Université de la Méditerranée (Aix–Marseille Université)
Faculté des Sciences de Luminy

Installation et utilisation d'*eclipse* pour le développement en Java

La documentation en ligne d'*eclipse* est abondante mais son approche n'est pas immédiate. Sans souci d'exhaustivité, nous donnons ici quelques indications pour débuter rapidement dans l'utilisation de cet excellent outil de développement.

A la date du 24 mai 2011, les versions courantes des logiciels en question sont

- *Java JDK 1.6.0 update 24*
- *Eclipse IDE Helios SR2*

Les versions décrites ici ne sont peut-être pas exactement celles-là mais elles leur sont fonctionnellement équivalentes

1. Se procurer le kit de développement Java

Eclipse ne contient ni le compilateur Java ni les autres outils basiques. Pour développer des programmes en Java il faut donc installer au préalable un kit de développement. Nous conseillons celui de Sun Microsystems, la maison mère de Java, qui est complet, à jour (par définition) et ~~libre~~ gratuit.

Vous pouvez l'obtenir chez Sun : site *java.sun.com*, menu *Downloads > Popular Downloads*, rubrique *Java for Developers*. Le produit à télécharger s'appelle, lors de la publication de cette notice, *Java SE 6 Update 25* (cliquer sur le bouton *Download JDK*).

> *Attention, ne confondez pas le JDK (Java Development Kit) avec le JRE (Java Runtime Environment), appelé parfois « plugin Java », qui ne contient que le nécessaire pour exécuter les programmes Java. Ne vous occupez pas de télécharger le JRE, à l'intérieur du JDK il y en a un exemplaire.*

Le fichier téléchargé est un installateur auto-extractible : une fois chargé, il suffit de le lancer et de suivre les instructions qui s'affichent. Au besoin, des renseignements supplémentaires sur l'installation du JDK sont donnés sur le site de Sun, aussi bien pour Windows que pour Linux.

Une fois l'installation terminée, vérifiez sa réussite en tapant « `java -version` » dans une console de commandes. Vous devez obtenir un message vous annonçant le numéro de version de la machine Java mise en place. Dans le cas de Windows cela ressemblera à ceci :

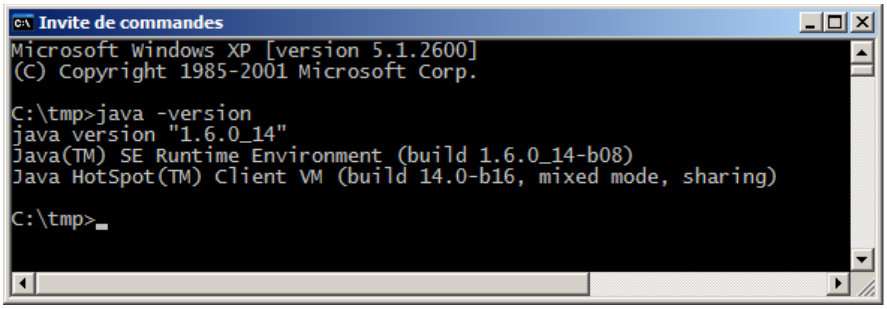

Fig. 1

Pour développer des programmes en Java il vous faut disposer également de la documentation de l'*API* (*Application Programmer Interface,* c'est-à-dire le volumineux ensemble de paquetages, classes, méthodes et variables qui constituent la bibliothèque système). Vous pouvez la consulter en ligne ou bien la télécharger depuis le site *java.sun.com*, menu *Downloads > Popular Downloads*, rubrique *Java for Developers*, produit *Java SE 6 Documentation* (quel que soit votre système d'exploitation, le fichier s'appelle quelque chose comme `jdk-6u10-docs.zip` et pèse dans les 60 Mo).

> *Note (cas de Windows).* Si vous souhaitez pouvoir employer le compilateur et les autres outils Java en dehors d'*eclipse*, c'est-à-dire en tapant des commandes dans une console *Invite de commandes*, alors vous devez procéder à la manipulation supplémentaire suivante : repérer le répertoire ® d'installation de Java et ajouter le chemin ®`\bin` dans la définition de la variable *Path*. Si vous avez laissé l'installateur de Java faire à sa guise, ® doit être quelque chose comme `C:\Program Files\Java\jdk1.6.0_04`.
> Vous pouvez examiner et modifier la valeur de la variable *Path* en cliquant avec le bouton droit sur l'icône du *Poste de travail*, puis *Propriétés > Avancé > Variables d'environnement > Variables système* ; sélectionner la ligne *Path* puis faire *Modifier*.
>
> *Dans le cas de Linux*, une manipulation analogue est nécessaire après l'installation du *JDK*. Nous ne l'expliquons pas car elle fait partie des opérations courantes sur ce système.

2. Télécharger *eclipse*

Eclipse est un logiciel libre que vous pouvez télécharger depuis le site *www.eclipse.org*, onglet *Downloads*. Le produit qui nous intéresse est *Eclipse IDE for Java Developers (92 MB)*.

Le fichier téléchargé se nomme

- dans le cas de Windows : `eclipse-java-galileo-win32.zip`
- dans le cas de Linux : `eclipse-java-galileo-linux-gtk.tar.gz`
- dans le cas de Max OS X : `eclipse-java-galileo-macosx-carbon.tar.gz`

Nous ne vous conseillons pas de télécharger une version française d'*eclipse*. Il existe bien des *plugin* de francisations de l'interface, mais outre le fait qu'elles sont assez imparfaites, elles servent surtout à vous empêcher d'utiliser la dernière version du logiciel.

3. Installer *eclipse*

A partir d'ici, les explications sont communes aux divers systèmes d'exploitation, ou bien ne concernent que Windows XP et Vista.

Pour installer *eclipse* il suffit de décompresser l'archive *zip* ou *tar.gz* téléchargée. Cela crée un dossier, nommé `eclipse`, que nous vous conseillons de placer aussi haut que vous le pouvez dans la hiérarchie de fichiers de votre système. *Dans la suite de cette note nous supposons que vous avez fait ainsi et que vous avez donc un dossier nommé* `C:\eclipse`.

Pour faciliter le lancement d'*eclipse* créez un raccourci vers le fichier `C:\eclipse\eclipse.exe` et placez-le sur le bureau, dans le menu démarrer ou ailleurs, selon vos goûts.

> *Note.* L'installation d'*eclipse* est donc bien plus légère que celle de beaucoup de logiciels ; en particulier, sous Windows elle ne produit pas d'inscription dans la base de registres. Par conséquent, pour désinstaller complètement *eclipse* il suffira, le moment venu, de mettre à la corbeille le dossier `C:\eclipse` et les divers espaces de travail (dossiers `workspace`, voir ci-dessous) créés ultérieurement.

4. Premier lancement d'*eclipse*

Lancez *eclipse*, par exemple en double-cliquant sur le raccourci que vous venez de créer. Au bout de quelques instants, on vous demandera de situer l'espace de travail dans lequel seront vos fichiers. Si vous travaillez sur un ordinateur partagé il est conseillé de

mettre l'espace de travail dans votre dossier *Documents* (sur Windows XP il se trouve dans le dossier *Documents and Settings*). Si vous êtes le seul utilisateur de votre système, mettez l'espace de travail où bon vous semble.

> *Sauf indication contraire, les fichiers sources de vos programmes se trouveront dans l'espace de travail. Il est donc important de se souvenir de l'emplacement de ce dernier pour accéder aux sources (par exemple, pour les transporter, les copier, etc.)*

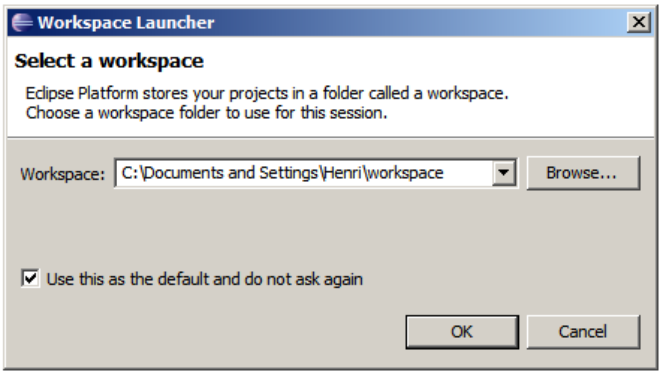

Fig. 2

Si vous cochez la case « *Use this as the default and do not ask again* » *eclipse* ne vous posera plus cette question (mais il y a toujours un moyen pour changer ultérieurement l'espace de travail : *File* > *Switch Workspace* > *Other*...).

