Bases de la programmation en Java

L'un des principes phares de Java réside dans sa machine virtuelle : celle-ci assure à tous les développeurs Java qu'un programme sera utilisable avec tous les systèmes d'exploitation sur lesquels est installée une machine virtuelle Java. Lors de la phase de compilation de notre code source, celui-ci prend une forme intermédiaire appelée **ByteCode** : c'est le fameux code inintelligible pour votre machine, mais interprétable par la machine virtuelle Java. Cette dernière porte un nom : on parle plus communément de **JVM (Java Virtual Machine).**

Le bytecode rend le programme indépendant de la plateforme. Il donne la possibilité de démarrer le programme sur un autre processeur sans recompilation, pour autant que la JVM soit installée. Par exemple, un programme compilé sur une machine Linux aura strictement le même comportement qu'il soit exécuté sur Windows, Linux, Mac Os, etc. Plus besoin de se soucier des spécificités liées à tel ou tel OS (Operating System, soit système d'exploitation). Nous pourrons donc nous consacrer entièrement à notre programme.

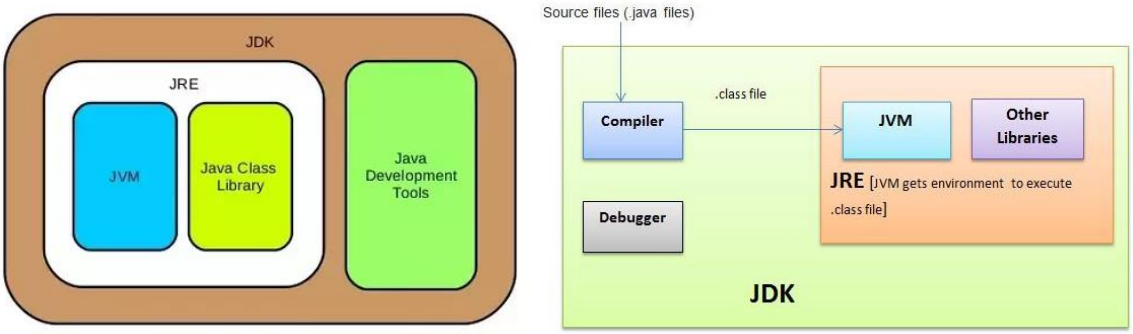
Les instructions après compilation sont placées dans un fichier spécifique portant l'extension ***.class***. Pour tout fichier de code Java, il existera, après compilation, un fichier ***.class*** dans lequel, on retrouvera le **bytecode** donnée pour la classe Java compilé.

JDK, JRE, JVM : késako ?

JDK (Java Development Kit) : C'est un kit de développement logiciel utilisé pour créer des applications à l'aide du langage de programmation Java. Il contient le JRE, un jeu de classes API, un compilateur Java et d'autres fichiers requis pour l'écriture d'applications Java.

JRE (Java Runtime Environment) : C'est un environnement nécessaire à l'exécution d'applications créés à l'aide du langage de programmation Java. Il contient la **JVM (Java Virtual Machine)**, les bibliothèques de base et d'autres composants supplémentaires pour l'exécution d'applications et d'applets écrits dans Java.

JVM (Java Virtual Machine) : C'est une machine virtuelle Java - elle exécute en fait un bytecode Java.



Afin de nous simplifier la vie, nous allons utiliser un outil de développement, ou IDE (Integrated Development Environment), pour nous aider à écrire nos futurs codes source… Nous allons donc avoir besoin de différentes choses afin de pouvoir créer des programmes Java.

Types d'applications Java

Java permet de développer différents types d'applications : il y a donc des environnements permettant de créer des programmes pour différentes plates-formes.

- J2SE (Java 2 Standard Edition) : permet de développer des applications dites « client lourd », par exemple Word, Excel, la suite OpenOffice.org… Toutes ces applications sont des « clients lourds ». C'est ce que nous allons faire dans ce cours.
- J2EE (Java 2 Enterprise Edition) : permet de développer des applications web en Java. On parle aussi de clients légers.
- J2ME (Java 2 Micro Edition) : permet de développer des applications pour appareils portables, comme des téléphones portables, des PDA…

Eclipse IDE

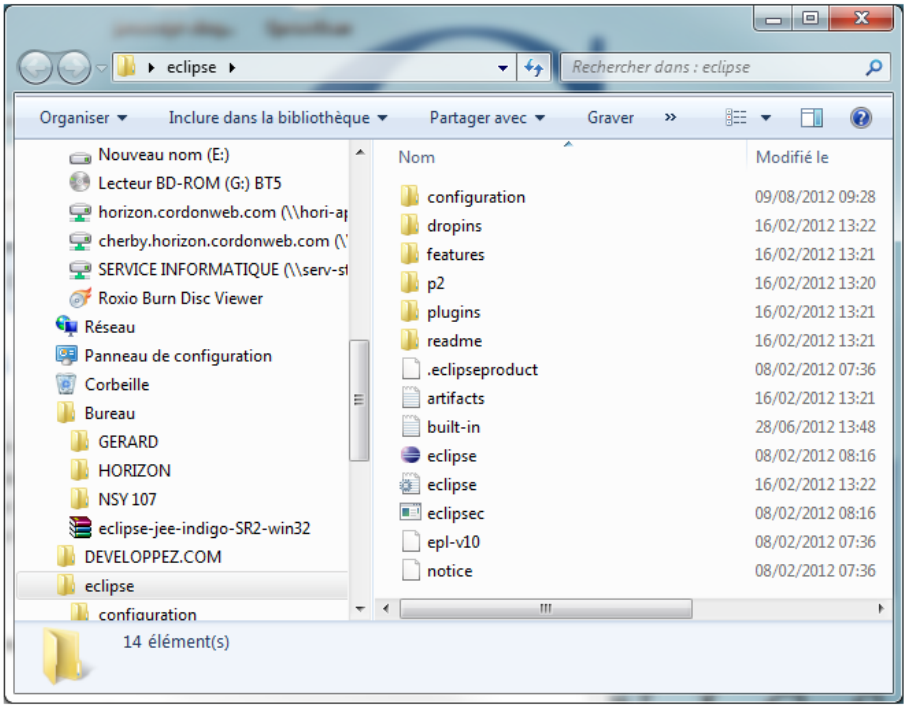
Avant toute chose, quelques mots sur le projet Eclipse. Eclipse IDE est un environnement de développement libre permettant de créer des programmes dans de nombreux langages de programmation (Java, C++, PHP…). C'est en somme l'outil que nous allons utiliser pour programmer.

