Partie 1 : Bien commencer en Java

Bon, vous ne connaissez rien à Java ? Eh bien c'est par ici que ça se passe ! Java est normalement un langage fait pour développer des applications graphiques, mais pour arriver à cela, nous devons tout de même passer par la programmation Java en mode console.

Donc, dans cette première partie, nous allons voir les bases du langage, ainsi que leur fidèle compagnon Eclipse.

Installer les outils de développement

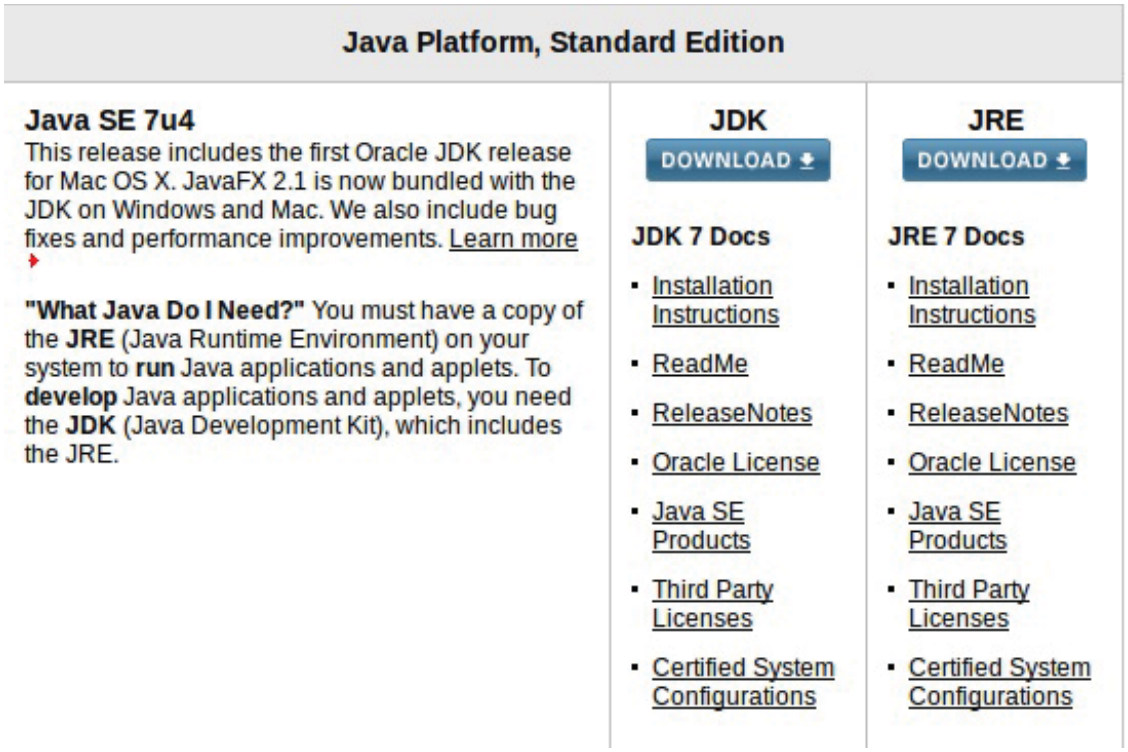
L'un des principes phares de Java réside dans sa machine virtuelle : celle-ci assure à tous les développeurs Java qu'un programme sera utilisable avec tous les systèmes d'exploitation sur lesquels est installée une machine virtuelle Java.
Lors de la phase de compilation de notre code source, celui-ci prend une forme intermédiaire appelée **byte code** : 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 **JRE** (**J**ava **R**untime **E**nvironment). 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.

Afin de nous simplifier la vie, nous allons utiliser un outil de développement, ou **IDE** (**I**ntegrated **D**evelopment **E**nvironment), pour nous aider à écrire nos futurs codes source… Nous allons donc avoir besoin de différentes choses afin de pouvoir créer des programmes Java : la première est ce fameux JRE !

Installer les outils nécessaires

JRE ou JDK

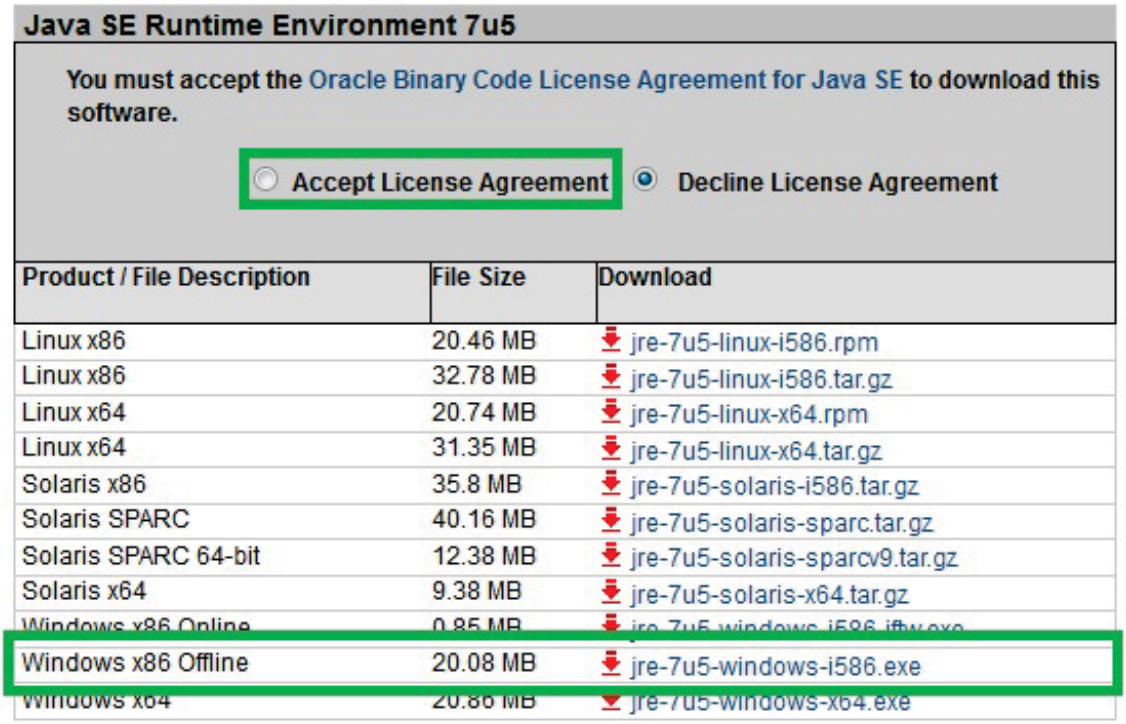
Commencez par télécharger l'environnement Java sur le site d'Oracle, comme le montre la figure suivante. Choisissez la dernière version stable.



