

Programmation Java TP #1

A. Pigeau / J. Cohen

L'objectif du TP est de découvrir l'environnement de programmation Java (en ligne de commande et sous l'IDE Eclipse) et de vous exercer à la programmation Java.

La première partie du sujet contient les exercices à réaliser et la deuxième partie présente l'IDE Eclipse. Cette dernière partie doit être lue après avoir fini le premier exercice. Seul le premier exercice est à compiler/exécuter en ligne de commande. Pour les autres, l'environnement de développement intégré (IDE) *Eclipse* doit être utilisé.

1 Exercice

Le premier exercice doit être compilé et exécuté avec `Javac` et `Java` en ligne de commande. Les autres seront compilés via Eclipse.

Pour la compilation manuelle :

- `javac Test.java` : le fichier `Test.class` est créé ;
- `java Test` : la fonction `main` du fichier `Test.class` est interprétée par la machine virtuelle Java.

Attention, le système d'exploitation doit connaître l'emplacement de ces commandes. Il peut être nécessaire de configurer la variable d'environnement `PATH`. Pour que `JAVA` sache où aller chercher les fichiers `.class`, il peut aussi être nécessaire de configurer la variable d'environnement `CLASSPATH`.

Configuration sous Windows :

- `PATH` : panneau de configuration → système → avancé → variable d'environnement. Dans les variable de l'utilisateur, ajouter la variable `PATH=%PATH%;c:\chemin`, où `chemin` est le chemin vers les commandes `JAVA`
- `CLASSPATH` : panneau de configuration → système → avancé → variable d'environnement. Dans les variable de l'utilisateur, ajouter la variable `CLASSPATH=%CLASSPATH%;.`, pour indiquer que les classes se situent dans le répertoire courant

Configuration sous Linux :

- `PATH` : taper la commande suivante `export PATH=$PATH:/home/user/mes_prog`, en remplaçant le chemin par l'emplacement de vos fichiers.
- `CLASSPATH` : taper la commande `export CLASSPATH=.:$CLASSPATH`. Une autre solution consiste à lancer la commande `java` en lui donnant en option le `classpath`, `java -cp . laClasseContenantLeMain`.

Pour compiler et exécuter :

- positionner vous à la racine du package de vos classes
- compilation : `javac ./tpJava/tp1/exercice1/Test.java`
- exécution : `java tpJava.tp1.exercice1.Test`

Enfin, petite présentation des conventions/contraintes à respecter en Java :

- un fichier `.java` ne peut contenir qu'une seule classe `public`
- le fichier porte le même nom que la classe qu'il contient (la classe `Voiture` est dans un fichier `Voiture.java`)

- le nom d’une classe commence toujours par une majuscule
- le nom d’une variable ou d’un attribut commence par une minuscule
- le nom d’une méthode commence par une minuscule
- tous les noms de variables/attributs/classes/méthodes composés de plusieurs mots ont une majuscule pour chacun des ces mots (`class PizzaNantaise`, `String ingredientDelicieux`, `getIngredients()`, ...)
- le nom d’une constante doit être en majuscule, avec ses différents mots séparés par `_` (`TAILLE_MAX`)

1.1 Exercice : [Classe, Objet, String & Javac/Java]

Questions :

1. demander à l’enseignant les explications sur les packages Java
2. créer un package `nomEtudiant.tp1.pile`
3. consulter la classe `String` de l’API Java
(Googler *API Java*)
4. implémenter une pile de caractères (classe `PileDeCar`) à l’aide d’un tableau de caractères primitif (100 cases maximum), en définissant l’*interface de programmation* suivante :
 - `estVide()` – retourne vrai si la pile est vide ;
 - `estPleine()` – retourne vrai si la pile est pleine ;
 - `empiler(char)` – ajoute un caractère au sommet de la pile ;
 - `depiler()` – soustrait le caractère situé en haut de la pile, et le retourne.Ajouter à cette interface, un certain nombre d’*accesseurs* (méthodes `getX()`) à :
 - sommet de la pile, pour tester la valeur sans la dépiler ;
 - capacité maximale de la pile, stockée en tant que *variable de classe*, i.e. partagée par toutes les instances (mot-clé : `static`) ;
 - nombre d’éléments dans la pile.
5. consulter la documentation sur les `StringBuilder`. Un `StringBuilder` est un objet contenant un tableau de caractères pouvant être modifié, contrairement aux objets de type `String` qui sont eux non modifiables
6. Écrire une méthode `miroir` qui accepte une phrase (chaîne de caractères) en paramètre, et empile chaque caractère de la chaîne correspondante pour ensuite les dépiler et retourner la phrase originale inversée.
Exemple : `"Hello World!"` est transformé en `"!dlroW olleH"`
Attention : `s = s + pile.depiler()` est similaire à l’instruction
`s = new String(s + pile.depiler())`. Cela est donc très coûteux. Il faut passer par un `StringBuilder` pour récupérer vos "dépilements" successifs.