Au bout de quelques instants (la première fois ce n'est pas très rapide) vous obtenez un écran qui présente le produit, comme ceci :

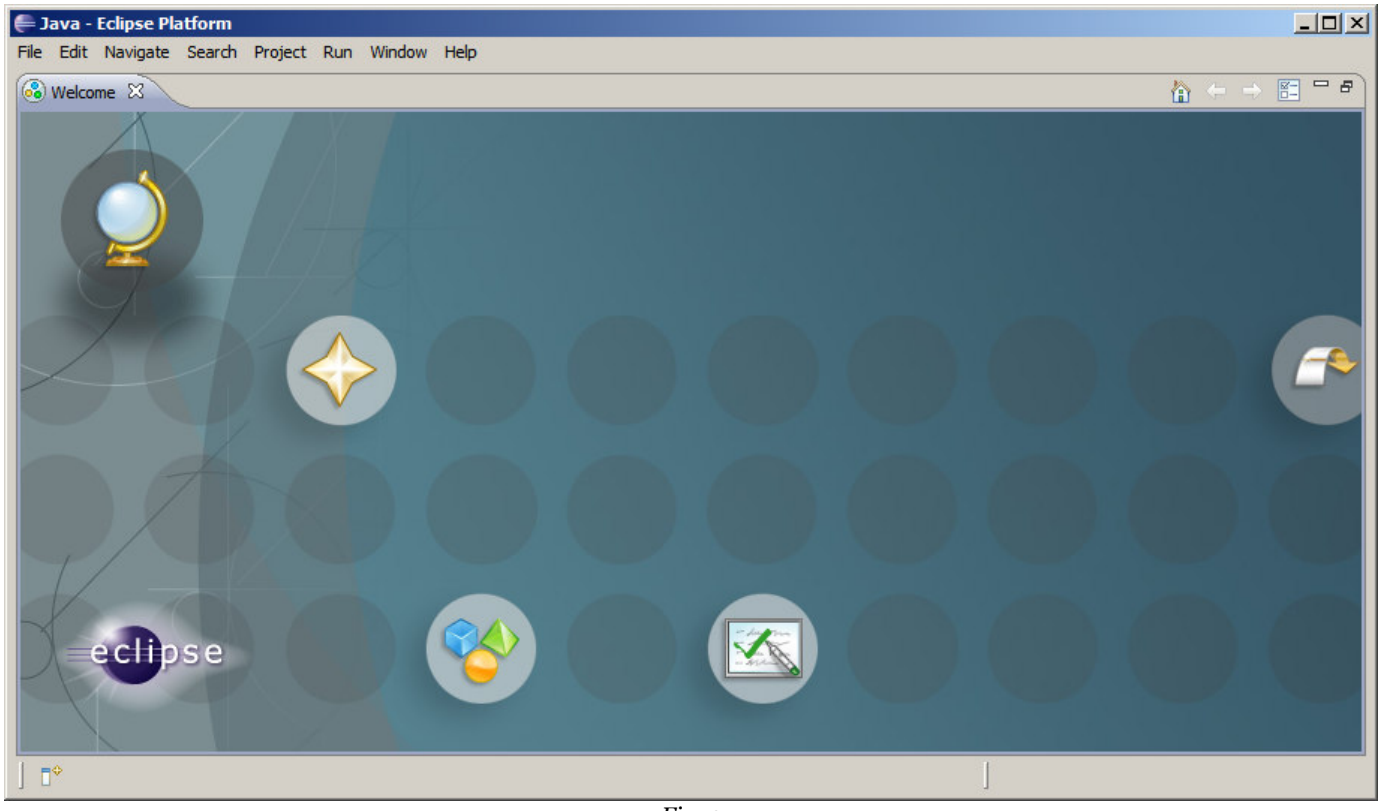

Fig. 3

Vous pouvez feuilleter cette présentation, elle est faite pour cela. Quand vous en aurez assez, cliquez sur le lien *Workbench* (la flèche représentée à droite de l'écran). Le contenu de la fenêtre devient tout de suite beaucoup plus sérieux :

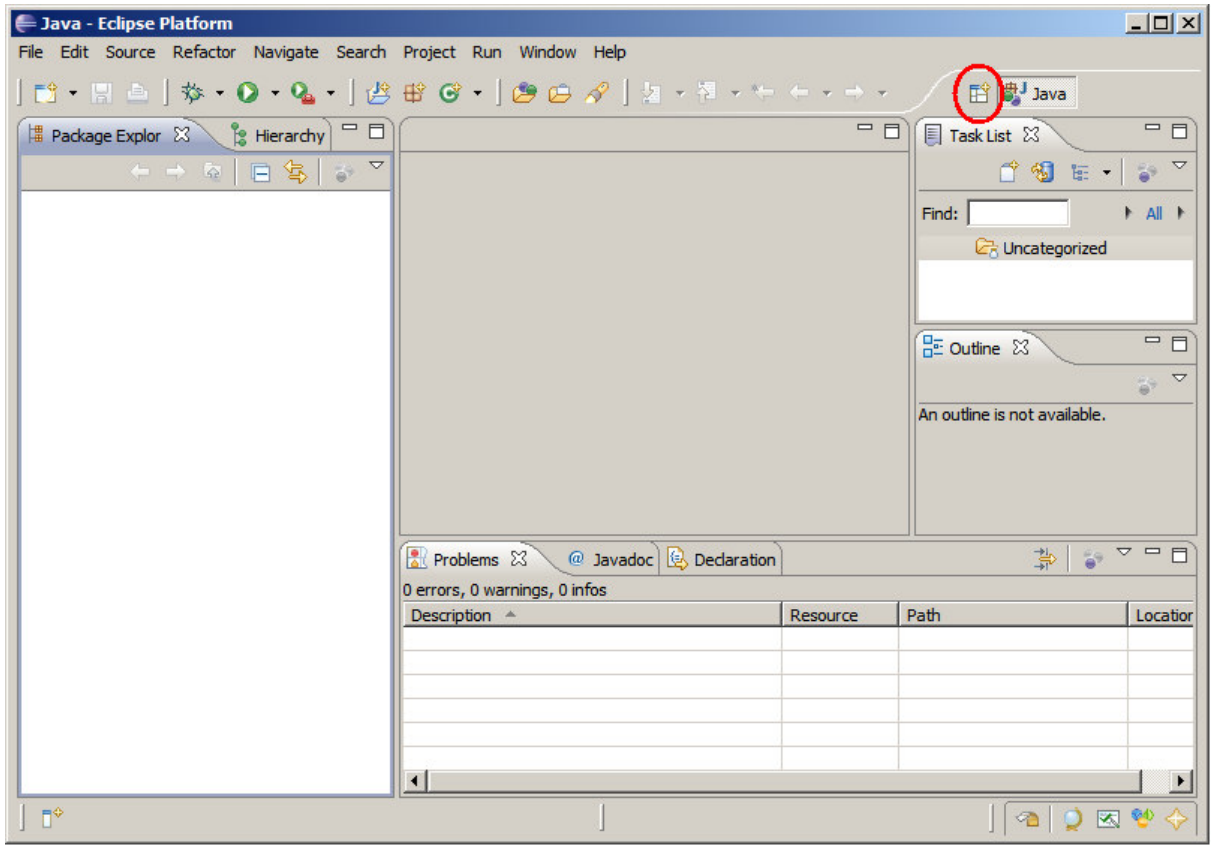

Fig. 4

5. Configurer *eclipse* pour faire du Java

Eclipse est un environnement qui permet une grande variété d'activités de développement (pour vous en convaincre, faites un tour chez *eclipse plugin central*, chez *eclipse plugins* ou bien sur le site francophone *eclipsetotale.com*). En standard, *eclipse* est prêt pour le développement en Java, encore faut-il veiller à ce que la *perspective* (c'est-à-dire l'arrangement des vues montrées à l'écran) soit celle qui convient le mieux à Java. Si ce n'est pas le cas, agissez sur la petite icône en haut à droite – cerclée de rouge sur la figure 4 –, étiquetée *Open perspective* et choisissez *Java*.