Encart de téléchargement

Vous avez sans doute remarqué qu'on vous propose de télécharger soit le JRE, soit le **JDK** (**J**ava **D**evelopment **K**it). La différence entre ces deux environnements est écrite, mais pour les personnes fâchées avec l'anglais, sachez que le JRE contient tout le nécessaire pour que vos programmes Java puissent être exécutés sur votre ordinateur ; le JDK, en plus de contenir le JRE, contient tout le nécessaire pour développer, compiler…

L'IDE contenant déjà tout le nécessaire pour le développement et la compilation, nous n'avons besoin que du JRE. Une fois que vous avez cliqué sur `Download JRE`, vous arrivez sur la page représentée à la figure suivante.



Choix du système d'exploitation

Cochez la case : `Accept License Agreement` puis cliquez sur le lien correspondant à votre système d'exploitation (`x86` pour un système 32 bits et `x64` pour un système 64 bits). Une popup de téléchargement doit alors apparaître.

Je vous ai dit que 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*, celui qui nous intéresse dans cet ouvrage) : 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 l'outil que nous allons utiliser pour programmer.

Eclipse IDE est lui-même principalement écrit en Java.

Je vous invite donc à télécharger Eclipse IDE. Une fois la page de téléchargement choisissez `Eclipse IDE for Java Developers`, en choisissant la version d'Eclipse correspondant à votre OS (*Operating System* = système d'exploitation), comme indiqué à la figure suivante.

Version Windows d'Eclipse IDE

Sélectionnez maintenant le miroir que vous souhaitez utiliser pour obtenir Eclipse. Voilà, vous n'avez plus qu'à attendre la fin du

téléchargement.

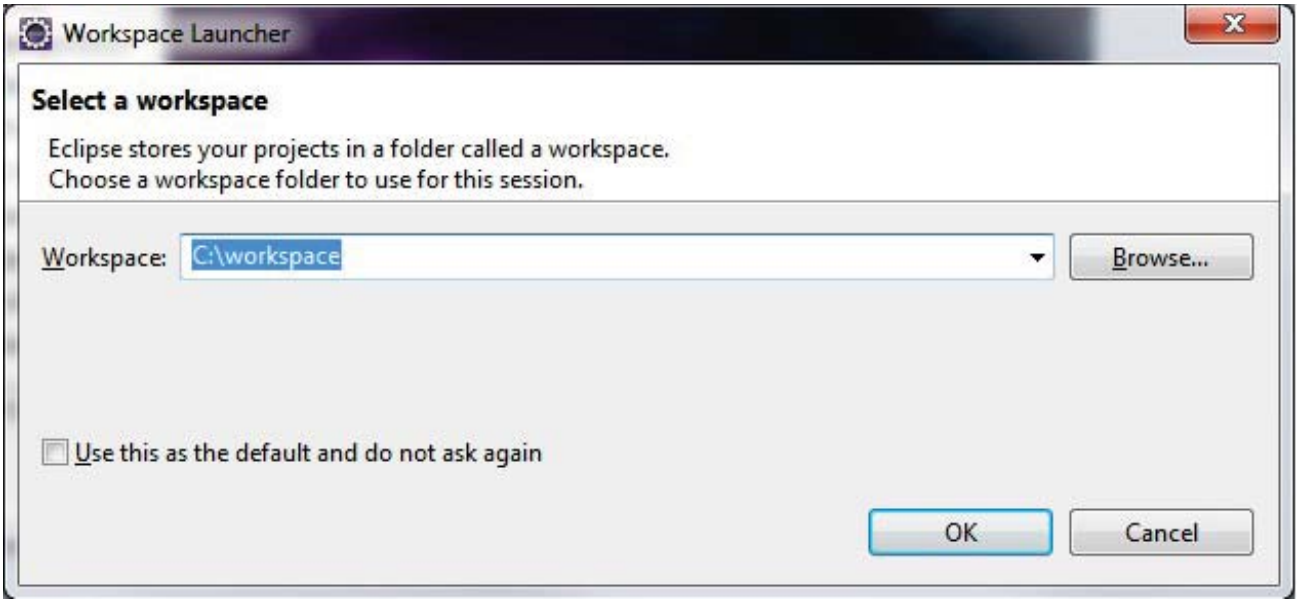
Pour ceux qui l'avaient deviné, Eclipse est le petit logiciel qui va nous permettre de développer nos applications ou nos applets, et aussi celui qui va **compiler** tout ça. Notre logiciel va donc permettre de traduire nos futurs programmes Java en langage **byte code**, compréhensible uniquement par votre JRE, fraîchement installé.