Consignes :

- exploiter au maximum l’interface de programmation de la classe `PileDeCar` ;

1.2 Exercice : [Classe, constructeur & ArrayList]

Soit le cahier des charges suivants :

- une personne est définie par un nom, un prénom, un âge et deux personnes comme parent
- l’instanciation d’une personne nécessite plusieurs constructeurs :
 - un constructeur prenant tous les paramètres
 - un constructeur ne prenant que le nom, le prénom et l’âge
 - un constructeur par défaut, avec comme nom, prénom et âge `John Doe – 25`
- une classe `GestionPersonne` gère un ensemble de personne
- la classe `GestionPersonne` permet de :

- calculer l'âge moyen des personnes
- ajout de personnes dans la liste
- affichage des personnes
- trouver et retourner une personne en la cherchant par son nom ou bien par son nom et prénom. **null** est retourné si la personne n'existe pas

À partir de cet exercice, les implémentations sont à réaliser sous Eclipse.

Questions :

1. lire la documentation sur Eclipse
2. créer un package `nomEtudiant.tp1.personne`
3. créer votre classe `Personne` et ajouter les attributs
4. implémenter le constructeur ayant le plus de paramètre
5. implémenter les deux autres constructeurs en utilisant la fonction `this(...)` qui permet, dans un constructeur, d'appeler un autre constructeur. L'appel à `this()` doit se faire obligatoirement sur la première ligne dans un constructeur
6. implémenter les accesseurs de la classe `Personne`
7. consulter la documentation sur la classe `Object`. En java, toute classe hérite par défaut de cette classe, et donc de toutes ces méthodes. Regarder en détail la méthode `toString()`
8. implémenter la méthode `public String toString()` dans votre classe `Personne`, en retournant une chaîne de caractères "nom/prénom/age"
9. implémenter un `Test` dans lequel plusieurs personnes seront créées. Chaque personne sera affichée avec l'instruction `System.out.println(personne1)`. Quelle méthode est appelée automatiquement pour transformer une personne en chaîne de caractères ?
10. consulter la documentation sur le conteneur `ArrayList`. Un `ArrayList` est un tableau dynamique, dans lequel nous allons ajouter les `Personne`
11. créer votre classe `GestionPersonne` et ajouter les attributs
12. implémenter les méthodes de `GestionPersonne`
13. implémenter un test dans une classe `Test`

1.3 Exercice : [Association & énumération]

Soit le cahier des charges suivants :

- une voiture est définie par un nom de marque, un nom de modèle et un prix
- les marques de voiture sont `Polygeot`, `Polynault` et `Polyniat`
- les modèles de voiture sont `A`, `AA` et `AAA`
- une voiture est composée de 4 roues et d'un moteur
- une roue est définie par une marque et un prix
- les marques de roues sont `PolyLin` et `PolyYear` et `PolyPneu`
- les modèles de roues sont `Super`, `Super+` et `Super++`
- un moteur est défini par une puissance et un prix
- la puissance d'un moteur est définie par les valeurs suivantes `80Ch`, `100ch` et `120ch`
- le prix, la marque et le modèle de chaque partie d'une voiture sera fourni à la construction des instances
- chaque marque a trois gammes de voiture : `Privilège`, `Classic` et `Standard`

Questions :

1. créer un package `nomEtudiant.tp1.voiture`
2. consulter la documentation sur les énumérations en Java