Fermez les vues *Task List* et *Outline* (à droite) ; pour afficher la structure des classes, la vue *Package explorer* (à gauche) suffit. Vous obtenez un cadre de travail tout à fait commode pour développer en Java :

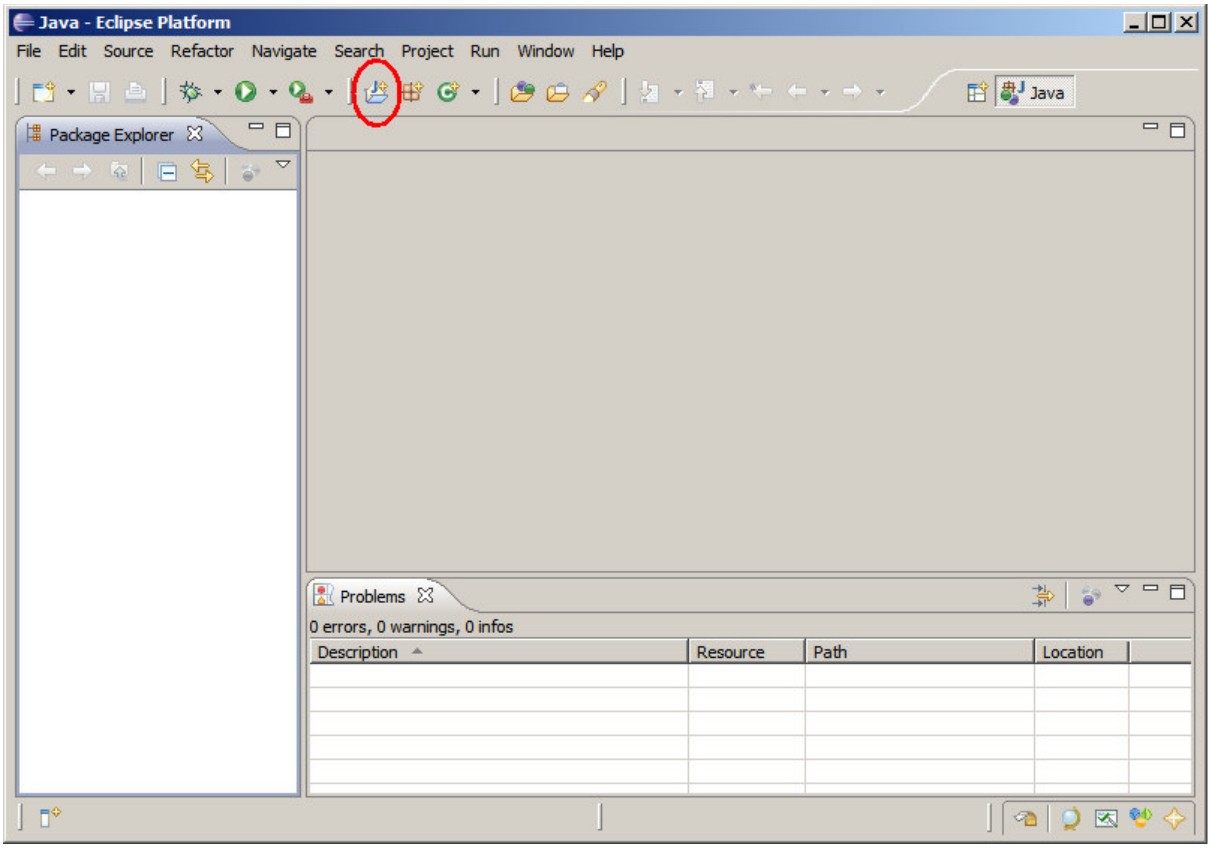

Fig. 5

6. Développer un programme Java

Pour commencer, créer un *projet*.

> *Note pour les étudiants.* Ne vous sentez pas obligés de créer un nouveau projet chaque fois que vous commencez un nouvel exercice de programmation : vous pouvez très bien avoir un seul projet, contenant tous les exercices que vous faites dans le cadre d'un enseignement. D'autant plus que cela ne vous empêchera pas de bien ranger vos fichiers : un projet peut contenir plusieurs *packages* java (qui se traduiront dans le système de fichiers par des répertoires différents).

Pour créer un projet, cliquez sur le premier des boutons d'assistants Java (cerclé de rouge sur la figure 5) etiqueté *« New Java Project »*. Vous obtenez le panneau *New Java Project* où, au minimum, vous devez donner un nom pour votre projet :

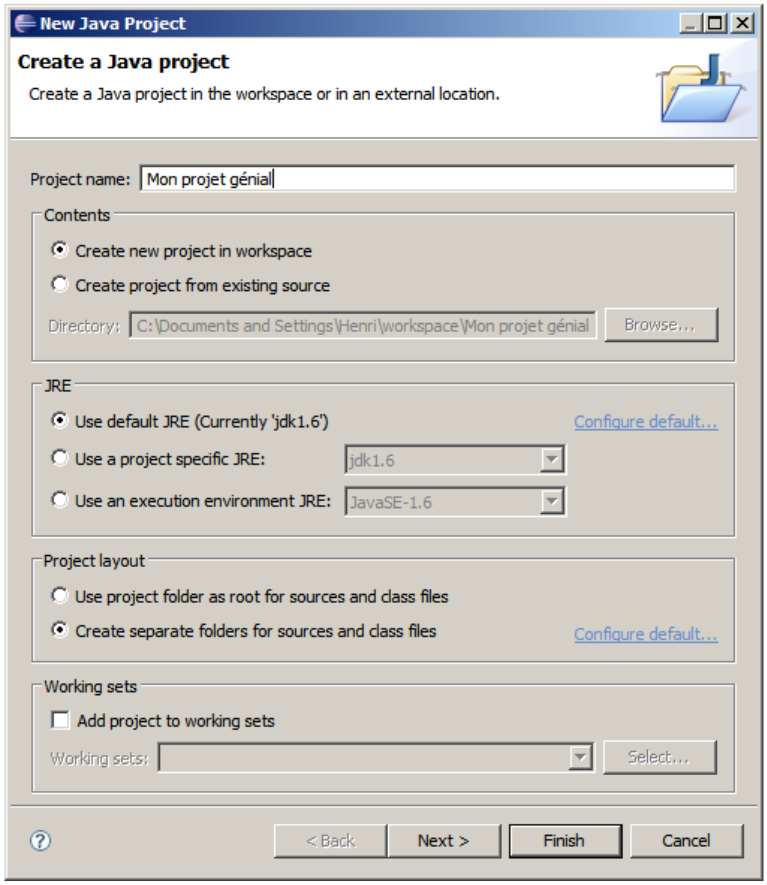

Fig. 6

La méthode rapide consiste à donner un nom de projet (si possible, moins bête que *Mon projet génial*...) et cliquer sur le bouton *Finish*. Notez que les autres « questions » posées dans ce panneau sont intéressantes. La troisième, notamment, permet de conserver *séparément* les fichiers sources (précieux) et les fichiers classes (qu'en cas de perte on peut toujours refaire).

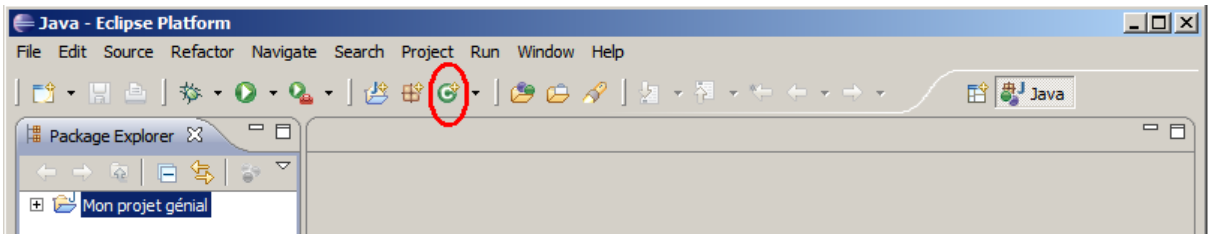

Fig. 7

Dans un projet sérieux nous commencerions par créer des packages (deuxième bouton des assistants Java, « *New Java Package* »). Mais, puisque nous débutons, allons à l'essentiel et ajoutons directement une ou plusieurs classes au projet : c'est le troisième des boutons d'assistants Java, « *New Java Class* » (cerclé de rouge dans la figure 7), qui fait cela. La méthode rapide consiste à donner le nom de la classe et cocher la case étiquetée *public static void main(String[] args)* :