La spécificité d'Eclipse IDE vient du fait que son architecture est totalement développée autour de la notion de *plugin*. Cela signifie que toutes ses fonctionnalités sont développées en tant que plugins. Pour faire court, si vous voulez ajouter des fonctionnalités à Eclipse, vous devez :

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

Lorsque vous téléchargez un nouveau plugin 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 plugin 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, entrez dans ce dossier et lancez Eclipse. Au démarrage, comme le montre la 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. Une fois cette étape effectuée, vous arrivez sur la page d'accueil d'Eclipse. Si vous avez envie d'y jeter un oeil, allez-y !



Vous devez indiquer où enregistrer vos projets

Présentation rapide de l'interface

Je vais maintenant vous faire faire un tour rapide de l'interface d'Eclipse. Voici les principaux menus :

- `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.
- `Edit` : Dans ce menu, nous pourrons utiliser les commandes « `copier` » , « `coller` », etc.

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

La barre d'outils

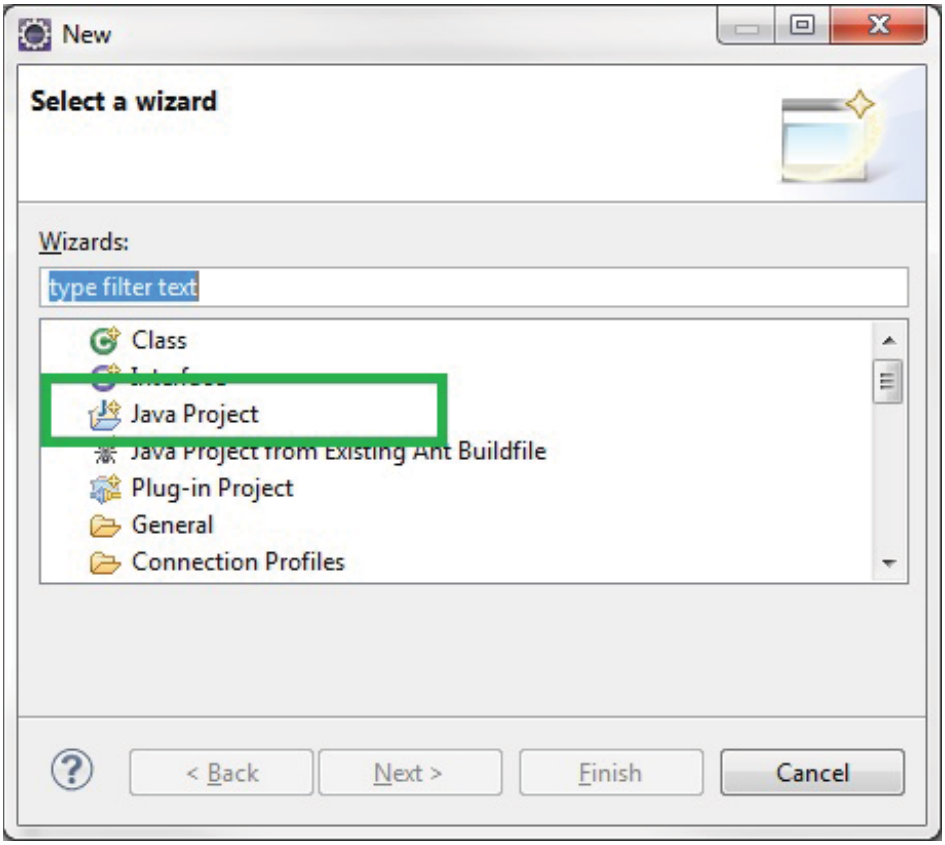
La barre d'outils ressemble à la figure suivante.

La barre d'outils d'Eclipse

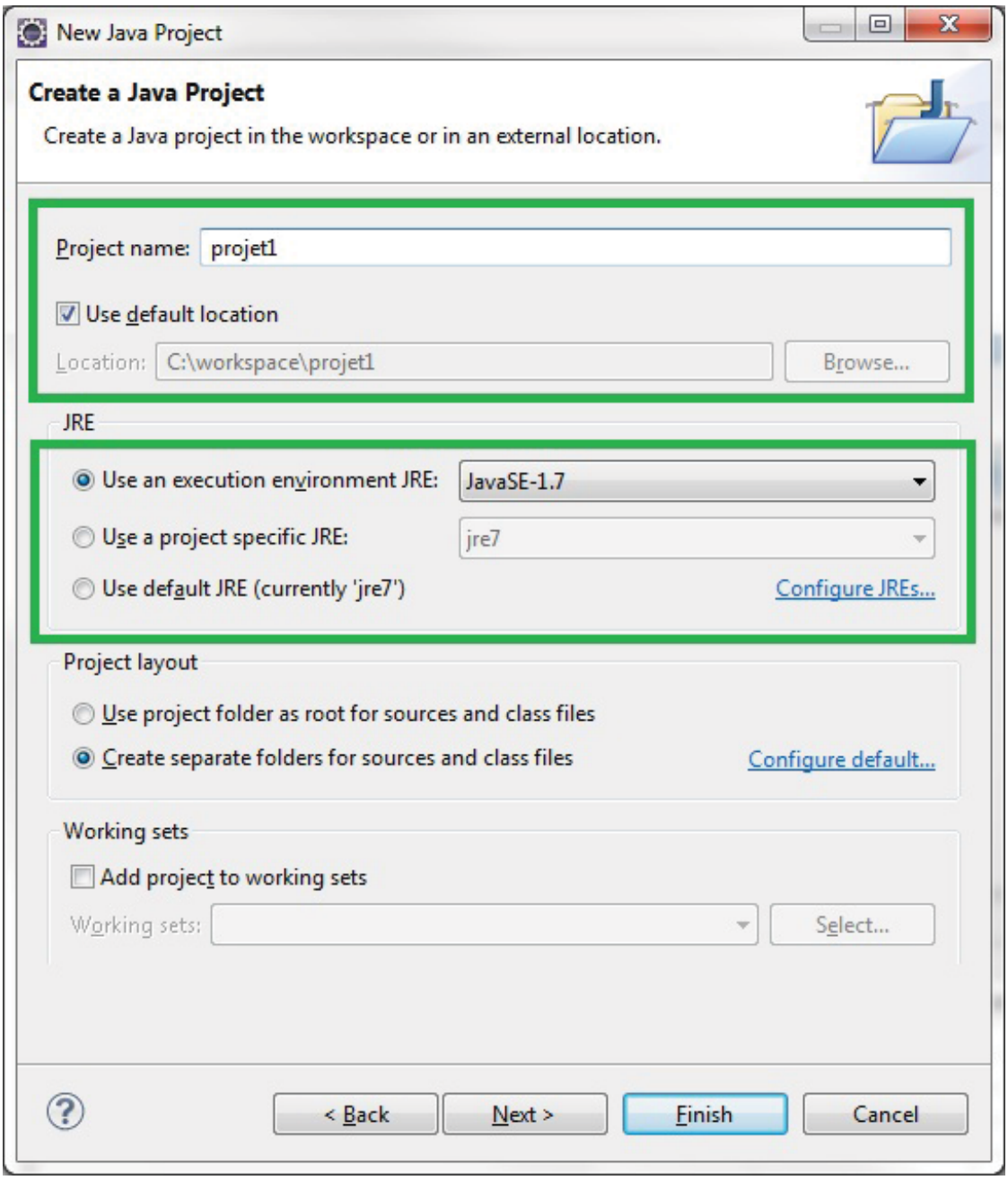
Nous avons dans l'ordre :