3. implémenter la classe `Moteur`
4. implémenter la classe `Roue`
5. implémenter la classe `Voiture`
6. implémenter un test dans une classe `Test`, ou plusieurs voitures seront instanciées. On définit par exemple la voiture `Standard` Polygeot par une voiture `Polygeot - A - 8000 euros` avec 4 roues `Polylin - Super - 400 euros` et un moteur `80ch - 2000 euros`. Une Polynault `Classic` est défini par une voiture `Polynault - AA - 10000 euros` avec 4 roues `Polylyear - Super+ - 400 euros` et un moteur `80ch - 2000 euros` (nous vous laissons définir les autres gammes de voiture pour chaque marque)
7. implémenter les méthodes `toString` de chaque classe. Faire en sorte, quand on affiche une voiture dans le `Test`, d'afficher toutes les informations sur les 4 roues et le moteur
8. implémenter la méthode `getPrix()` de `Voiture` retournant le prix total de la voiture (avec roues et moteur)
9. instancier plusieurs fois les voitures de base de votre `Test`. Vous avez fait un copier/coller ? combien de lignes de code à modifier pour le client si la gamme standard d'une marque est modifiée ?
10. implémenter une classe `FactoryVoiture` ayant seulement des méthodes `static` retournant une instance d'une voiture particulière en fonction de sa gamme
11. modifier votre `Test` en utilisant maintenant vos méthodes de votre classe `FactoryVoiture` pour créer plusieurs instances de vos voitures. combien de lignes de code à modifier pour le client si la gamme standard d'une marque est modifiée ?

1.4 Exercice Bonus : [Garbage collector]

Afin de tester le garbage collector de la machine virtuelle, nous allons implémenter un test créant beaucoup d'objets et libérant les pointeurs les référençant.

Questions :

1. créer un package `nomEtudiant.tp1.garbage`
2. implémenter une classe `ObjectToDelete` contenant le numéro d'instance de l'objet (sous forme d'un entier - le premier objet instancié aura un numéro d'instance égale à 1). Ce numéro sera initialisé à partir d'une variable `static` comptabilisant le nombre d'instances de `ObjectToDelete` créées ;
3. redéfinir la méthode `finalize()` de `ObjectToDelete`. Cette méthode fera simplement un affichage sur `stdout`. Pour information :
 - tout objet en Java hérite implicitement de la classe `Object`. Chaque objet hérite donc de quelques méthodes comme `finalize()`, `toString()`, ... ;
 - la méthode `finalize()` est appelée par le garbage collector juste avant de supprimer l'objet.
4. si vous n'avez utilisé aucun raccourci pour générer la méthode `finalize()`, re-lire la section *Introduction à Eclipse* de ce document (section 2) ;
5. fournir un test créant beaucoup d'instances de `ObjectToDelete` et libérant les pointeurs les référençant. Quand le garbage collector est activé, il supprimera les instances non référencées ;
6. quand le garbage collector est-il appelé ? est-il possible de l'appeler explicitement ? est ce une bonne solution de l'appeler explicitement ?
7. relancer vos tests en utilisant l'option `-Xms5m` comme paramètre de la commande `java`, ce qui limitera la mémoire disponible à 5Mo. Pouvez-vous tirer de nouvelles informations de cette expérience ?

1.5 Exercice Bonus : [Number]

Soit le cahier des charges suivants d'une classe `GestionMesure` :

- la classe `GestionMesure` gère une liste de `Number`
- chaque élément de la liste est un `Double`
- la classe `GestionMesure` permet de calculer la moyenne, l'écart type, la médiane
- l'interface de la classe doit permettre d'ajouter des `Integer`, `Float` et `Double` dans la liste
- l'interface de la classe doit permettre d'ajouter des nombre à partir de liste de `Number`
- l'interface doit permettre de créer une instance avec par défaut une liste de nombre vide ou une instance à partir d'une liste de `Number`

Questions :

1. créer un package `nomEtudiant.tp1.number`
2. consulter la classe `Number`
3. consulter l'interface `List`
4. créer votre classe `GestionMesure` et implémenter les attributs
5. implémenter les constructeurs
6. implémenter les méthodes d'ajout de nombre dans la liste
7. implémenter les méthodes de moyenne, écart type et médiane
8. implémenter un test où chaque méthode sera testée