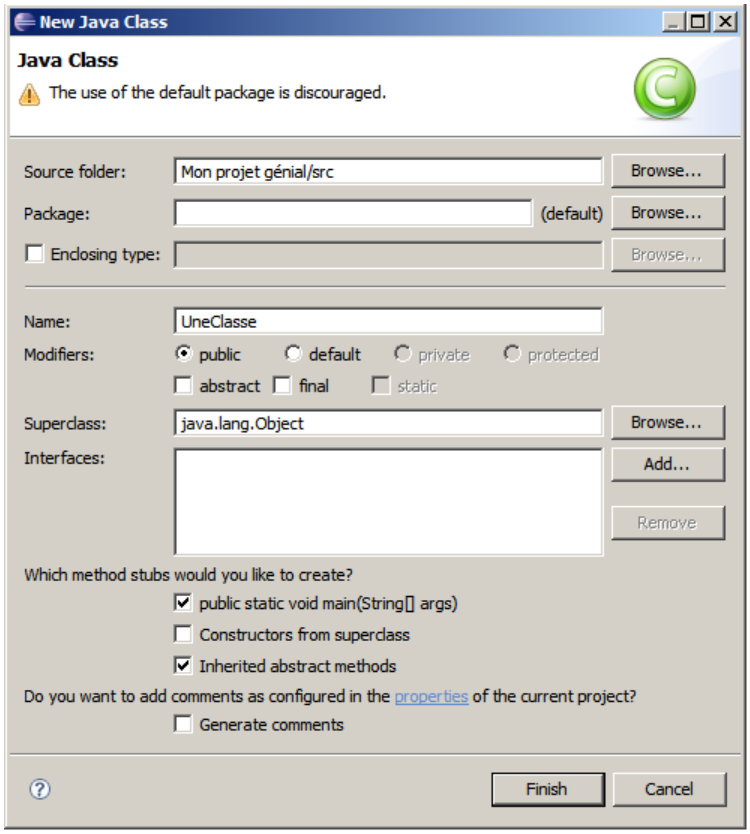

Fig. 8

Notez qu'*eclipse* critique notre démarche, nous indiquant que *l'emploi du package par défaut (sans nom) est découragé*. Cela ne fait rien, nous construisons ici une application de débutant.

Eclipse crée alors un fichier source contenant une classe rudimentaire, correcte mais creuse, que vous n'avez plus qu'à compléter pour en faire le programme voulu :

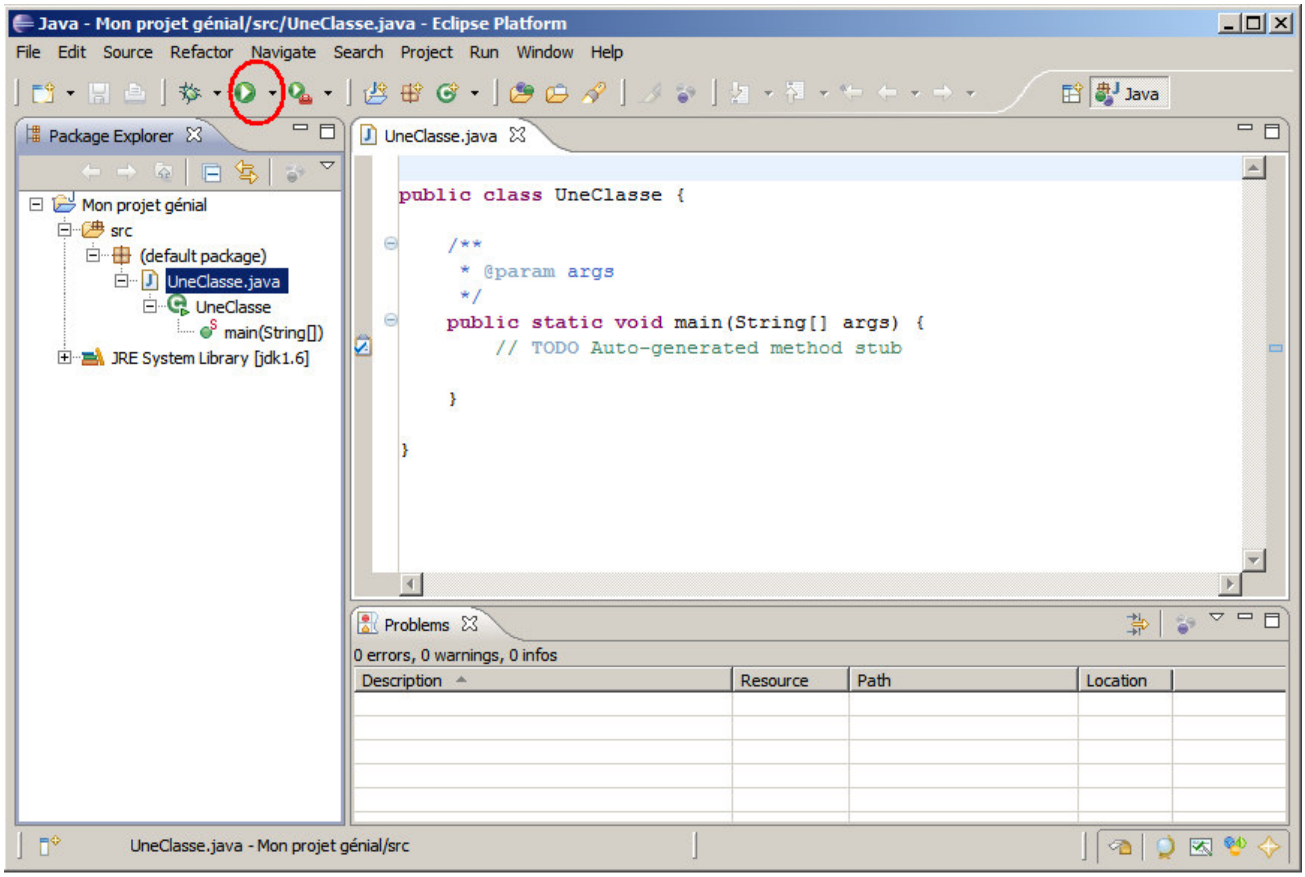

Fig. 9

Note. Lorsqu'un commentaire contient l'expresson *TODO*, *eclipse* affiche une marque bleue dans la marge qui permet de se rendre rapidement à cet endroit. C'est très pratique pour retrouver dans les gros fichiers ces commentaires qui signalent des morceaux en chantier.

Pour essayer votre programme vous allez taper le classique `System.out.println("Bonjour à tous!");` à l'intérieur de la fonction `main`. Au fur et à mesure que vous tapez, remarquez comment :

- la vue *Package Explorer* montre les packages (répertoires) qui composent votre projet, les classes que ces paquetages contiennent, les membres de ces classes, etc. Bien entendu, double cliquer sur une de ces entités vous positionne dessus dans le texte source.
- si vous marquez une pause lorsque vous tapez un point, *eclipse* vous montre la liste de ce que vous pouvez taper ensuite,
- si vous laissez traîner le curseur sur un identificateur, *eclipse* affiche la documentation correspondante,
- si vous faites une faute *eclipse* vous la signale immédiatement et, dans le cas d'erreurs sémantiques, vous suggère des corrections,
- le simple fait de sauver le programme en provoque la compilation

Pour exécuter le programme assurez-vous que la vue éditeur contient une classe exécutable (c'est-à-dire une classe *publique* avec une méthode `public static void main(String[] args);`) et alors activez la commande *Run as* > *Java Application* du menu attaché au bouton cerclé de rouge sur la figure 9 :