1. `nouveau général` : cliquer sur ce bouton revient à faire `Fichier > Nouveau` ;
2. `enregistrer` : revient à faire CTRL + S ;
3. imprimer : ai-je besoin de préciser ?
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` : créer un nouveau fichier. Cela revient à faire `Fichier > Nouveau > Classe`.

Maintenant, je vais vous demander de créer un nouveau projet Java, comme indiqué aux figures suivantes.



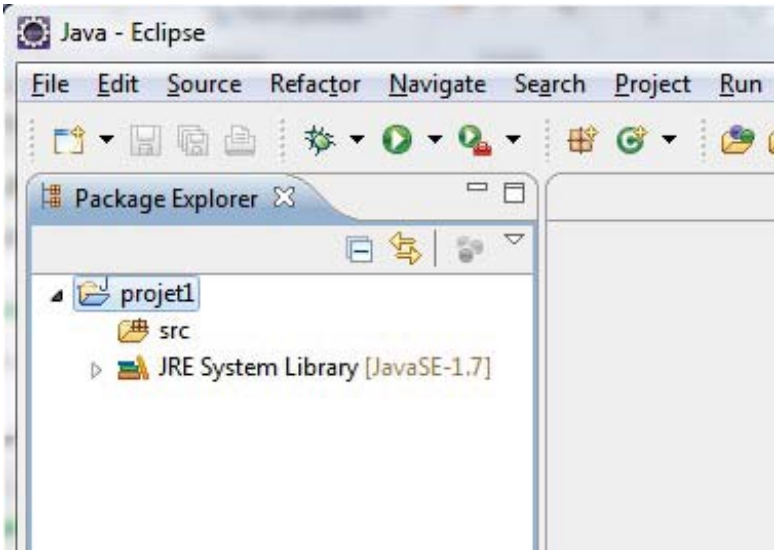
Création de projet Java - étape 1



Création de projet Java - étape 2

Renseignez le nom de votre projet comme je l'ai fait dans le premier encadré de la deuxième figure. Vous pouvez aussi voir où sera enregistré ce projet. Un peu plus compliqué, maintenant : vous avez 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, comme sur le deuxième encadré de la deuxième figure. 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`.