1.6 Exercice Bonus : [Arbre & Static]

Nous désirons modéliser un arbre binaire défini où chaque noeud contient un entier. Les opérations suivantes devront être disponibles :

- afficher les valeurs des noeuds en préfixé
- trouver la valeur maximale
- déterminer la profondeur
- retourner le nombre de noeud créé

Questions :

1. créer un package `nomEtudiant.tp1.arbre`
2. implémenter votre classe `Element`
3. implémenter un test (construire un arbre de profondeur 2)
4. implémenter la fonction `toString`
5. implémenter la fonction `getMax`
6. implémenter la fonction `getProfondeur`
7. vérifier que les méthodes n'utilisant pas la référence `this` sont bien `static` (suivant la solution proposée)
8. implémenter un test des fonctions précédentes

2 Introduction à Eclipse

Eclipse IDE¹ est un environnement de développement intégré libre extensible, universel et polyvalent, permettant de créer des projets de développement mettant en oeuvre les langages les plus répandus. Eclipse IDE est principalement écrit en Java (à l'aide de la bibliothèque graphique SWT, d'IBM), et ce langage, grâce à des bibliothèques spécifiques, est également utilisé pour écrire des extensions.

Eclipse IDE est développé autour de la notion de plugin. Ainsi l'éditeur peut être étendu facilement avec de nouvelles fonctionnalités. De nombreux projets ont été développés et permettent de programmer dans différents langages : Java, C++, Php, ...

Dans ce tutorial², nous vous présentons tout d'abord comment installer Eclipse et créer votre première application Java. Ensuite nous présentons l'utilisations du débogueur.

3 Installation & Démarrage :

Pour installer Eclipse, il suffit de télécharger un programme exécutable. Eclipse n'a pas besoin de s'installer sur le système d'exploitation :

- connectez vous sur le site www.eclipse.org, section downloads.
- sélectionnez la version *Eclipse pour les développeurs Java*
- décompressez le fichier

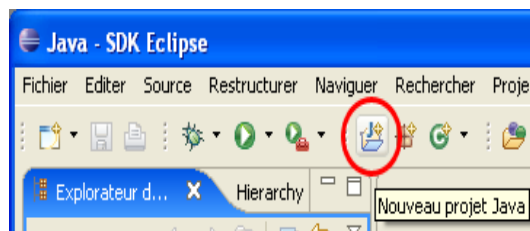
Pour fonctionner Eclipse nécessite un JDK, à télécharger sur le site de Sun (<http://java.sun.com/javase/downloads/index.jsp>).

Pour lancer l'éditeur :

- Exécuter le fichier eclipse.exe
- Après un premier écran d'attente une boîte de dialogue s'ouvre demandant l'emplacement du répertoire de travail à utiliser (le 'workspace'). Ce répertoire contiendra tous vos paramètres de configuration et généralement vos projets.

4 Création d'un projet Java :

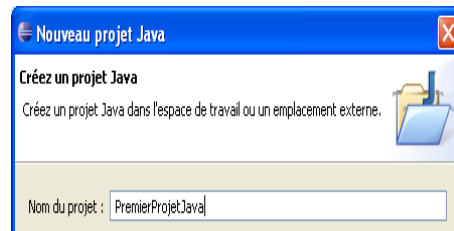
La première étape pour développer une application Java avec Eclipse est de créer un 'Projet Java'. L'ouverture de l'assistant se fait soit en utilisant le menu Fichier>nouveau projet, soit en utilisant le bouton correspondant dans la barre de boutons :



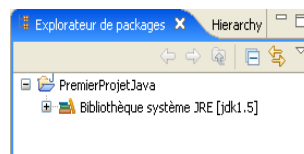
1. Wikipedia

2. Le tutorial est tiré du site <http://www.eclipse totale.com/articles/premierPas.html>

La seule information nécessaire pour l'assistant est le nom du projet. Une fois le nom saisi, le bouton 'Terminer' est activé, cliquer sur ce bouton pour demander la création du projet (L'assistant est composé d'une deuxième page qui permet de configurer de façon précise le chemin de compilation du projet. Les paramètres offerts par cette deuxième page peuvent être fait ultérieurement dans les propriétés du projet).