Eclipse est le petit logiciel qui va nous permettre de développer nos applications ou nos applets (Un applet est un logiciel qui s'exécute dans la fenêtre d'un navigateur web), et aussi celui qui va les compiler. Notre logiciel va donc permettre de traduire nos futurs programmes Java en langage **byte code**, compréhensible uniquement par notre JRE, fraîchement installé. La spécificité d'Eclipse IDE vient du fait que son architecture est totalement développée autour de la notion de **plug-in**. Cela signifie que toutes ses fonctionnalités sont développées en tant que plug-

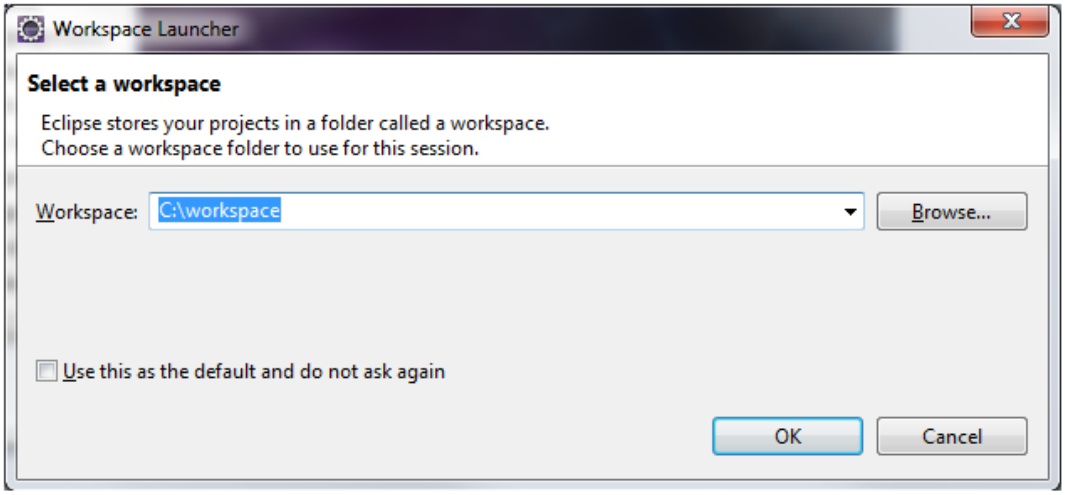
ins. Pour faire court, si vous voulez ajouter des fonctionnalités à Eclipse, vous devez :

- télécharger le plug-in correspondant ;
- copier les fichiers spécifiés dans les répertoires spécifiés ;
- démarrer Eclipse, et ça y est !

Lorsque vous téléchargez un nouveau plug-in pour Eclipse, celui-ci se présente souvent comme un dossier contenant généralement deux sous-dossiers : un dossier « plugins » et un dossier « features ». Ces dossiers existent aussi dans le répertoire d'Eclipse. Il vous faut donc copier le contenu des dossiers de votre plug-in vers le dossier correspondant dans Eclipse (plugins dans plugins, et features dans features). Vous devez maintenant avoir une archive contenant Eclipse. Décompressez-la où vous voulez, puis entrez dans ce dossier (figure suivante).



Cliquez sur l'icône d'Eclipse pour le lancer… Ici (figure suivante), Eclipse vous demande dans quel dossier vous souhaitez enregistrer vos projets ; sachez que rien ne vous empêche de spécifier un autre dossier que celui proposé par défaut.



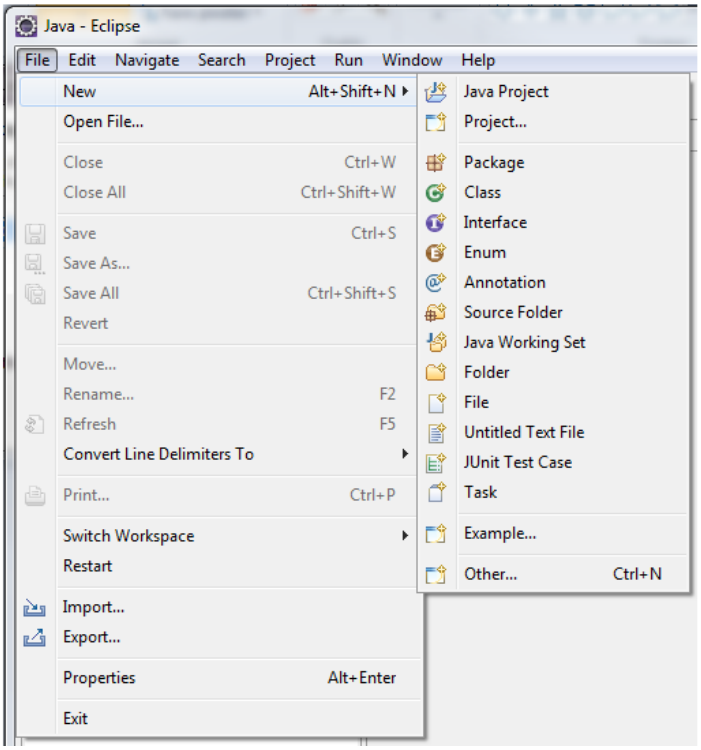
Première fenêtre d'Eclipse

Une fois cette étape effectuée, vous arrivez sur la page d'accueil d'Eclipse.

Présentation de l'interface d'Eclipse

Faisons un tour rapide de l'interface d'Eclipse.

Le menu « File »

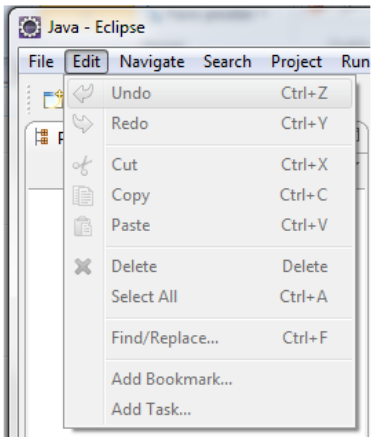


Menu File

C'est ici que nous pourrons créer de nouveaux projets Java, les enregistrer et les exporter le cas échéant. Les raccourcis à retenir sont :

- ALT + SHIFT + N : nouveau projet ;
- CTRL + S : enregistrer le fichier où l'on est positionné ;
- CTRL + SHIFT + S : tout sauvegarder ;
- CTRL + W : fermer le fichier où l'on est positionné ;
- CTRL + SHIFT + W : fermer tous les fichiers ouverts.

Le menu « Edit »

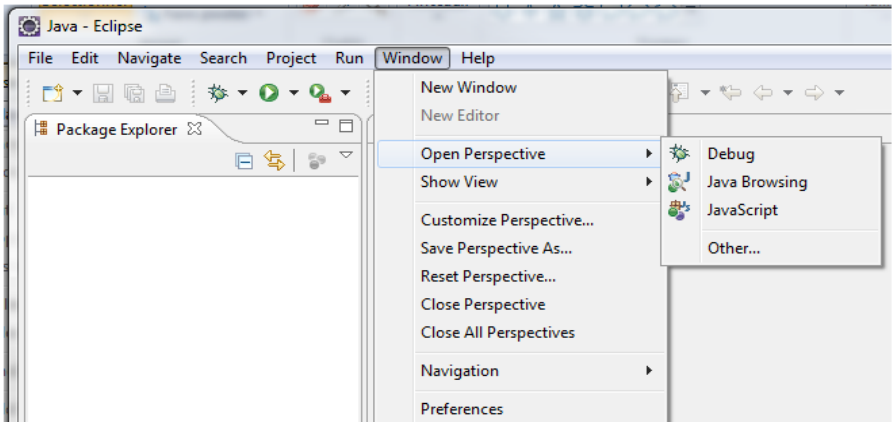


Menu Edit

Dans ce menu, nous pourrons utiliser les commandes « copier », « coller », etc. Ici, les raccourcis à retenir sont :

- CTRL + C : copier la sélection ;
- CTRL + X : couper la sélection ;
- CTRL + V : coller la sélection ;
- CTRL + A : tout sélectionner ;
- CTRL + F : chercher-remplacer.