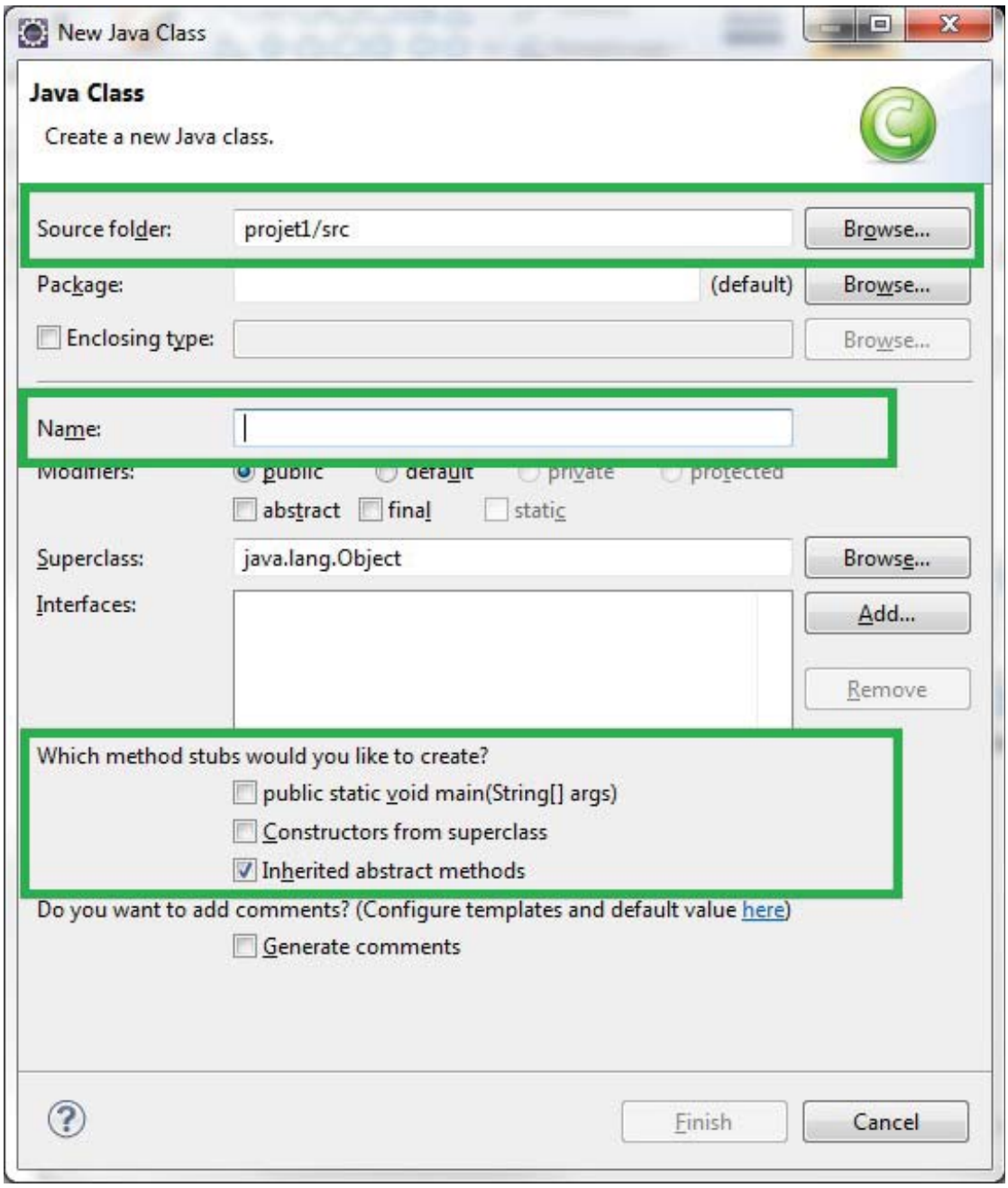
Vous devriez avoir un nouveau projet dans la fenêtre de gauche, comme à la figure suivante.



Explorateur de projet

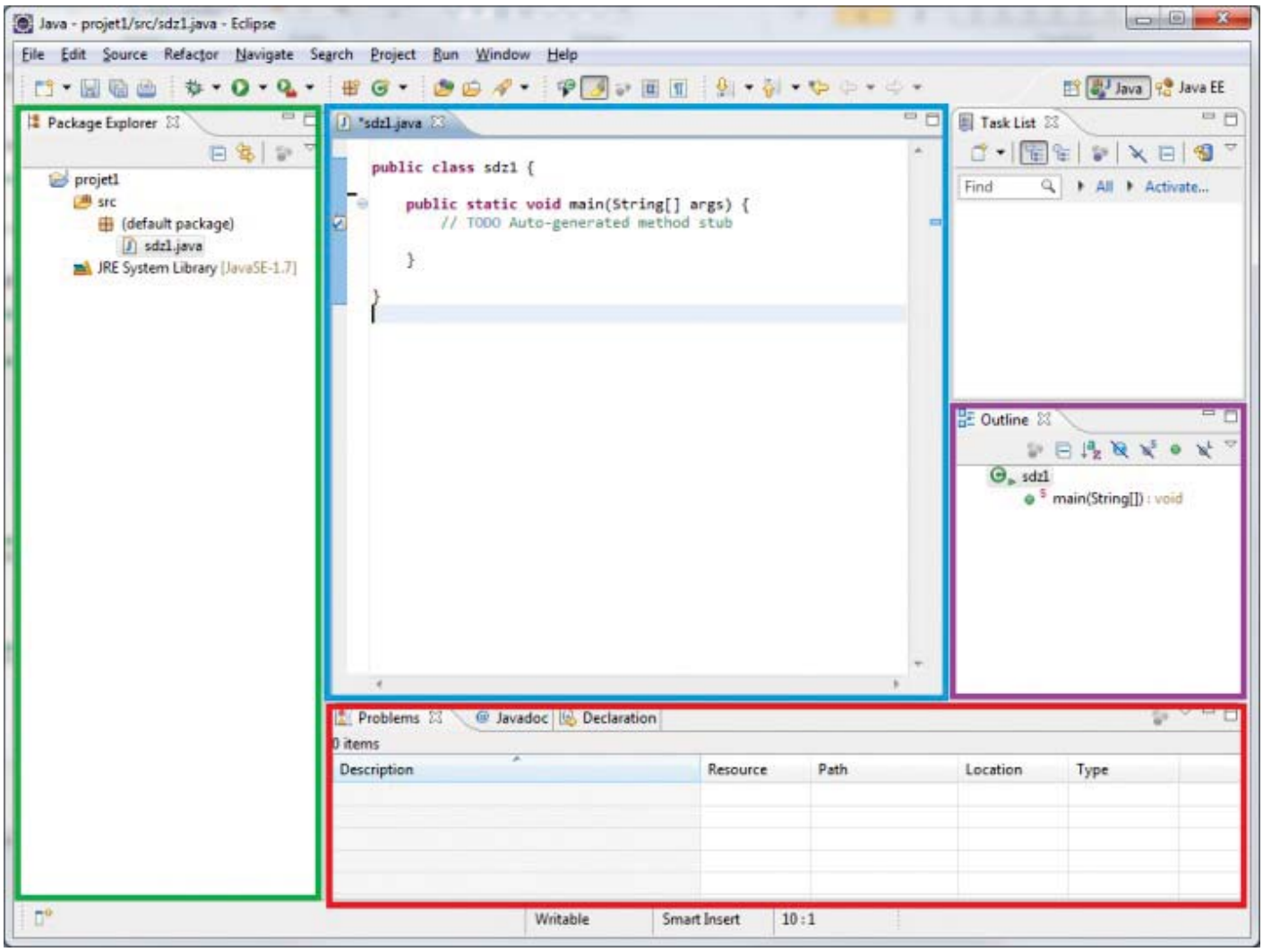
Pour boucler la boucle, ajoutons dès maintenant une nouvelle **classe** dans ce projet comme nous avons appris à le faire plus tôt *via* la barre d'outils. La figure suivante représente la fenêtre sur laquelle vous devriez tomber.