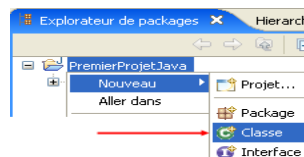


Après la création, le projet apparaît dans la vue 'Explorateur de projets' :

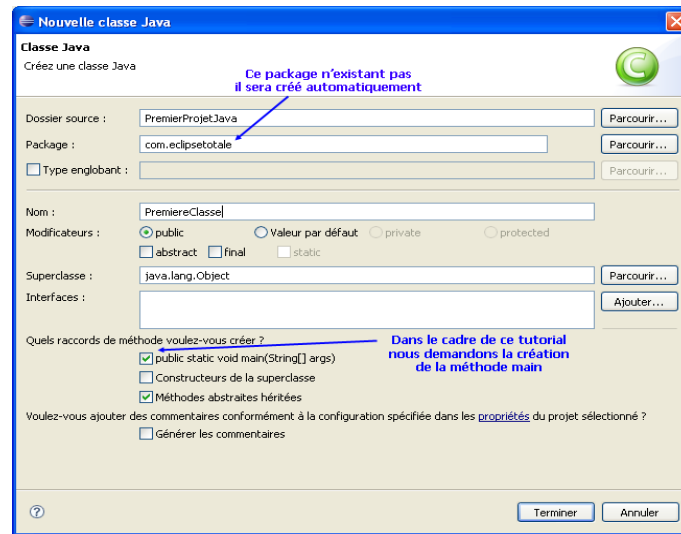


5 Création d'une classe :

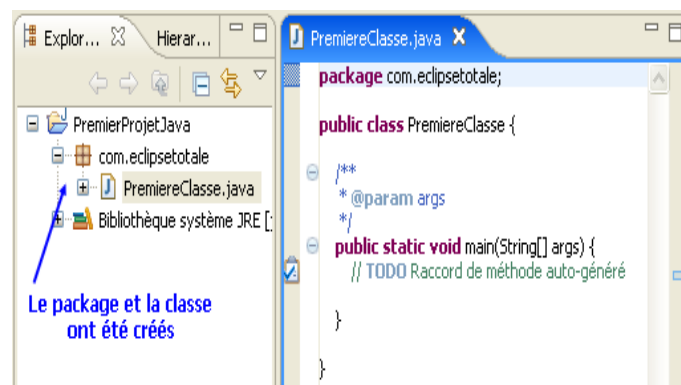
Pour ouvrir l'assistant de création de classes Java : faire afficher le menu contextuel du projet Java précédemment créé et sélectionner l'option 'New->Class'.



Les informations importantes attendues par cet assistant sont le nom de la classe et le nom de son package. Entrer un nom de package (le package sera créé automatiquement), saisir le nom de la classe et cocher la case indiquant à l'assistant de générer une méthode main.

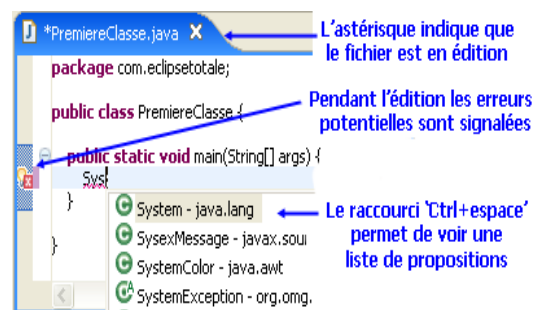


Après avoir cliqué sur le bouton 'Terminer', la classe est créée et ouverte en édition :



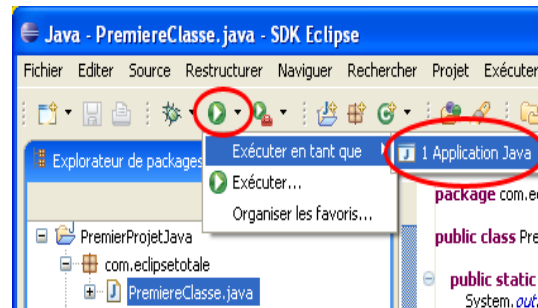
Dans l'éditeur, compléter la méthode 'main'. Une fois le code saisi, demander l'enregistrement (Ctrl+S ou menu Fichier), la classe est enregistrée et compilée.