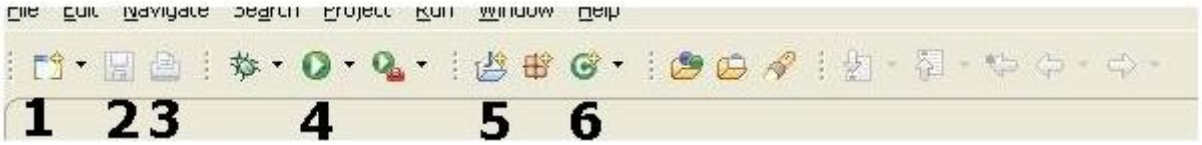
Le menu « window »



Menu window

Dans celui-ci, nous pourrons configurer Eclipse selon nos besoins.

La barre d'outils

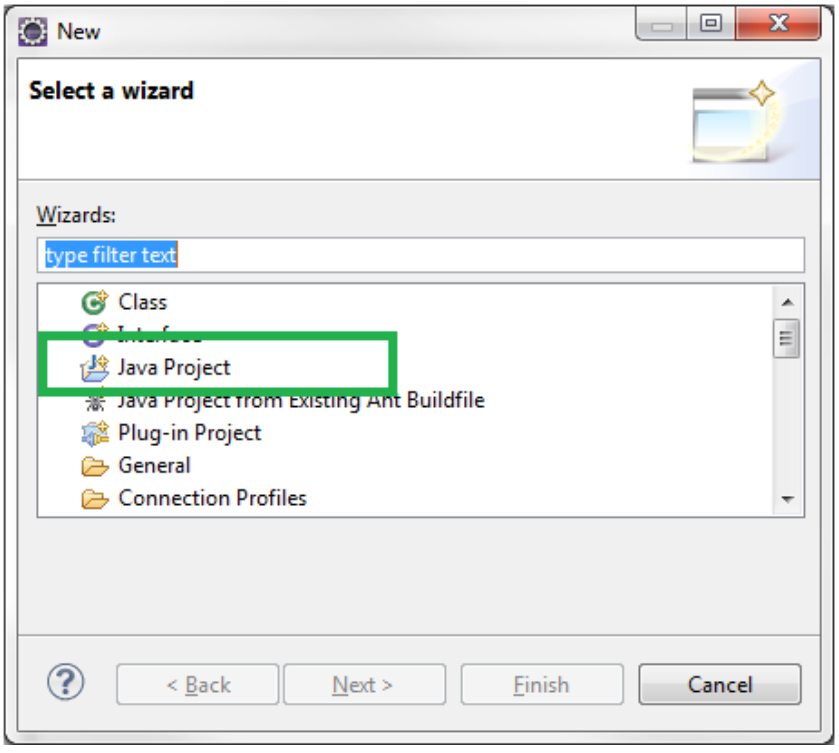


Barre d'outils

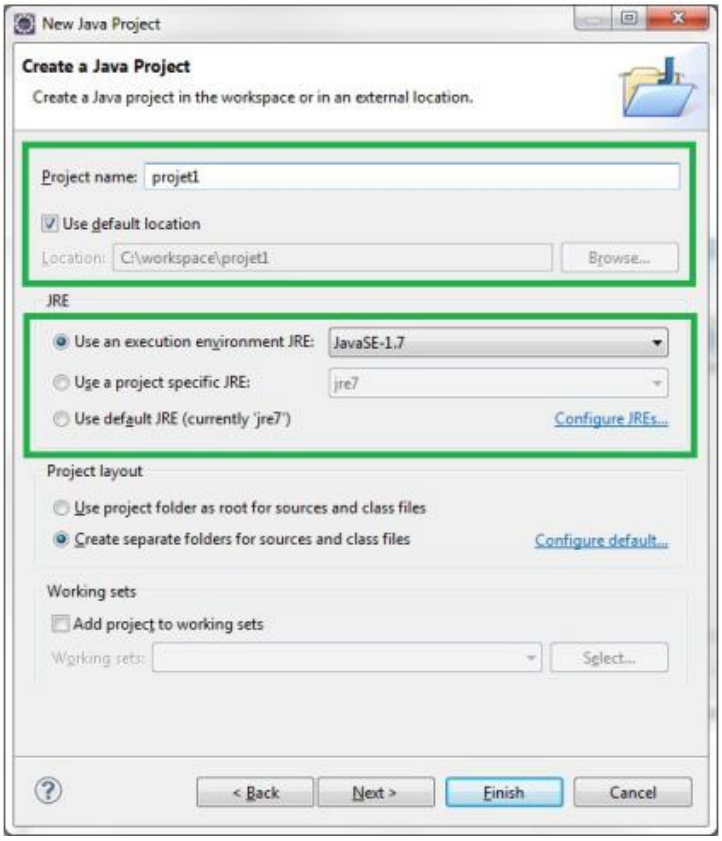
Nous avons dans l'ordre :

1. nouveau général. Cliquer sur ce bouton revient à faire « Fichier / Nouveau » ;
2. enregistrer. Revient à faire CTRL + S ;
3. imprimer ;
4. exécuter la classe ou le projet spécifié. Nous verrons ceci plus en détail ;
5. créer un nouveau projet. Revient à faire « Fichier/Nouveau/Java Project » ;
6. créer une nouvelle classe, c'est-à-dire en fait un nouveau fichier. Revient à faire « Fichier / Nouveau / Classe ».

Maintenant, nous allons créer un nouveau projet Java.