Une classe est un ensemble de codes contenant plusieurs instructions que doit effectuer votre programme. Ne vous attardez pas trop sur ce terme, nous aurons l'occasion d'y revenir.



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 ; moi, j'ai choisi « sdz1 ». Dans l'encadré 3, Eclipse vous demande si cette classe a quelque chose de particulier. Eh bien oui ! Cochez `public static void main(String[] args)` (nous reviendrons plus tard sur ce point), puis cliquez sur `Finish`. La fenêtre principale d'Eclipse se lance, comme à la figure suivante.



Fenêtre principale d'Eclipse

Avant de commencer à coder, nous allons explorer l'espace de travail. Dans l'encadré de gauche (le vert), 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 (le bleu), je pense que vous avez deviné : c'est ici que nous allons écrire nos codes source. Dans l'encadré du bas (le rouge), 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 (le violet), dès que nous aurons appris à coder nos propres fonctions et nos objets, que la liste des méthodes et des variables sera affichée.

Votre premier programme

Comme je vous l'ai maintes fois répété, les programmes Java sont, avant d'être utilisés par la machine virtuelle, précompilés en byte code (par votre IDE ou à la main). Ce byte code n'est compréhensible que par une JVM, et c'est celle-ci qui va faire le lien entre ce code et votre machine.

Vous aviez sûrement remarqué que sur la page de téléchargement du JRE, plusieurs liens étaient disponibles :

- un lien pour Windows ;
- un lien pour Mac ;
- un lien pour Linux.

Ceci, car la machine virtuelle Java se présente différemment selon qu'on se trouve sous Mac, sous Linux ou encore sous Windows. Par contre, le byte code, lui, reste le même quel que soit l'environnement avec lequel a été développé et précompilé votre programme Java. Conséquence directe : quel que soit l'OS sous lequel a été codé un programme Java, n'importe quelle machine pourra l'exécuter si elle dispose d'une JVM !

Tu n'arrêtes pas de nous rabâcher byte code par-ci, byte code par-là… Mais c'est quoi, au juste ?

Eh bien, un byte code (il existe plusieurs types de byte code, mais nous parlons ici de celui créé par Java) n'est rien d'autre qu'un code intermédiaire entre votre code Java et le code machine. Ce code particulier se trouve dans les fichiers précompilés de vos programmes ; en Java, un fichier source a pour extension `.java` et un fichier précompilé a l'extension `.class` : c'est dans ce dernier que vous trouverez du byte code. Je vous invite à examiner un fichier `.class` à la fin de cette partie (vous en aurez au moins un), mais je vous préviens, c'est illisible !

Par contre, vos fichiers `.java` sont de simples fichiers texte dont l'extension a été changée. Vous pouvez donc les ouvrir, les créer ou encore les mettre à jour avec le Bloc-notes de Windows, par exemple. Cela implique que, si vous le souhaitez, vous pouvez écrire des programmes Java avec le Bloc-notes ou encore avec Notepad++.

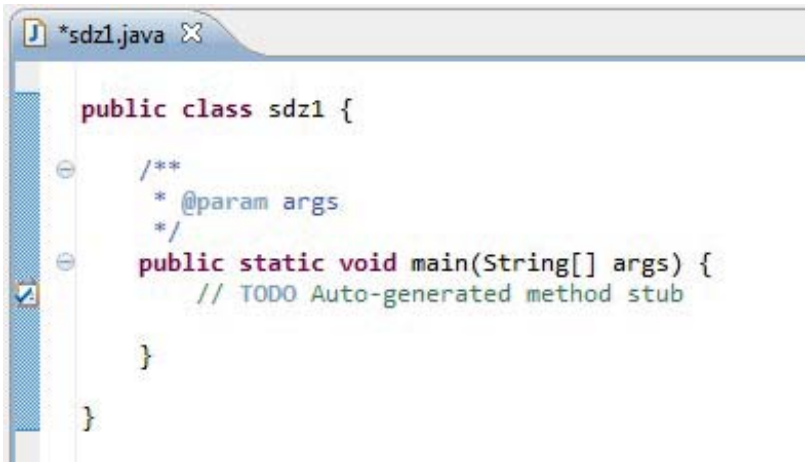
Reprenons. Vous devez savoir que **tous les programmes Java sont composés d'au moins une classe**. Elle doit contenir une méthode appelée `main` : ce sera le point de démarrage de notre programme.

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.

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.

Je vous avais demandé de créer un projet Java ; ouvrez-le ! Vous voyez la fameuse classe dont je vous parlais ? Ici, elle s'appelle « sdz1 », comme à la figure suivante. Vous pouvez voir que le mot **`class`** est précédé du mot **`public`**, dont nous verrons la signification lorsque nous programmerons des objets.



Méthode principale

Pour le moment, ce que vous devez 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 les accolades de la méthode `main`. La syntaxe de cette méthode est toujours la même :

Code : Java

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

Excuse-nous, mais… pourquoi as-tu écrit « //Contenu de votre classe » et pas « Contenu de votre classe » ?