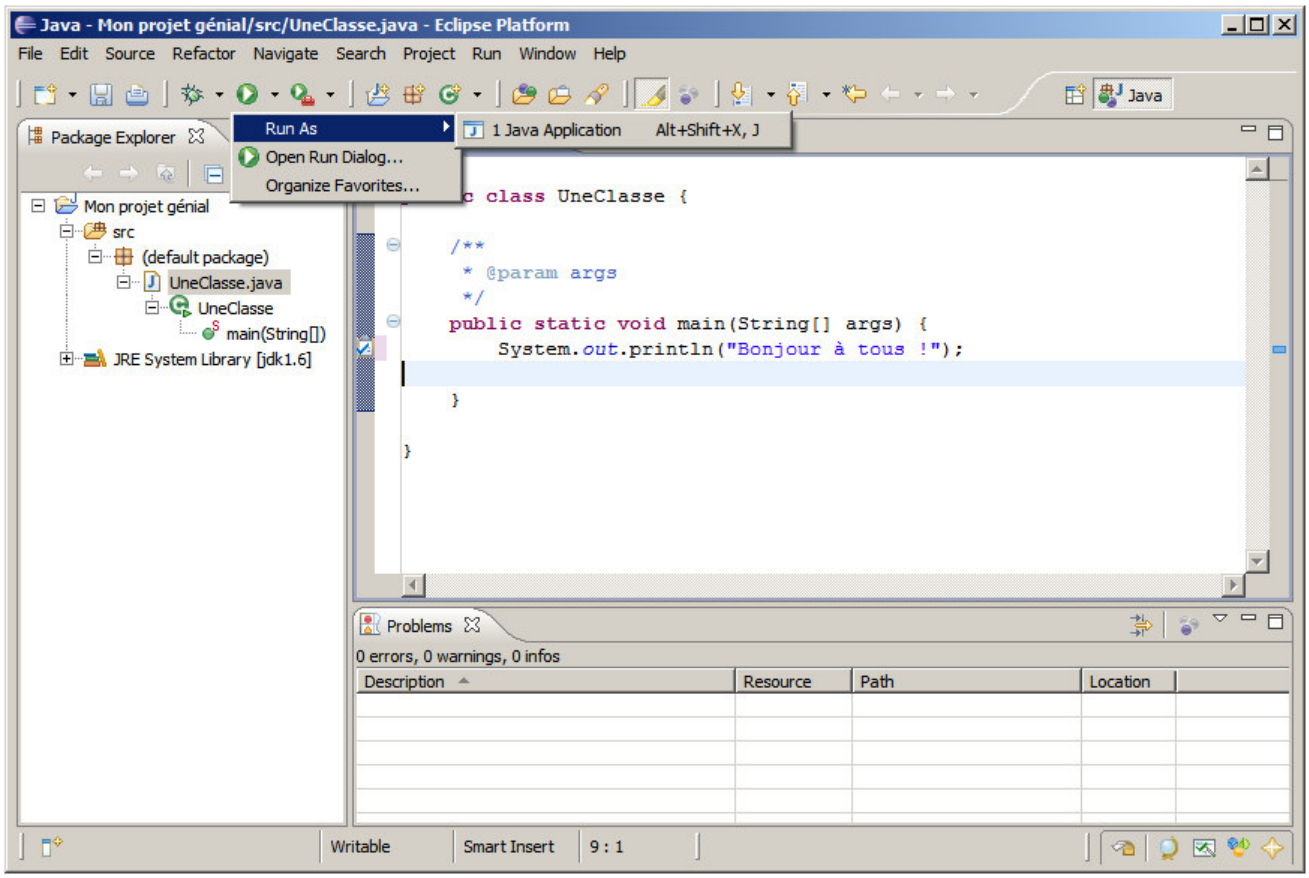

Fig. 10

L'application s'exécute et, si des sorties sont à afficher, une vue *Console* apparaît au-dessous de la vue éditeur :

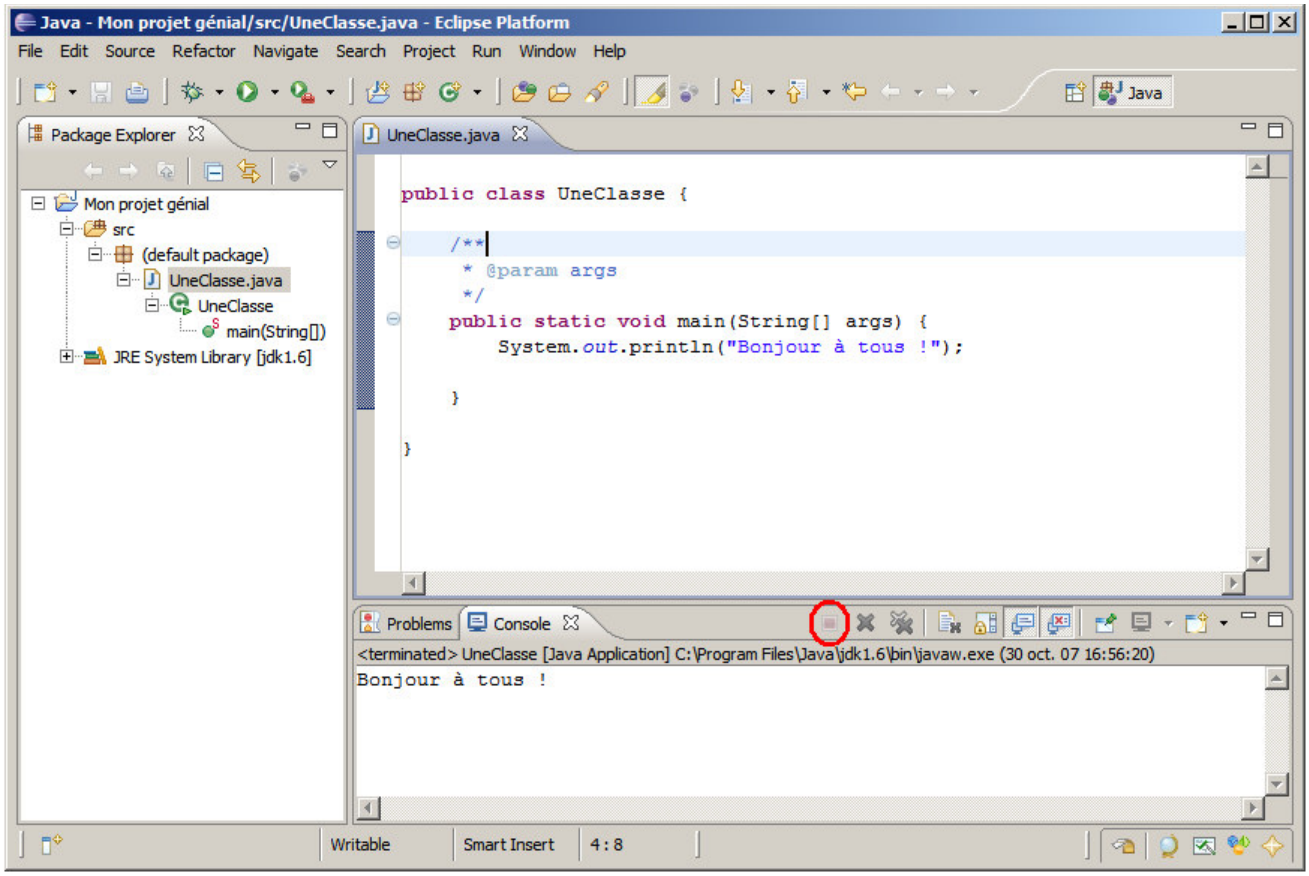

Fig. 11

Notez que dans la vue *Console* il y a un bouton – cerclé de rouge sur la figure 11 – qui permet d'arrêter une application qui bouclerait indéfiniment. Ce bouton est rouge quand l'application est vivante, gris (estompé) lorsque l'application est morte.

7. Où sont mes fichiers sources ?

Cette question se pose par exemple lorsque, après avoir développé une application dans *eclipse*, vous souhaitez récupérer vos fichier sources pour les amener sur un autre système, les compiler dans un autre environnement ou tout simplement les ranger dans vos archives.

La réponse se trouve dans les figures 2 et 6 : si lors de la création du projet vous avez laissé l'option par défaut « *Create new project in workspace* » (cf. figure 6) alors les sources, rangés dans des dossiers correspondant aux packages, sont dans le dossier *workspace*, lui-même placé à l'endroit que vous avez indiqué au lancement d'*eclipse* (cf. figure 2).

8. Comment amener dans *eclipse* des fichiers créés ailleurs ?

Deux cas possibles : ces fichiers forment déjà un projet *eclipse* (par exemple créé sur un autre système), ou bien il ne s'agit que d'un ensemble de fichiers sources en vrac.

1. Vous avez déjà un projet eclipse

Copiez le dossier du projet *eclipse* où vous voulez (par exemple dans le dossier *workspace*, mais ce n'est pas une obligation), puis faites la commande *File* > *Import...* Ensuite, choisissez *General* puis *Existing Projects into Workspace*. Le projet que vous venez d'importer apparaît dans la fenêtre *Package explorer*, c'est terminé.