Pendant la saisie :



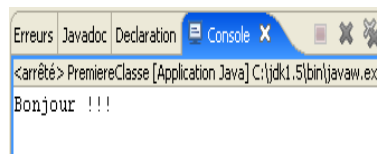
6 Exécution :

Pour demander l'exécution de la classe : déplier le menu associé au bouton et sélectionner l'option 'Exécuter en tant que...' -> 'Application Java' :



Une autre méthode pour exécuter votre programme est de passer par le menu 'Run Configurations' (toujours sous la flèche verte). En cliquant dessus, Eclipse vous ouvre une fenêtre regroupant les différentes configurations d'exécution de vos programmes. Double-cliquez sur 'Java Application' pour configurer le lancement d'une application Java : dans la partie droite, vous pouvez sélectionner le `main` à exécuter, les arguments pris en paramètre par le `main`,...

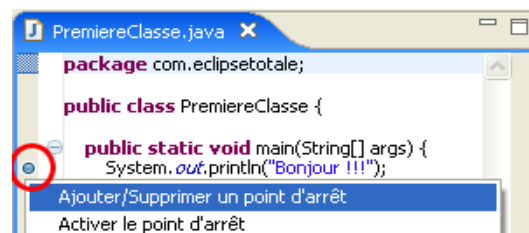
La vue 'Console' affiche le résultat :



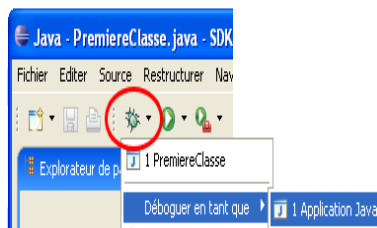
7 Le débogage :

Le débogage permet d'exécuter votre programme pas à pas tout en consultant les valeurs de vos variables. Des points d'arrêts doivent être placés dans votre programme pour déterminer à partir de quelles instructions vous voulez vous arrêter.

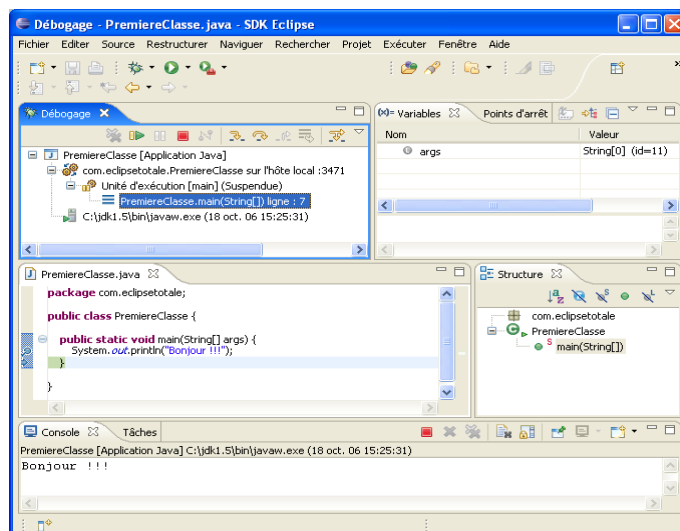
Placer un point d'arrêt en double-cliquant dans la marge ou en utilisant le menu contextuel dans la marge :





Lancer l'application en mode 'Débogage' :



Lorsque que le point d'arrêt est rencontré, Eclipse propose d'afficher le débogueur :



Les boutons  permettent d'avancer en pas à pas, le premier en rentrant dans le code de la méthode appelée, le second en passant à l'instruction suivante. Utiliser le bouton  pour reprendre l'exécution normale.

Testez le débogueur avec le programme *Test*, présenté dans le listing 1.

8 Trucs et astuces :

8.1 Fonctionnalités importantes de l'éditeur :

La richesse de l'éditeur Java est une des bases du succès d'Eclipse. Dans la première partie de ce tutorial deux caractéristiques de l'éditeur ont été signalées :

- la complétion : le raccourci *Ctrl+espace* affiche une liste de propositions.
- la marquage des erreurs : le code est analysé au fur et à mesure de la saisie, tout comme dans un traitement de texte le code non valide est souligné en rouge.