Bonne question ! Je vous ai dit précédemment que votre programme Java, avant de pouvoir être exécuté, doit être précompilé en

byte code. Eh bien, la possibilité de forcer le compilateur à ignorer certaines instructions existe ! C'est ce qu'on appelle des **commentaires**, et deux syntaxes sont disponibles pour commenter son texte :

1. 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 ;
2. Les commentaires multilignes : ils sont introduits par les symboles « /* » et se terminent par les symboles « */ ».

Code : Java

```
public static void main(String[] args){
  //Un commentaire
  //Un autre
  //Encore un autre

  /*
Un commentaire
Un autre
Encore un autre
*/

Ceci n'est pas un commentaire !
}
```

D'accord, mais ça sert à quoi ?

C'est simple : au début, vous ne ferez que de très petits programmes. Mais dès que vous aurez pris de la bouteille, leurs tailles et le nombre de classes qui les composeront vont augmenter. Vous serez contents de trouver quelques lignes de commentaires au début de votre classe pour vous dire à quoi elle sert, ou encore des commentaires dans une méthode qui effectue des choses compliquées afin de savoir où vous en êtes dans vos traitements…

Il existe en fait une troisième syntaxe, mais elle a une utilité particulière. Elle permettra de générer une documentation pour votre programme (on l'appelle « Javadoc » pour « Java Documentation »). Je n'en parlerai que très peu, et pas dans ce chapitre. Nous verrons cela lorsque nous programmerons des objets, mais pour les curieux, je vous conseille le très bon cours de dworkin sur ce sujet disponible sur le Site du Zéro.

À partir de maintenant et jusqu'à ce que nous programmions des interfaces graphiques, nous allons faire ce qu'on appelle des programmes procéduraux. Cela signifie que le programme s'exécutera de façon procédurale, c'est-à-dire qui s'effectue de haut en bas, une ligne après l'autre. Bien sûr, il y a des instructions qui permettent de répéter des morceaux de code, mais le programme en lui-même se terminera une fois parvenu à la fin du code. Cela vient en opposition à la programmation événementielle (ou graphique) qui, elle, est basée sur des événements (clic de souris, choix dans un menu…).

Hello World

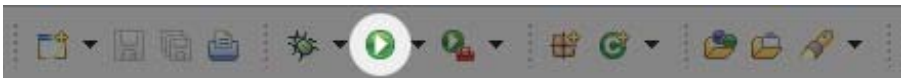
Maintenant, essayons de taper le code suivant :

Code : Java

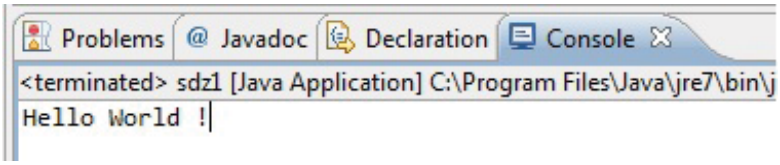
```
public static void main(String[] args){
  System.out.print("Hello World !");
}
```

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 vous souvenez bien de la présentation faite précédemment, vous devez cliquer sur la flèche blanche dans un rond vert, comme à la figure suivante.


Bouton de lancement du programme

Si vous regardez dans votre console, dans la fenêtre du bas sous Eclipse, vous devriez voir quelque chose ressemblant à la figure suivante.



La console d'Eclipse

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` » . Avant que vous arrachiez les cheveux, voici quelques précisions :

- `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 (entre les parenthèses).

Prenons le code suivant :

Code : Java

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

Lorsque vous l'exécutez, vous devriez voir des chaînes de caractères qui se suivent sans saut de ligne. Autrement dit, ceci s'affichera dans votre console :

Code : Console

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

Je me doute que vous souhaiteriez insérer un retour à la ligne pour que votre texte soit plus lisible… Pour cela, vous avez 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 : Java

```
System.out.print("Hello World ! \n");
```

```
        System.out.println("My name is");
        System.out.println("\nCysboy");
```

Avec pour résultat :

Code : Console

```
Hello World !
My name is

Cysboy
```

Vous pouvez voir que :

- lorsque vous utilisez le caractère d'échappement `\n`, quelle que soit la méthode appelée, celle-ci ajoute immédiatement un retour à la ligne à son emplacement ;
- lorsque vous utilisez la méthode `println()`, celle-ci ajoute automatiquement un retour à la ligne à la fin de la chaîne passée en paramètre ;
- un caractère d'échappement peut être mis dans la méthode `println()`.

J'en profite au passage pour vous mentionner deux autres caractères d'échappement :

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

Vous avez sûrement remarqué que la chaîne de caractères que l'on affiche est entourée par des « " ». 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 » avec un « \ », ce qui donnerait : `"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.

Je vous propose maintenant de passer un peu de temps sur la compilation de vos programmes en ligne de commande. Cette partie n'est pas obligatoire, loin de là, mais elle ne peut être qu'enrichissante.

Compilation en ligne de commande (Windows)

Bienvenue donc aux plus curieux ! Avant de vous apprendre à compiler et à exécuter un programme en ligne de commande, il va vous falloir le JDK (**J**ava SE **D**evelopment **K**it). C'est avec celui-ci que nous aurons de quoi compiler nos programmes. Le nécessaire à l'exécution des programmes est dans le JRE… mais il est également inclus dans le JDK. Je vous invite donc à retourner sur le site d'Oracle et à télécharger ce dernier. Une fois cette opération effectuée, il est conseillé de mettre à jour votre variable d'environnement `%PATH%`.

Euh… quoi ?