2. Vous n'avez qu'un ensemble de fichiers sources Java

Prenez un projet qui existe déjà, ou bien créez un noveau projet. Ensuite :

- soit, à l'aide de la commande *File* > *Import...* > *General* > *File system*, vous naviguez à la recherche du(des) fichier(s) en question et vous les *importez* dans ce projet,
- soit, plus simplement :
  - vous copiez les fichiers dans le dossier où sont les sources d'un des projets connus dans eclipse

- vous sélectionnez ce projet dans la vue *Package Explorer*
- vous exécutez la commande *File* > *Refresh*

9. Déboguer les programmes

Un programme « bogué » est un programme qui ne donne pas les résultats qu'il devrait. « Déboguer » un programme c'est chercher les erreurs de programmation à l'origine de tels dysfonctionnements. Pour aider le programmeur dans cette recherche, *eclipse* offre un mode *debug* permettant, entre autres choses :

- la pose de marques, appelées *points d'arrêt*, sur des lignes du programme source, de telle manière que l'exécution s'arrêtera lorsque ces instructions seront atteintes,
- lors de tels arrêts, l'examen des valeurs qu'ont alors les variables locales et les membres des objets,
- à partir de là, l'exécution du programme pas à pas (c'est-à-dire ligne à ligne)

Pour déboguer simplement un programme il suffit de poser un point d'arrêt au début de l'endroit qu'on souhaite examiner en détail. Pour cela il faut double-cliquer dans la marge, à gauche de la ligne en question, ce qui fait apparaître un disque bleu (cerclé de rouge dans la figure 12) qui représente le point d'arrêt.

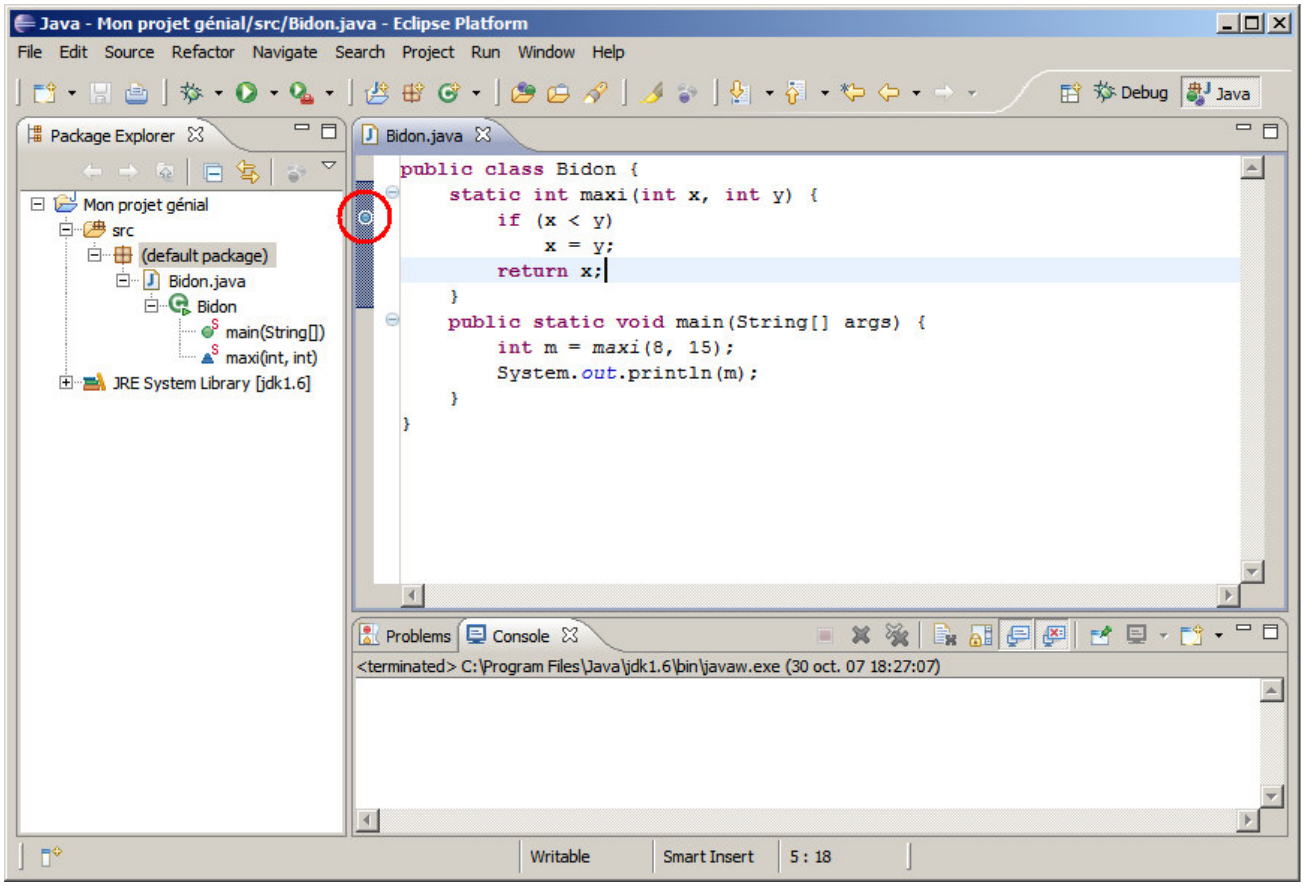

Fig. 12

Il faut ensuite lancer le débogage, à l'aide du bouton à gauche de celui qui lance l'exécution, représentant une punaise (bug) :

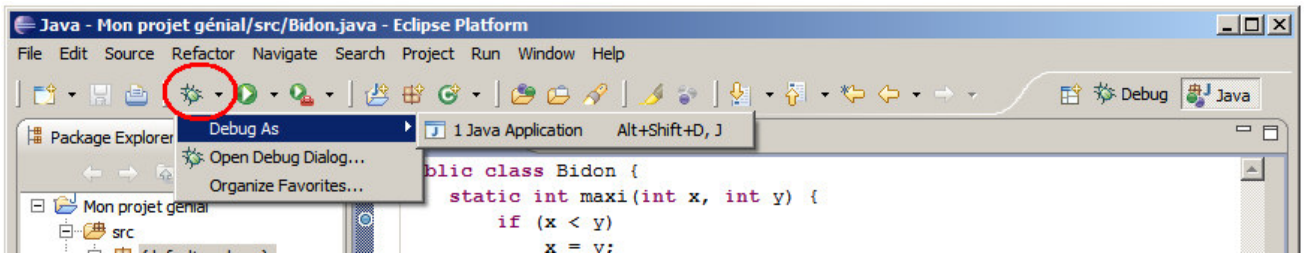

Fig. 13

L'exécution est alors lancée et se déroule normalement jusqu'à atteindre le point d'arrêt. *Eclipse* demande alors la permission de changer de *perspective* (ensemble et disposition des vues montrées) et adopte l'apparence de la figure 14 :