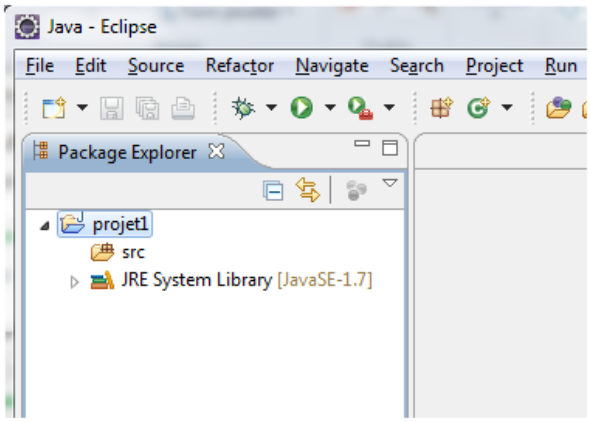
Création de projet JAVA 1



Création de projet JAVA 2

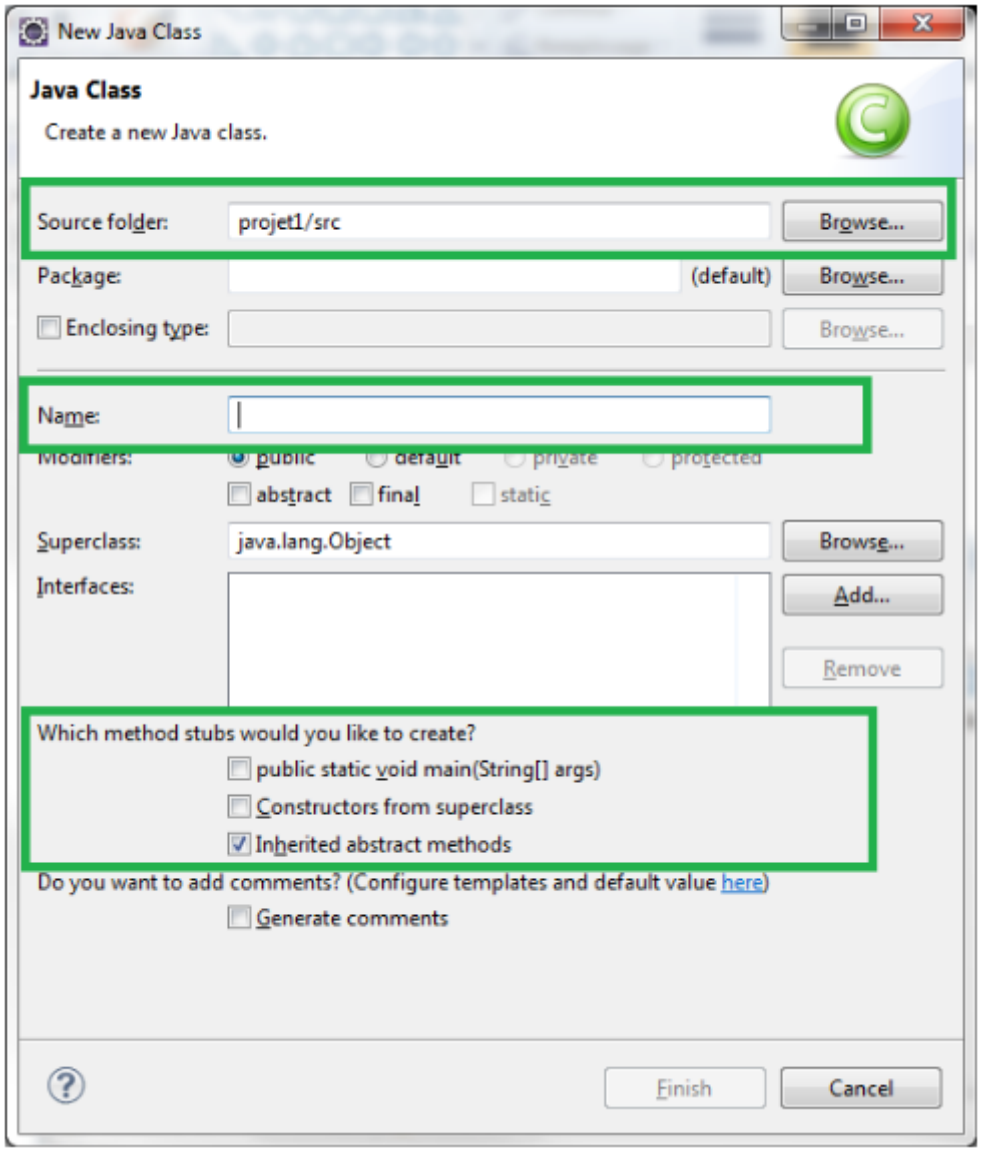
Renseignez le nom de votre projet comme cela a été fait (encadré 1). Vous pouvez aussi voir où sera enregistré ce projet. Un peu plus compliqué, maintenant : vous avez donc un environnement Java sur votre machine, mais dans le cas où vous en auriez plusieurs, vous pouvez aussi spécifier à Eclipse quel JRE utiliser pour ce projet (encadré 2). Vous pourrez changer ceci à tout moment dans Eclipse en allant dans « Window / Preferences », en dépliant l'arbre « Java » dans la fenêtre et en choisissant «Installed JRE».

Vous devriez avoir un nouveau projet dans la fenêtre de gauche (figure suivante).



Explorateur de Fichier

Pour boucler la boucle, ajoutons dès maintenant une nouvelle classe (Une classe est un ensemble de codes contenant plusieurs instructions que doit effectuer le programme) dans ce projet comme nous avons appris à le faire plus tôt. Voici la fenêtre sur laquelle vous devriez tomber :

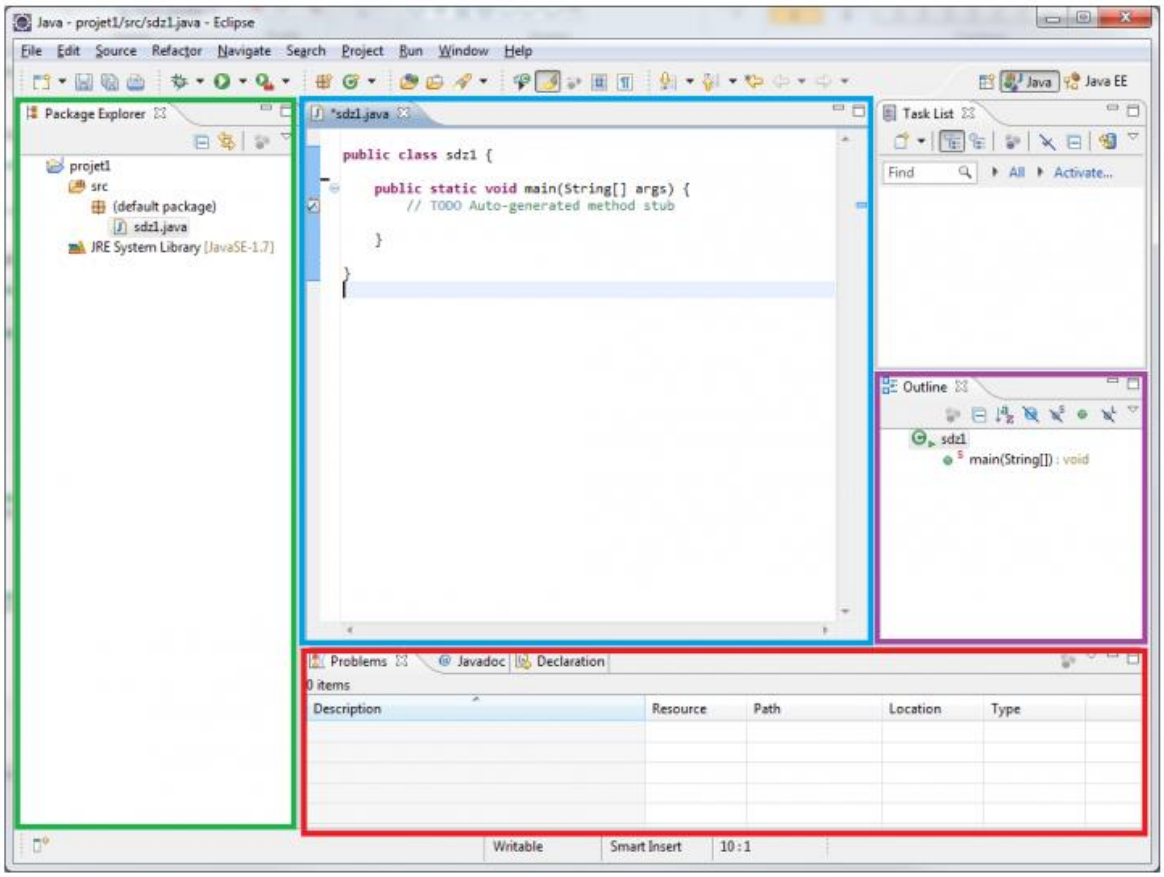


Création d'une classe

Dans l'encadré 1, nous pouvons voir où seront enregistrés nos fichiers Java.

Dans l'encadré 2, nommez votre classe Java ;

Dans l'encadré 3, Eclipse vous demande si cette classe a quelque chose de particulier. Cochez « public static void main(String[] args) » (nous reviendrons plus tard sur ce point) , puis cliquez sur « Finish » .