L'éditeur propose de nombreuses autres fonctionnalités, en voici quelques unes :

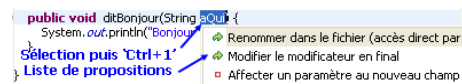
Listing 1 – Exemple de code Java à déboguer

```
import java.io.*;
1
2
public class Test {
3
4     public static int sommeDesCarres(int n){
5         int i , somme=0;
6         for( i =0; i <=n ; i ++){
7             somme+= i * i ;
8         }
9         return somme;
10    }
11    public static void main(String [] argv){
12        int m=0, s ;
13        String lec;
14
15        System.out.println("Entrez un nombre positif : ");
16        BufferedReader br = new BufferedReader( new InputStreamReader( System.in ) );
17        try {
18            lec = br.readLine();
19            m = Integer.parseInt(lec);
20        } catch (IOException e) {
21            e.printStackTrace();
22        }
23        s = Test.sommeDesCarres(m) ;
24        System.out.println("La somme des "+m+" premiers entiers est "+s+"\n");
25    }
26 }
```

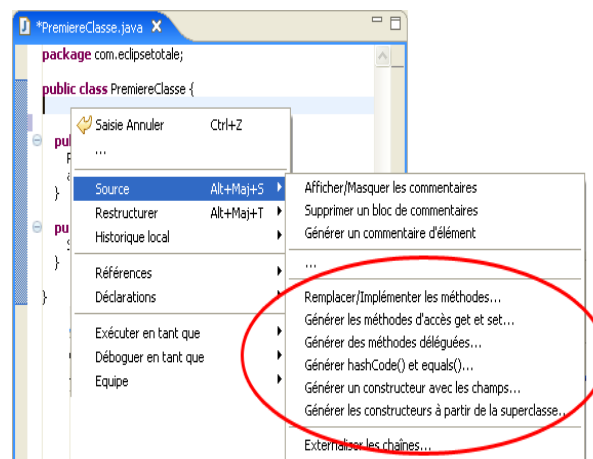
- Complétion et import : lorsque la complétion est utilisée pour un nom de classe, Eclipse ajoute automatiquement la déclaration de l'import si ce dernier est absent.
- Complétion et notion de 'modèles' : en plus des noms de types, de méthodes ou de variables la liste des choix proposés peut correspondre à des modèles. Ces modèles sont des blocs de code prédéfinis, leur liste est consultable dans la section **Window->Preferences->Java->Editor->Templates** (Il est possible de créer ces propres modèles).



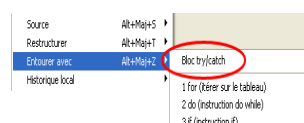
- Correction rapide : le raccourci **Ctrl+1** affiche le 'QuickFix'. Ce 'correcteur rapide' propose une liste d'actions possibles pour résoudre une erreur (généralement une ampoule apparaît aussi dans la marge). A noter que le correcteur peut être utilisé même s'il n'y a pas d'erreur, il propose alors diverses actions comme le renommage d'une variable.



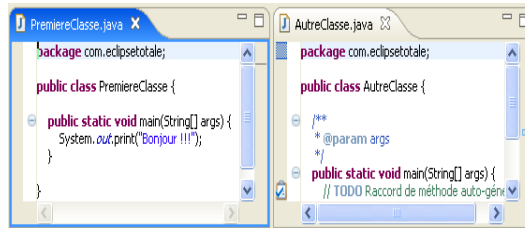
- Gestion des imports : le raccourci **Ctrl+Shift+O** (aussi accessible par le menu contextuel 'Source->Organiser les importations') gère la liste des imports : les imports manquants sont ajoutés, les inutiles supprimés et la liste est classée. Ces actions se font conformément aux paramètres définis dans **Préférences->Java->Style de code->Organiser les importations**.
- Formatage : le raccourci **Ctrl+Shift+F** déclenche le formatage soit de tout le fichier soit des lignes sélectionnées. Les options de formatage sont configurables dans **Préférences->Java->Style de Code->Formater**.
- Génération de méthodes : dans l'éditeur, la section 'Source' du menu contextuel permet de générer plusieurs types de méthode, par exemple les accesseurs (méthodes get et set) ou encore des constructeurs.



- Génération de blocs try/catch : dans le menu contextuel de l'éditeur, la section 'Entourer avec' permet d'encadrer le bloc de code sélectionné par différents types d'instructions (try/catch, boucle, condition...). Dans le cas du 'try/catch', un bloc catch est généré pour chaque exception possible.



