void main(String[] args) {
        if(args.length > 0){
            for(int i=0;i<args.length;i++){
                System.out.println(args[i]);
            }
        }
        // TODO Auto-generated method stub
        Affichage af=new Affichage();
        af.affiche();
    }
}
```

The Console window at the bottom shows the output of the program:

```
<terminated> Principal [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (13 mars 2011 21:39:42)
Bonjour
Le
Monde
Hello World ... encore
```

The status bar at the bottom indicates the editor is in 'Writable' mode, 'Smart Insert' is active, and the cursor is at line 14, column 43.

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

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left shows a project structure with a package named 'ex3' containing several Java files, including 'Division.java'. The main editor displays the source code for 'Division.java' with the following content:

```
/**
 * Cette classe implemente l'operateur mathematique de la division
 * @author sangnier
 */

public class Division {

    /**
     * Le terme gauche de la division
     */
    public int a1;
    /**
     * Le terme a droite de la division
     */
    public int a2;

    /**
     * @param a1 le terme a gauche de la division
     * @param a2 le terme a droite de la division
     */
    public Division(int a1, int a2) {
        super();
        this.a1 = a1;
        this.a2 = a2;
    }

    /**
     * Cette methode realise la division et verifie que l'on ne divise par 0
     * @return le resultat de la division
     */
}
```

The Outline view on the right shows the class structure with methods: 'a1 : int', 'a2 : int', 'Division(int, int)', 'divise() : int', and 'main(String[]) : void'. The Console view at the bottom shows the output of the Javadoc generation process:

```
<terminated> Javadoc Generation
Building index for all classes...
Generating /Users/sangnier/Home/Eclipse/Workspace/ED-ex3/doc/allclasses-frame.html...
Generating /Users/sangnier/Home/Eclipse/Workspace/ED-ex3/doc/allclasses-noframe.html...
Generating /Users/sangnier/Home/Eclipse/Workspace/ED-ex3/doc/index.html...
Generating /Users/sangnier/Home/Eclipse/Workspace/ED-ex3/doc/help-doc.html...
Generating /Users/sangnier/Home/Eclipse/Workspace/ED-ex3/doc/stylesheet.css...
```

Générer la javadoc avec Eclipse

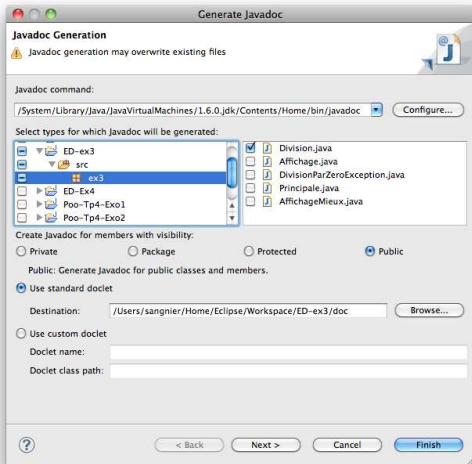
The screenshot shows the Eclipse IDE interface. The 'Project' menu is open, and 'Generate Javadoc...' is selected. The background shows a Java project structure in the Package Explorer and the source code of a 'Division' class in the editor. The console at the bottom shows the output of the Javadoc generation process.

```
public class Division {  
    /**  
     * Le terme gauche de la division  
     */  
    public int a1;  
    /**  
     * Le terme a droite de la division  
     */  
    public int a2;  
    /**  
     * @param a1 le terme a gauche de la division  
     * @param a2 le terme a droite de la division  
     */  
    public Division(int a1, int a2) {  
        super();  
        this.a1 = a1;  
        this.a2 = a2;  
    }  
    /**  
     * Cette méthode realise la division et verifie que l'on ne divise par 0  
     * @return le resultat de la division  
     */  
}
```

Console Output:

```
<terminated> Javadoc Generation  
Building index for all classes...  
Generating /Users/sangnier/Home/Eclipse/Workspace/ED-ex3/doc/allclasses-frame.html...  
Generating /Users/sangnier/Home/Eclipse/Workspace/ED-ex3/doc/allclasses-noframe.html...  
Generating /Users/sangnier/Home/Eclipse/Workspace/ED-ex3/doc/index.html...  
Generating /Users/sangnier/Home/Eclipse/Workspace/ED-ex3/doc/help-doc.html...  
Generating /Users/sangnier/Home/Eclipse/Workspace/ED-ex3/doc/stylesheet.css...
```


Générer la javadoc avec Eclipse



Personnaliser Eclipse

The screenshot displays the Eclipse IDE interface. The top menu bar includes 'Eclipse', 'File', 'Edit', 'Source', 'Refactor', 'Navigate', 'Search', 'Project', 'Run', 'Window', and 'Help'. The title bar shows the current file path: 'Java - ED-ex3/src/ex3/Division.java - Eclipse SDK - /Users/sangnier/Home/Eclipse/Workspace'. The left sidebar contains a 'Préférences...' menu and a project tree for 'ED-Ex2' and 'ED-Ex3'. The main editor window shows the source code for the 'Division' class, which implements a mathematical division operator. The code includes comments in French, two integer attributes 'a1' and 'a2', and a 'Division' constructor. The bottom status bar indicates 'Writable', 'Smart Insert', and '16 : 40'.

```
/**
 * Cette classe implémente l'opérateur mathématique de la division
 * @author sangnier
 */
public class Division {

    /**
     * Le terme gauche de la division
     */
    public int a1;
    /**
     * Le terme à droite de la division
     */
    public int a2;

    /**
     * @param a1 le terme à gauche de la division
     * @param a2 le terme à droite de la division
     */
    public Division(int a1, int a2) {
        super();
        this.a1 = a1;
        this.a2 = a2;
    }

    /**
     * Cette méthode réalise la division et vérifie que l'on ne divise par 0
     * @return le résultat de la division
     */
}
```