Fenêtre principale d'Eclipse

Avant de commencer à coder, nous allons explorer l'espace de travail.

Dans l'encadré de gauche, vous trouverez le dossier de votre projet ainsi que son contenu. Ici, vous pourrez gérer votre projet comme bon vous semble (ajout, suppression…).

Dans l'encadré positionné au centre, nous allons écrire nos codes source.

Dans l'encadré du bas, c'est là que vous verrez apparaître le contenu de vos programmes… ainsi que les erreurs éventuelles.

Et pour finir, c'est dans l'encadré de droite, dès que nous aurons appris à coder nos propres fonctions et nos objets, que la liste des méthodes et des variables sera affichée.

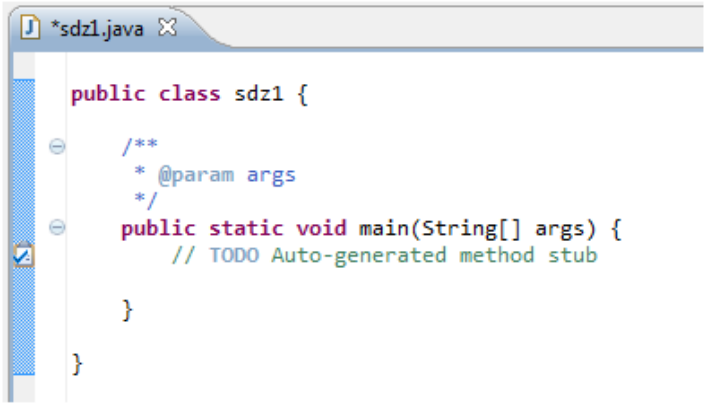
Premier programme JAVA

Il faut savoir que **tous les programmes Java sont composés d'au moins une classe**. Cette classe doit contenir une méthode appelée main : ce sera le point de démarrage du programme. Une méthode est une suite d'instructions à exécuter. C'est un morceau de logique de notre programme. Une méthode contient :

- **un en-tête :** celui-ci va être en quelque sorte la carte d'identité de la méthode ;
- **un corps :** le contenu de la méthode, délimité par des accolades ;
- **une valeur de retour :** le résultat que la méthode va retourner.

Remarque : Vous verrez un peu plus tard qu'un programme n'est qu'une multitude de classes qui s'utilisent l'une l'autre. Mais pour le moment, nous n'allons travailler qu'avec une seule classe.

Vous aviez créé un projet Java ; ouvrez-le :



Méthode principale

Vous voyez la fameuse classe dont nous parlons? Ici, elle s'appelle « sdz1 ». Vous pouvez voir que le mot **class** est précédé du mot public, dont nous verrons la signification lorsque nous programmerons des objets.

Pour le moment, ce que nous devons retenir, c'est que votre classe est définie par un mot clé (**class**), qu'elle a un nom (ici, sdz1) et que son contenu est délimité par des accolades ({}).

Nous écrirons nos codes sources entre la méthode main. La syntaxe de cette méthode est toujours la même :

```
public static void main(String[] args){
//Contenu de votre classe
}
```

Pour rendre le code écrit compréhensible, on insert des phrases d'explications qui ne sont pas pris en compte lors de la compilation grâce à une méthode un peu particulière. Il s'agit des commentaires. Il en existe de deux types :

- **les commentaires unilignes** : introduits par les symboles //, ils mettent tout ce qui les suit en commentaire, du moment que le texte se trouve sur la même ligne que les //.

```
public static void main(String[] args){
//Un commentaire
//Un autre
//Encore un autre
Ceci n'est pas un commentaire !
}
```

- **les commentaires multilignes** : ils sont introduits par les symboles **/*** et se terminent par les symboles ***/**.

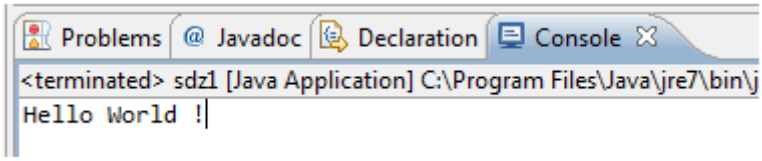
```
public static void main(String[] args){

/*
Un commentaire
Un autre
Encore un autre
*/
Ceci n'est pas un commentaire !
}
```

Essayons de taper le code suivant :

```
public static void main(String[] args){
    System.out.print("Hello World !");
}
```

Remarque : N'oubliez surtout pas le " ; " à la fin de la ligne ! **Toutes les instructions en Java sont suivies d'un point-virgule.**

Une fois que vous avez saisi cette ligne de code dans votre méthode main, il vous faut lancer le programme. Si vous regardez dans votre console, dans la fenêtre du bas sous Eclipse, vous devriez voir la figure suivante.



Expliquons un peu cette ligne de code. Littéralement, elle signifie « la méthode **print()** va écrire **Hello World !** en utilisant l'objet **out** de la classe **System** ».

- **System** : ceci correspond à l'appel d'une classe qui se nomme « System ». C'est une classe utilitaire qui permet surtout d'utiliser l'entrée et la sortie standard, c'est-à-dire la saisie clavier et l'affichage à l'écran.
- **out** : objet de la classe System qui gère la sortie standard.
- **print** : méthode qui écrit dans la console le texte passé en paramètre.

Si vous mettez plusieurs **System.out.print**, voici ce qui se passe.

Prenons ce code:

```
System.out.print("Hello World !");
System.out.print("My name is");
System.out.print("Cysboy");
```

On a après compilation:

```
Hello World !My name isCysboy
```

Pour insérer un retour à la ligne, il existe plusieurs solutions :

- soit vous utilisez un caractère d'échappement, ici **\n** ;
- soit vous utilisez la méthode **println()** à la place de la méthode **print().**

Donc, si nous reprenons notre code précédent et que nous appliquons cela, voici ce que ça donnerait :

Code:

```
System.out.print("Hello World ! \n");
System.out.println("My name is");
System.out.println("\nCysboy");
```

Résultat:

```
Hello World !
My name is

Cysboy
```

Profitons au passage pour mentionner deux autres caractères d'échappement :

- **\r** va insérer un retour chariot, parfois utilisé aussi pour les retours à la ligne ;
- **\t** va faire une tabulation.

Remarque : Vous avez sûrement remarqué que la chaîne de caractères que l'on affiche est entourée de "<chaîne>". En Java, les guillemets doubles sont des délimiteurs de chaînes de caractères ! Si vous voulez afficher un guillemet double dans la sortie standard, vous devrez « l'échapper » (terme désignant le fait de désactiver : ici, désactiver la fonction du caractère « " ») avec un \, ce qui donnerait : System.out.println("Coucou mon \"chou\" ! ");.

Il n'est pas rare de croiser le terme anglais quote pour désigner les guillemets droits. Cela fait en quelque sorte partie du jargon du programmeur.

En résumé

- La JVM est le cœur de Java.
- Elle fait fonctionner vos programmes Java, précompilés en byte code.
- Les fichiers contenant le code source de vos programmes Java ont l'extension .java.
- Les fichiers précompilés correspondant à vos codes source Java ont l'extension .class.
- Le byte code est un code intermédiaire entre celui de votre programme et celui que votre machine peut comprendre.
- Un programme Java, codé sous Windows, peut être précompilé sous Mac et enfin exécuté sous Linux.
- Votre machine NE PEUT PAS comprendre le byte code, elle a besoin de la JVM.
- Tous les programmes Java sont composés d'au moins une classe.
- Le point de départ de tout programme Java est la méthode public static void main(String[] args).
- On peut afficher des messages dans la console grâce à ces instructions :
- System.out.println, qui affiche un message avec un saut de ligne à la fin ;
- System.out.print, qui affiche un message sans saut de ligne.