Votre « variable d'environnement ». C'est grâce à elle que Windows trouve des exécutables sans qu'il soit nécessaire de lui spécifier le chemin d'accès complet. Vous — enfin, Windows — en a plusieurs, mais nous ne nous intéresserons qu'à une seule. En gros, cette variable contient le chemin d'accès à certains programmes.

Par exemple, si vous spécifiez le chemin d'accès à un programme X dans votre variable d'environnement et que, par un malheureux hasard, vous n'avez plus aucun raccourci vers X, vous l'avez définitivement perdu dans les méandres de votre PC. Eh

bien vous pourrez le lancer en faisant `Démarrer > Exécuter` et en tapant la commande `X.exe` (en partant du principe que le nom de l'exécutable est « X.exe »</minicode>).

D'accord, mais comment fait-on ? Et pourquoi doit-on faire ça pour le JDK ?

J'y arrive. Une fois votre JDK installé, ouvrez le répertoire `bin` de celui-ci, ainsi que celui de votre JRE. Nous allons nous attarder sur deux fichiers.

Dans le répertoire `bin` de votre JRE, vous devez avoir un fichier nommé `java.exe`, que vous retrouvez aussi dans le répertoire `bin` de votre JDK. C'est grâce à ce fichier que votre ordinateur peut lancer vos programmes par le biais de la JVM. Le deuxième ne se trouve que dans le répertoire `bin` de votre JDK, il s'agit de `javac.exe` (Java compiler). C'est celui-ci qui va précompiler vos programmes Java en byte code.

Alors, pourquoi mettre à jour la variable d'environnement pour le JDK ? Eh bien, compiler et exécuter en ligne de commande revient à utiliser ces deux fichiers en leur précisant où se trouvent les fichiers à traiter. Cela veut dire que si l'on ne met pas à jour la variable d'environnement de Windows, il nous faudrait :

- ouvrir l'invite de commande ;
- se positionner dans le répertoire `bin` de notre JDK ;
- appeler la commande souhaitée ;
- préciser le chemin du fichier `.java` ;
- renseigner le nom du fichier.

Avec notre variable d'environnement mise à jour, nous n'aurons plus qu'à :

- nous positionner dans le dossier de notre programme ;
- appeler la commande ;
- renseigner le nom du fichier Java.

Allez dans le panneau de configuration de votre PC ; de là, cliquez sur l'icône `Système` ; choisissez l'onglet `Avancé` et vous devriez voir en bas un bouton nommé `Variables d'environnement` : cliquez dessus. Une nouvelle fenêtre s'ouvre. Dans la partie inférieure intitulée `Variables système`, cherchez la variable `Path`. Une fois sélectionnée, cliquez sur `Modifier`. Encore une fois, une fenêtre, plus petite celle-ci, s'ouvre devant vous. Elle contient le nom de la variable et sa valeur.

Ne changez pas son nom et n'effacez pas son contenu ! Nous allons juste ajouter un chemin d'accès.

Pour ce faire, allez jusqu'au bout de la valeur de la variable, ajoutez-y un point-virgule s'il n'y en a pas et ajoutez le chemin d'accès au répertoire `bin` de votre JDK, en terminant celui-ci par un point-virgule ! Chez moi, ça donne ceci : `C:\Sun\SDK\jdk\bin`.

Auparavant, ma variable d'environnement contenait, avant mon ajout :

Code : Java

```
%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;
```

Et maintenant :

Code : Java

```
%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;C:\Sun\SDK\jdk\bin;
```

Validez les changements : vous êtes maintenant prêts à compiler en ligne de commande.

Pour bien faire, allez dans le répertoire de votre premier programme et effacez le `.class`. Ensuite, faites `Démarrer > Exécuter` (ou encore touche Windows + R et tapez « cmd » .

Dans l'invite de commande, on se déplace de dossier en dossier grâce à l'instruction `cd`.
`cd <nom du dossier enfant>` pour aller dans un dossier contenu dans celui dans lequel nous nous trouvons,
`cd ..` pour remonter d'un dossier dans la hiérarchie.

Par exemple, lorsque j'ouvre la console, je me trouve dans le dossier `C:\toto\titi` et mon application se trouve dans le dossier `C:\sdz`, je fais donc :

Code : Console

```
cd ..
cd ..
cd sdz
```

Après de la première instruction, je me retrouve dans le dossier `C:\toto`. Grâce à la deuxième instruction, j'arrive à la racine de mon disque. *Via* la troisième instruction, je me retrouve dans le dossier `C:\sdz`. Nous sommes maintenant dans le dossier contenant notre fichier Java ! Cela dit, nous pouvions condenser cela en :

Code : Console

```
cd ../../sdz
```

Maintenant, vous pouvez créer votre fichier `.class` en exécutant la commande suivante :

Code : Console

```
javac <nomDeFichier.java>
```

Si, dans votre dossier, vous avez un fichier `test.java`, compilez-le en faisant :

Code : Console

```
javac test.java
```

Et si vous n'avez aucun message d'erreur, vous pouvez vérifier que le fichier `test.class` est présent en utilisant l'instruction `dir` qui liste le contenu d'un répertoire. Cette étape franchie, vous pouvez lancer votre programme Java en faisant ce qui suit :

Code : Console

```
java <nomFichierClassSansExtension>
```

Ce qui nous donne :

Code : Console

```
java test
```

Et normalement, le résultat de votre programme Java s'affiche sous vos yeux ébahis !

Attention : il ne faut pas mettre l'extension du fichier pour le lancer, mais il faut la mettre pour le compiler.

Voilà : vous avez compilé et exécuté un programme Java en ligne de commande… Vous avez pu voir qu'il n'y a rien de vraiment compliqué et, qui sait, vous en aurez peut-être besoin un jour.

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.