- Scinder la zone d'édition : par défaut, Eclipse ouvre un nouvel onglet dans la zone d'édition pour chaque fichier. Il est d'une part possible de limiter le nombre d'onglets à partir de 'Préférences->Général->Editeurs->Nombre d'éditeurs ouverts avant fermeture', d'autre part la zone d'édition peut être scindée : sélectionner un onglet et le déplacer pour scinder la zone d'édition.



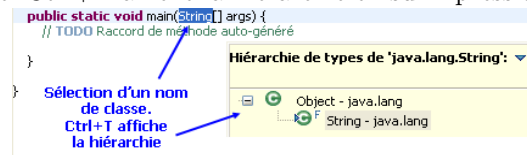
8.2 Navigation

L'éditeur propose plusieurs raccourcis utiles pour naviguer de classe en classe et pour visualiser des informations sur un nom de classe ou de méthode sélectionné :

- Ouvrir le code de l'élément sélectionné : le raccourci F3 affiche le code de l'élément actuellement sélectionné dans l'éditeur (en ouvrant un autre éditeur si nécessaire). Il est aussi possible d'utiliser la touche Ctrl seule : si la touche Ctrl est enfoncée l'éditeur affiche un lien hypertexte quand un nom de classe ou de méthode est survolé.



- Afficher la hiérarchie d'une classe : le raccourci F4 affiche la classe dont le nom est sélectionné dans la vue 'Hiérarchie'. Le raccourci Ctrl+T affiche la hiérarchie en surimpression :

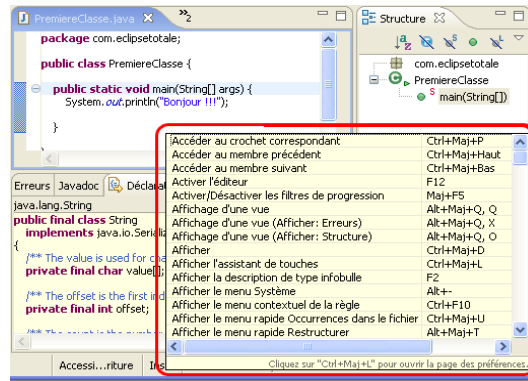


8.3 Plus de trucs et astuces

L'environnement Eclipse propose de nombreuses fonctionnalités, ce tutorial n'a pour ambition que de vous donner les bases pour débiter avec l'outillage Java d'Eclipse. Pour aller plus loin et notamment découvrir plus de trucs et astuces voici deux premières pistes :

- consultez la section 'Tips and tricks' de l'aide d'Eclipse accessible par le menu 'Help->Tips and tricks'
- prenez le temps de parcourir l'ensemble des pages de préférences d'Eclipse pour voir les paramétrages disponibles.

Les trucs et astuces les plus intéressants sont généralement accessibles rapidement via un raccourci clavier. Si vous ne souhaitez en retenir qu'un seul, Ctrl+Shift+L peut faire l'affaire car ce dernier affiche en surimpression la liste de tous les raccourcis clavier.



8.4 Les raccourcis importants

Certains raccourcis doivent être connus par coeur pour gagner du temps. En pratique, ils permettent :

- de générer du code automatiquement
- de se déplacer dans votre code rapidement
- de connaître les liens d'associations entre vos classes
- de nettoyer votre code

Les raccourcis à connaître sont :

- Ctrl+Espace pour la complétion (par exemple "sysous" + Ctrl+Espace)
- Ctrl+Maj+'o' (appliqué dans un fichier source) : import automatique des packages dans le fichier (et suppression des "import" inutiles)
- 'F3' (appliqué sur une méthode, un attribut ou une classe) : déplacement du prompt sur la définition de l'attribut, la méthode ou la classe
- Ctrl+Alt+'h' (appliqué sur une méthode, un attribut ou une classe) : affichage d'une liste de méthode utilisant l'attribut, la méthode ou la classe
- Ctrl+'a' suivie de Ctrl+'i' (appliqué dans un fichier source) : Ctrl+'a' permet de sélectionner tout le code du fichier et Ctrl+'i' permet d'indenter le code correctement