Les variables et les opérateurs

Les différents types de variables

Nous allons commencer par découvrir comment créer des variables dans la mémoire. Pour cela, il faut les déclarer. Une déclaration de variable se fait comme ceci :

```
<Type de la variable> <Nom de la variable> ;
```

Cette opération se termine toujours par un point-virgule « ; » (Comme toutes les instructions de ce langage). Ensuite, on l'initialise en entrant une valeur.

En Java, nous avons deux types de variables :

- des variables de type simple ou « primitif » ;
- des variables de type complexe ou des « objets ».

Ce qu'on appelle des **types simples** ou **types primitifs**, en Java, ce sont tout bonnement des nombres entiers, des nombres réels, des booléens ou encore des caractères, et vous allez voir qu'il y a plusieurs façons de déclarer certains de ces types.

Les variables du type numérique

Le type ***byte*** (1 octet) peut contenir les entiers entre -128 et +127.

```
byte temperature;
temperature = 64;
```

Le type ***short*** (2 octets) contient les entiers compris entre -32768 et +32767.

```
short vitesseMax;
vitesseMax = 32000;
```

Le type ***int*** (4 octets) va de $-2*10^9$ à $2*10^9$.

```
int temperatureSoleil;
temperatureSoleil = 15600000;
```

Le type **long** (8 octets) peut aller de $-9*10^{18}$ à $9*10^{18}$

```
long anneeLumiere;
anneeLumiere = 9460700000000000L;
```

Afin d'informer la JVM que le type utilisé est long, vous DEVEZ ajouter un "L" à la

fin de votre nombre, sinon le compilateur essaiera d'allouer ce dernier dans une taille d'espace mémoire de type entier et votre code ne compilera pas si votre nombre est trop grand...

Le type ***float*** (4 octets) est utilisé pour les nombres avec une virgule flottante.

```
float pi;
pi = 3.141592653f;
float nombre;
nombre = 2.0f;
```

Remarque : Vous remarquerez que nous ne mettons pas une virgule, mais un **point** ! Et vous remarquerez aussi que même si le nombre en question est rond, on écrit .0 derrière celui-ci, le tout suivi de « ***f*** ».

Le type ***double*** (8 octets) est identique à float, si ce n'est qu'il contient plus de chiffres derrière la virgule.

```
double division;
division = 0.33333333333333333333333333333333333333333334d;
```

Les variables stockant des caractères

Le **type** char contient UN caractère stocké entre apostrophes (« ' ' »), comme ceci :

```
char caractere;
caractere = 'A';
```

Les variables du type Booléen

Le type boolean, lui, ne peut contenir que deux valeurs : true (vrai) ou false (faux), sans guillemets (ces valeurs sont natives dans le langage. Il les comprend directement et sait les interpréter).

```
boolean question;
question = true;
```

Le type string

Le type ***String*** permet de gérer les chaînes de caractères, c'est-à-dire le stockage de texte. Il s'agit d'une variable d'un type plus complexe que l'on appelle ***objet***. Vous verrez que celle-ci s'utilise un peu différemment des variables précédentes :

```
String phrase;
phrase = "Titi et Grosminet";
//Deuxième méthode de déclaration de type String
String str = new String();
str = "Une autre chaîne de caractères";
//La troisième
String string = "Une autre chaîne";
//Et une quatrième pour la route
String chaine = new String("Et une de plus ! ");
```

Remarques : String commence par une majuscule ! Et lors de l'initialisation, on utilise ici des guillemets doubles (« " " »).

Convention de nommage : En effet, String commence par une lettre majuscule contrairement à tout ce qui a été présenté jusqu'ici. C'est le cas parce que String est un objet et non un type de variable. Les conventions de nommage ne se limitent pas à cela :

- tous vos noms de classes doivent commencer par une majuscule ;
- tous vos noms de variables doivent commencer par une minuscule ;
- si le nom d'une variable est composé de plusieurs mots, le premier commence par une minuscule, le ou les autres par une majuscule, et ce, sans séparation ;
- tout ceci sans accentuation !

Exemple :

```
public class Toto{}
public class Nombre{}
public class TotoEtTiti{}
String chaine;
String chaineDeCaracteres;
int nombre;
int nombrePlusGrand;
//…
```

Pour aller plus vite, on peut déclarer les variables des manières suivantes :

```
int entier = 32;
float pi = 3.1416f;
char carac = 'z';
String mot = new String("Coucou");
 int nbre1 = 2, nbre2 = 3, nbre3 = 0;
```

Les opérateurs arithmétiques

Ce sont ceux que l'on apprend à l'école primaire…

- « + » : permet d'additionner deux variables numériques (mais aussi de concaténer des chaînes de caractères).
- « - » : permet de soustraire deux variables numériques.
- « * » : permet de multiplier deux variables numériques.
- « / » : permet de diviser deux variables numériques (mais je crois que vous aviez deviné).
- « % » : permet de renvoyer le reste de la division entière de deux variables de type numérique ; cet opérateur s'appelle le **modulo**.

Exemple de calcul :

```
int nbre1, nbre2, nbre3;  //déclaration des variables
nbre1 = 1 + 3;          //nbre1 vaut 4
nbre2 = 2 * 6;          //nbre2 vaut 12
nbre3 = nbre2 / nbre1; //nbre3 vaut 3
nbre1 = 5 % 2;          //nbre1 vaut 1, car 5 = 2 * 2 + 1
nbre2 = 99 % 8;         //nbre2 vaut 3, car 99 = 8 * 12 + 3
nbre3 = 6 % 3;          //là, nbre3 vaut 0, car il n'y a pas de reste
```

Remarque : On ne peut faire du traitement arithmétique que sur des variables de même type sous peine de perdre de la précision lors du calcul. On ne s'amuse pas à diviser un int par un float, ou pire, par un char ! Ceci est valable pour tous les opérateurs arithmétiques et pour tous les types de variables numériques. Essayez de garder une certaine rigueur pour vos calculs arithmétiques.

Pour afficher les variables à l'écran en le précédent d'une chaîne de caractères, on utilise l'opérateur « + » appelé opérateur de concaténation. Il permet de mélanger du texte brut et des variables. Voici un exemple d'affichage avec une perte de précision :

```
double nbre1 = 10, nbre2 = 3;
int resultat = (int)(nbre1 / nbre2);
System.out.println("Le résultat est = " + resultat);
```

Les conversions, ou « cast »

Comme expliqué plus haut, les variables de type double contiennent plus d'informations que les variables de type int. Nous allons apprendre à convertir un type de variable en un autre.

D'un type int en type float :

```
int i = 123;
float j = (float)i; //j vaut 123.0
```

D'un type int en double :

```
int i = 123;
double j = (double)i; //j vaut 123.0
```

Et inversement :

```
double i = 1.23;
double j = 2.9999999;
int k = (int)i; //k vaut 1
k = (int)j;     //k vaut 2
```

Ce type de conversion s'appelle une conversion d'ajustement, ou cast de variable. Vous l'avez vu : nous pouvons passer directement d'un type int à un type double.