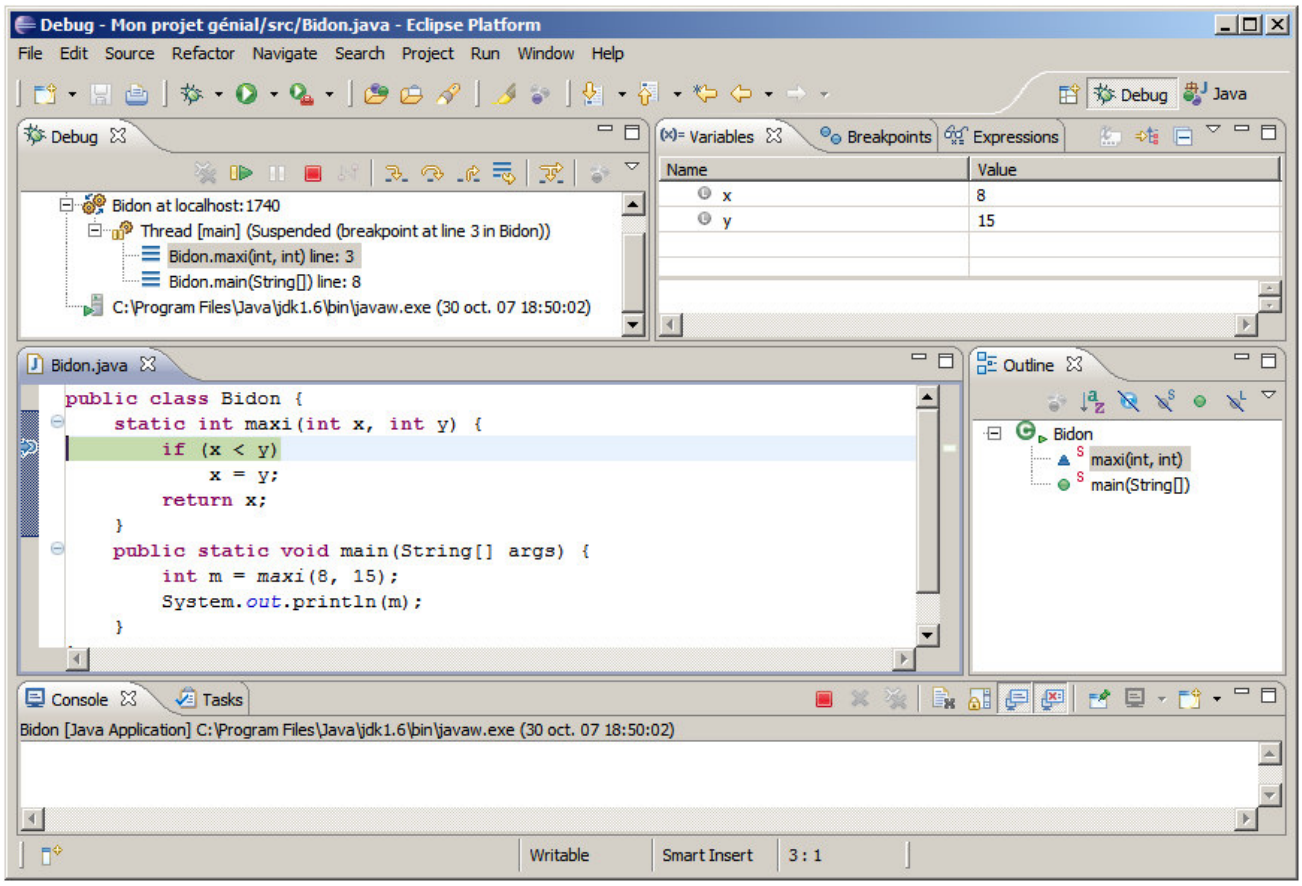

Fig. 14

La vue *Debug*, en haut à gauche de la fenêtre, montre la *pile d'exécution*, c'est-à-dire, pour chaque thread, l'empilement des méthodes qui se sont mutuellement appelées (méthodes commencées et non terminées). Dans la figure 14, par exemple, on attire notre attention sur la méthode *Bidon.main*, plus précisément la ligne 8 du fichier source, où a été appelée la méthode *Bidon.maxi*, dans laquelle l'exécution est arretée, à la ligne 3.

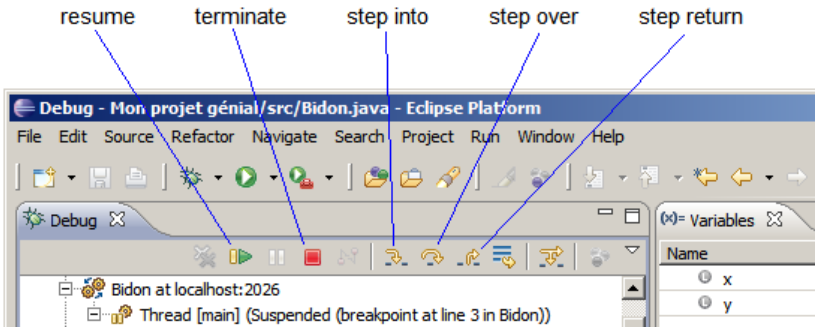

Fig. 15

En haut de cette vue (figure 15) se trouvent des boutons très utiles. Parmi les principaux :

- *step over* : faire avancer l'exécution d'une ligne. Si cette dernière contient un appel de méthode, *ne pas détailler* l'activation de celle-ci, c'est-à-dire considérer l'appel comme une instruction indivisible,
- *step into* : avancer l'exécution d'une ligne. Si un appel de méthode est concerné, détailler son activation, c'est-à-dire aller dans la méthode et s'arrêter sur sa première ligne,
- *step return* : relancer l'exécution normale, jusqu'à la fin de la méthode dans laquelle on est arrêté et le retour à la méthode qui a appelé celle-ci,
- *resume* : relancer l'exécution normale, jusqu'à la fin du programme ou le prochain point d'arrêt,
- *terminate* : terminer l'exécution.

En haut et à droite de la fenêtre principale se trouvent les vues *Variables* et *Expressions*. La première affiche les valeurs courantes des variables locales de la méthode en cours, la deuxième affiche les valeurs courantes des expressions sélectionnées avec la commande *Watch* (cliquer avec le bouton droit sur l'expression à surveiller).

Le debogueur d'*eclipse* possède bien d'autres commandes très puissantes, comme les *points d'arrêt conditionnels* et la possibilité de

modifier les valeurs des variables du programme. Prenez un peu de temps pour les explorer, c'est payant.

10. Le « refactoring »

Réusiner (*refactor*) un programme *correct* c'est modifier son texte source sans changer son fonctionnement et ses résultats. Par exemple, changer le nom d'une variable ou d'une méthode parce que, par suite de l'évolution du programme – ou du programmeur –, le nom initialement choisi est devenu moins adapté ou expressif qu'un autre.

Le *refactoring* se traduit généralement par des opérations globales, fastidieuses, qu'il n'est pas facile d'automatiser. Heureusement *eclipse* offre de puissantes fonctions pour effectuer ce travail. Voici leur description, directement traduite de l'aide en ligne du logiciel.

Commandes du menu Refactor

Nom	Description
Rename	Renomme l'élément selectionné et (si activé) corrige toutes les références à cet élément, y compris dans les autres fichiers. Pour : Méthodes, paramètres des méthodes, champs (variables d'instance et de classe), variables locales, types (classes, interfaces, enums...), paramètres des types (types génériques), constantes des enum (types énumérations), unités de compilation, paquetages, dossiers sources, projets et toute sélection de texte qui renvoie à un élément d'un de ces types. Clavier : Alt + Shift + R Options : Le renommage d'un type (classe, interface, enum...) permet de renommer aussi les variables et méthodes ayant des noms similaires. Pour cela, cocher la case "Update similarly named variables and methods" du dialogue "Rename Compilation Unit" qui apparait quand, dans la vue "Package Explorer", on fait clic-droit sur un nom de classe, puis "Refactor" > "Rename..." Sélectionner "Configure..." pour choisir la stratégie de reconnaissance des noms des types. Le renommage d'un paquetage permet celui de ses sous-paquetages. Cocher la case "Rename subpackages" dans le dialogue "Rename Package" qui apparaît quand, dans la vue "Package Explorer", on fait clic-droit sur un nom de paquetage, puis "Refactor" > "Rename..." Cocher la case "Keep original method as delegate to changed method" pour garder la méthode originale. De manière optionnelle, vous pouvez rendre désapprouver (deprecate) l'ancienne méthode.
Move	Déplace l'élément sélectionné et (si activé) corrige toutes les références à cet élément, y compris dans les autres fichiers. Pour : Méthodes d'instance (qui peuvent être déplacées dans un composant, comme une classe interne), méthodes et variables de classe, types, unités de compilation, paquetages, dossiers sources et projets et toute sélection de texte qui renvoie à un élément d'un de ces types. Clavier : Alt + Shift + V Options : Vous pouvez aussi initier cette opération en faissant un « traîner & déposer » (Drag & Drop) dans la vue "Package Explorer"
Change Method Signature	Change les noms des paramètres d'une méthode, leur type, leur ordre et met à jour toutes les références à cette méthode. De plus, des paramètres peuvent être enlevés ou ajoutés et le type du résultat et la visibilité de la méthode peuvent être modifiés. Pour : Toute méthodes et toute sélection de texte qui renvoie à une méthode. Clavier : Alt + Shift + C Options : Pour garder quand même la méthode originale (sous forme de délégation, c'est-à-dire réduite à un appel de la nouvelle méthode), cocher la case "Keep original method as delegate to changed method" dans le dialogue "Change Method Signature"
Extract Method	Crée une nouvelle méthode contenant les instructions ou expressions couramment sélectionnées et remplace la sélection par un appel de la nouvelle méthode. Cette fonctionnalité est utile pour nettoyer les méthodes trop longues, imbriquées ou excessivement complexes. Pour : Une sélection de texte qui peut constituer le corps d'une méthode. Vous pouvez utiliser "Expand Selection to" du menu "Edit" afin d'obtenir une sélection valide.