L'inverse, cependant, ne se déroulera pas sans une perte de précision. En effet, comme vous avez pu le constater, lorsque nous castons un double en int, la valeur de ce double est tronquée, ce qui signifie que l'int en question ne prendra que la valeur entière du double, quelle que soit celle des décimales. Pour en revenir à notre problème de tout à l'heure, il est aussi possible de caster le résultat d'une opération mathématique en la mettant entre « () » et en la précédant du type de cast souhaité.

```
double nbre1 = 10, nbre2 = 3;
int resultat = (int)(nbre1 / nbre2);
System.out.println("Le résultat est = " + resultat);
```

Voilà qui fonctionne parfaitement. Pour bien faire, vous devriez mettre le résultat de l'opération en type double. Et si on fait l'inverse : si nous déclarons deux entiers et que nous mettons le résultat dans un double ? Voici une possibilité :

```
int nbre1 = 3, nbre2 = 2;
double resultat = nbre1 / nbre2;
System.out.println("Le résultat est = " + resultat);
```

Vous aurez **1** comme résultat. Je ne caste pas ici, car un double peut contenir un int. En voici une autre :

```
int nbre1 = 3, nbre2 = 2;
double resultat = (double)(nbre1 / nbre2);
System.out.println("Le résultat est = " + resultat);
```

Nous obtenons le même résultat. Afin de comprendre pourquoi, nous devons savoir qu'en Java, comme dans d'autres langages d'ailleurs, il y a la notion de **priorité d'opération** ; et là, nous en avons un très bon exemple !

NB : Sachez que l'affectation, le calcul, le cast, le test, l'incrémentation… toutes ces choses sont des opérations! Et Java les fait dans un certain ordre, il y a des priorités.

Dans le cas qui nous intéresse, il y a trois opérations :

- un calcul ;
- un cast sur le résultat de l'opération ;
- une affectation dans la variable resultat.

Eh bien, Java exécute notre ligne dans cet ordre ! Il fait le calcul (ici 3/2), il caste le résultat en double, puis il l'affecte dans notre variable resultat. On se demande sûrement pourquoi vous n'avez pas 1.5…

C'est simple : lors de la première opération de Java, la JVM voit un cast à effectuer, mais sur un résultat de calcul. La JVM fait ce calcul (division de deux int qui, ici, nous donne 1), puis le cast (toujours 1), et affecte la valeur à la variable (encore et toujours 1). Donc, pour avoir un résultat correct, il faudrait caster chaque nombre avant de faire l'opération, comme ceci :

```
int nbre1 = 3, nbre2 = 2;
double resultat = (double)(nbre1) / (double)(nbre2);
System.out.println("Le résultat est = " + resultat);
//affiche : Le résultat est = 1.5
```

Nous n'allons pas trop détailler ce qui suit (vous verrez cela plus en détail dans la partie sur la programmation orientée objet) ; mais vous allez maintenant apprendre à transformer l'argument d'un type donné, int par exemple, en String.

```
int i = 12;
String j = new String();
j = j.valueOf(i);
```

j est donc une variable de type String contenant la chaîne de caractères 12. Sachez que ceci fonctionne aussi avec les autres types numériques. Voyons maintenant comment faire marche arrière en partant de ce que nous venons de faire.

```
int i = 12;
String j = new String();
j = j.valueOf(i);
int k = Integer.valueOf(j).intValue();
```

Maintenant, la variable k de type int contient le nombre **12**.

NB : Il existe des équivalents à intValue() pour les autres types numériques : floatValue(), doubleValue()…

Lire les entrées clavier

Après la lecture de ce chapitre, vous pourrez saisir des informations et les stocker dans des variables afin de pouvoir les utiliser a posteriori. En fait, jusqu'à ce que nous voyions les interfaces graphiques, nous travaillerons en mode console. Donc, afin de rendre nos programmes plus ludiques, il est de bon ton de pouvoir interagir avec ceux-ci. Par contre, ceci peut engendrer des erreurs (on parlera d'exceptions, mais ce sera traité plus loin). Afin de ne pas surcharger le chapitre, nous survolerons ce point sans voir les différents cas d'erreurs que cela peut engendrer

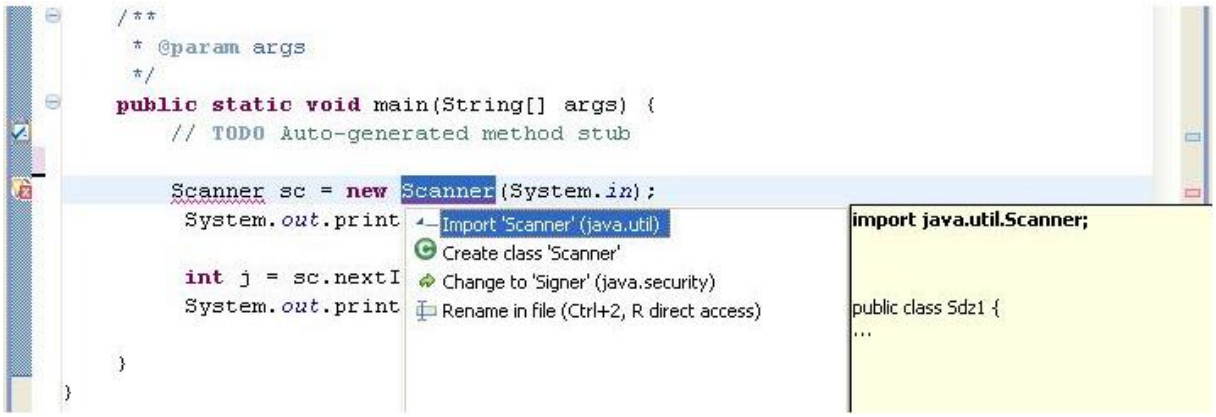
La classe Scanner

Je me doute qu'il vous tardait de pouvoir communiquer avec votre application... Le moment est enfin venu ! Mais je vous préviens, la méthode que je vais vous donner présente des failles. Je vous fais confiance pour ne pas rentrer n'importe quoi n'importe quand...

Je vous ai dit que vos variables de type String sont en réalité des objets de type String. Pour que Java puisse lire ce que vous tapez au clavier, vous allez devoir utiliser un objet de type Scanner. Cet objet peut prendre différents paramètres, mais ici nous n'en utiliserons qu'un : celui qui correspond à l'entrée standard en Java. Lorsque vous faites System.out.println(); , je vous rappelle que vous appliquez la méthode println() sur la sortie standard ; ici, nous allons utiliser l'entrée standard System.in. Donc, avant d'indiquer à Java qu'il faut lire ce que nous allons taper au clavier, nous devrons **instancier** un objet Scanner. Avant de vous expliquer ceci, créez une nouvelle classe et tapez cette ligne de code dans votre méthode main :

```
Scanner sc = new Scanner(System.in);
```

Vous devez avoir une jolie vague rouge sous le mot Scanner. Cliquez sur la croix rouge sur la gauche et faites un double-clic sur « Import 'Scanner' java.util » (figure suivante). Et là, l'erreur disparaît !



Importer la classe Scanner

Maintenant, regardez au-dessus de la déclaration de votre classe, vous devriez voir cette ligne :

```
import java.util.Scanner;
```

Voilà ce que nous avons fait. Je vous ai dit qu'il fallait indiquer à Java où se trouve la classe Scanner.

Pour faire ceci, nous devons importer la classe Scanner grâce à l'instruction **import**.

La classe que nous voulons se trouve dans le package java.util. Un package est un ensemble de classes. En fait, c'est un ensemble de dossiers et de sous-dossiers contenant une ou plusieurs classes, mais nous verrons ceci plus en détail lorsque nous ferons nos propres packages. Les classes qui se trouvent dans les packages autres que java.lang (package automatiquement importé par Java, on y trouve entre autres la classe System) sont à importer à la main dans vos classes Java pour pouvoir vous en servir.

La façon dont nous avons importé la classe java.util.Scanner dans Eclipse est très commode. Vous pouvez aussi le faire manuellement :

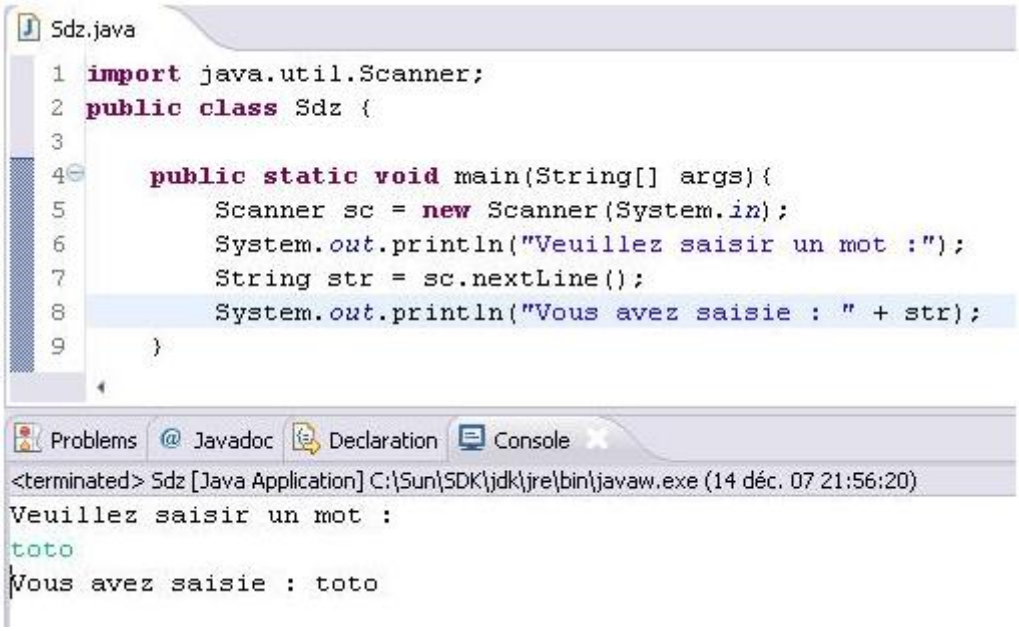
```
//Ceci importe la classe Scanner du package java.util
import java.util.Scanner;
//Ceci importe toutes les classes du package java.util
import java.util.*;
```

Récupérer ce que vous tapez

Voici l'instruction pour permettre à Java de récupérer ce que vous avez saisi pour ensuite l'afficher :

```
Scanner sc = new Scanner(System.in);
System.out.println("Veuillez saisir un mot :");
String str = sc.nextLine();
System.out.println("Vous avez saisi : " + str);
```

Une fois l'application lancée, le message que vous avez écrit auparavant s'affiche dans la console, en bas d'Eclipse. Pensez à cliquer dans la console afin que ce que vous saisissez y soit écrit et que Java puisse récupérer ce que vous avez inscrit (figure suivante) !



Saisie utilisateur dans la console

Si vous remplacez la ligne de code qui récupère une chaîne de caractères comme suit :

```
Scanner sc = new Scanner(System.in);
System.out.println("Veuillez saisir un nombre :");
int str = sc.nextInt();
System.out.println("Vous avez saisi le nombre : " + str);
```

Vous devriez constater que lorsque vous introduisez votre variable de type Scanner et que vous introduisez le point permettant d'appeler des méthodes de l'objet, Eclipse vous propose une liste de méthodes associées à cet objet (ceci s'appelle l'autocomplétion) ; de plus, lorsque vous commencez à taper le début de la méthode nextInt(), le choix se restreint jusqu'à ne laisser que cette seule méthode. Exécutez et

testez ce programme : vous verrez qu'il fonctionne à la perfection. Sauf... si vous saisissez autre chose qu'un nombre entier ! Vous savez maintenant que pour lire un int, vous devez utiliser nextInt(). De façon générale, dites-vous que pour récupérer un type de variable, il vous suffit d'appeler

next<Type de variable commençant par une majuscule>

(rappelez-vous de la convention de nommage Java).

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();
double d = sc.nextDouble();
long l = sc.nextLong();
byte b = sc.nextByte();
//Etc.
```

Remarque : il y a un type de variables primitives qui n'est pas pris en compte par la classe Scanner : il s'agit du type char. Voici comment on pourrait récupérer un caractère :

```
System.out.println("Saisissez une lettre :");
Scanner sc = new Scanner(System.in);
String str = sc.nextLine();
char carac = str.charAt(0);
System.out.println("Vous avez saisi le caractère : " + carac);
```

Qu'avons-nous fait ici ?

Nous avons récupéré une chaîne de caractères, puis utilisé une méthode de l'objet String (ici, charAt(0)) afin de récupérer le premier caractère saisi. Même si vous tapez une longue chaîne de caractères, l'instruction charAt(0) ne renverra que le premier caractère. Vous devez vous demander pourquoi charAt(0) et non charAt(1) : nous aborderons ce point lorsque nous verrons les tableaux ...

Jusqu'à ce qu'on aborde les exceptions, je vous demanderai d'être rigoureux et de faire attention à ce que vous attendez comme type de données afin d'utiliser la méthode correspondante.

Une précision s'impose, toutefois : la méthode nextLine() récupère le contenu de toute la ligne saisie et replace la « tête de lecture » au début d'une autre ligne. Par contre, si vous avez invoqué une méthode comme nextInt(), nextDouble() et que vous invoquez directement après la méthode nextLine(), celle-ci ne vous invitera pas à saisir une chaîne de caractères : elle videra la ligne commencée par les autres instructions. En effet, celles-ci ne repositionnent pas la tête de lecture, l'instruction nextLine() le fait à leur place. Pour faire simple, ceci :

```
import java.util.Scanner;

public class Main {
   public static void main(String[] args){
      Scanner sc = new Scanner(System.in);
      System.out.println("Saisissez un entier : ");
      int i = sc.nextInt();
      System.out.println("Saisissez une chaîne : ");
      String str = sc.nextLine();
      System.out.println("FIN ! ");
   }
}
```

... ne vous demandera pas de saisir une chaîne et affichera directement « Fin ». Pour pallier ce problème, il suffit de vider la ligne après les instructions ne le faisant pas automatiquement :

```
import java.util.Scanner;

public class Main {
   public static void main(String[] args){
      Scanner sc = new Scanner(System.in);
      System.out.println("Saisissez un entier : ");
      int i = sc.nextInt();
      System.out.println("Saisissez une chaîne : ");
      //On vide la ligne avant d'en lire une autre
      sc.nextLine();
      String str = sc.nextLine();
      System.out.println("FIN ! ");
   }
}
```