	Clavier : Alt + Shift + M
Extract Local Variable	Crée une nouvelle variable initialisée par l'expression couramment sélectionnée et remplace la sélection par une référence à cette nouvelle variable. Pour : Une sélection de texte qui constitue une expression (i.e. qui détermine une valeur). Vous pouvez utiliser "Expand Selection to" du menu "Edit" afin d'obtenir une sélection valide. Clavier : Alt + Shift + L
Extract Constant	Crée une variable statique finale initialisée avec l'expression couramment sélectionnée et remplace cette expression par une référence à la variable créée. De manière optionnelle, réecrit également les autres expressions où apparaît l'expression extraite. Pour : Toute expression constante ou sélection de texte qui renvoie à une telle expression.
Inline	Développe « en ligne » une variable locale, une méthode ou un constante : – dans le cas d'une constante, cela signifie que les occurrences de son nom sont remplacées partout par l'expression utilisée pour initialiser la variable (la déclaration de cette dernière *doit* être associée à une initialisation). – dans le cas d'une méthode, cela signifie que ses appels sont remplacés partout par le corps de la méthode, après substitution textuelle des paramètres formels par les paramètres effectifs. – dans le cas d'une constante, cela signifie que les occurrences de son nom sont remplacées partout par la valeur de la constante. Pour : Variables locales, méthodes, champs statiques finaux et toute sélection de texte qui renvoie à une méthode, un champ statique final ou une variable locale. Clavier : Alt + Shift + I
Convert Anonymous Class to Nested	Convertit une classe interne anonyme en une classe membre (c'est-à-dire une classe interne nommée). Pour : Expression définissant une classe interne anonyme

Convert Member Type to Top Level	Crée une nouvelle unité de compilation Java (c'est-à-dire un fichier source) pour le type (classe, interface, enum...) interne sélectionné, en mettant à jour toutes les références nécessaires.
	Pour les types internes qui ne sont pas statiques, un champ est ajouté si nécessaire pour permettre l'accès à l'instance qui était précédemment englobante. Pour comprendre ce charabia, sélectionnez `ClasseInterne` dans l'exemple suivant, exécutez la commande "Refactor" > "Convert Member Type to Top Level" et examinez le code produit.
	`public class ClasseEnglobante {`
	`  Object membreDeClasseEnglobante;`
	`  class ClasseInterne {`
	`  void methodeDeClasseInterne() {`
	`  membreDeClasseEnglobante = 0;`
	`  }`
	`  }`
	`}`
	Pour : Type membre (classe, interface, enum... interne) ou un texte qui renvoie à un tel type.

Convert Local Variable to Field	Convertit une variable locale en un champ (c.-à-d. une variable d'instance). Si la variable est initialisée lors de sa déclaration, alors l'expression d'initialisation est transportée soit dans la déclaration du champ soit, si nécessaire, dans les constructeurs de la classe. Pour : Toute sélection de texte qui renvoie à une variable locale.
Extract Superclass	Extrait une super-classe commune à un ensemble de classes apparentées, lesquelles deviennent alors des sous-classes directes du la nouvelle super-classe. Pour : Tous types (classes, interfaces, etc.) Options : Activez "Use the extracted class where possible" pour utiliser la classe nouvellement créée partout où c'est possible. Voyez la rubrique *Use Supertype Where Possible*.
Extract Interface	Crée une nouvelle interface avec un ensemble de méthodes, à choisir parmi les méthodes publiques de la classe, et déclare la classe en question comme une implémentation de l'interface nouvelle. Pour : Classes Options : Activez "Use the extracted class where possible" pour utiliser la classe nouvellement créée partout où c'est possible. Voyez la rubrique *Use Supertype Where Possible*.

Use Supertype Where Possible	Remplace les occurrences d'un type par une de ses super-classes après avoir identifié tous les endroits où cela est possible. Pour : Types (classes, interface, enum, etc.)
Push Down	Déplace un ensemble de méthodes et variables d'instance d'une classe vers une de ses sous-classes. Pour : Un ou plusieurs méthodes et variables déclarées dans la même classe ou dans une sélection de texte à l'intérieur d'une variable ou méthodes.
Pull Up	Déplace vers une super-classe une variable d'instance ou une méthode. Pour une méthode, on a le choix entre déplacer la méthode vers la super-classe, ou bien déclarer dans cette dernière une méthode abstraite. Pour : Une ou plusieurs méthodes, variables d'instance ou classes internes déclarés dans la même classe ou dans une sélection de texte à l'intérieur d'une variable, d'une méthode ou d'une classe interne.
Introduce Indirection	Crée une méthode de classe (statique) délègant sa tâche à la méthode d'instance sélectionnée. Par exemple, si dans une classe *UneClasse* on a sélectionné une méthode de signature *resultat uneMethode(arguments)* alors cette commande crée une méthode *static resultat uneMethode(UneClasse unObjet, arguments)* dont le corps est réduit à l'appel *unObjet.uneMethode(arguments)*. Pour : Une méthode ou une sélection de texte qui renvoie à une méthode. Options: Activez "Redirect all method invocations" pour remplacer tous les appels de la méthode originale par des appels de la méthode créée (*x.uneMethode(y)* sera remplacé par *UneClasse.uneMethode(x, y)*).
Introduce Factory	Crée une méthode « usine » (*factory method*) correspondant à un constructeur sélectionné. C'est-à-dire si on a sélectionné un constructeur de signature *UneClasse(arguments)*, ajoute une méthode telle que *static UneClasse createUneClasse(arguments) { return new UneClasse(arguments); }*. De plus, remplace les expressions de la forme *new UneClasse(arguments)* par des expressions *UneClasse.createUneClasse(arguments)*. Pour : Déclarations de constructeurs
Introduce Parameter Object	Remplace l'ensemble des paramètres d'une fonction par un unique objet, instance d'une classe nouvellement définie à cet effet, dont les variables d'instance sont les paramètres originaux. Pour : Une méthode ou une sélection de texte qui renvoie à une méthode. Options: Activez "Keep original method as delegate to changed method", dans la boîte de dialogue qu'ouvre cette commande, pour conserver quand-même la méthode originale.
Introduce Parameter	Remplace une expression par un argument formel ajouté à la méthode où elle apparaît et met à jour tous les appels de cette méthode en leur ajoutant un nouveau paramètre effectif qui n'est autre que l'expression en question. Pour : Sélections de texte qui correspondent à des expressions.
Encapsulate Field	Remplace toutes les références à un champ (variable d'instance) par des méthodes « get » et « set ». Pour : Tout champ et toute sélection de texte qui renvoie à un champ.
Generalize Declared Type	Permet de choisir un des super-types (classes, interfaces) du type de la référence couramment sélectionnée, en vérifiant qu'il n'y aura pas de problème si le type de cette dernière est changé vers le type sélectionné ; le changement est ensuite effectué. Pour : Déclarations de variables de classe, d'instance ou locales, et paramètres formels des méthodes.
Infer Generic Type Arguments	Remplace les occurrences brutes de types génériques par des types paramétrés, après avoir identifié tous les endroits où ce remplacement est possible. Pour : Projets, paquetages et types. Options: "Assume clone() returns an instance of the receiver type" (« Supposer que clone() renvoie une instance du type du destinataire ») : les classes bien écrites respectent généralement cette règle mais, si vous souhaitez que votre code la viole, ne cochez pas cette case. "Leave unconstrained type arguments raw (rather than inferring <?>)". Cochez cette case si vous préférez laisser *bruts* les types non contraints, au lieu de leur donner le paramètre *joker* <?>
Migrate JAR File	Met à jour un fichier JAR appartenant au « build path » d'un projet de votre espace de travail (*workspace*) en essayant d'utiliser l'information sur le refactoring effectué, afin d'éviter des erreurs provoquées par ce refactoring. Pour : Fichiers JAR du « build path »

Create Script	Crée un « script » avec les opérations de refactoring qui ont été appliquées dans le espace de travail (*workspace*). Ces scripts peuvent être enregistrés dans un fichier ou bien copiés dans le presse-papiers. Voyez "Apply Script". Pour : Toujours possible
Apply Script	Applique un script fait d'opérations de refactoring aux projets de votre espace de travail (*workspace*). Les scripts de refactoring peuvent être chargés depuis un fichier ou bien depuis le presse-papiers. Voyez "Create Script". Pour : Toujours possible
History	Feuillette l'historique des refactorings en permettant d'en effacer certains. Pour : Toujours possible

16.07.09