Cyrille Herby

APPRENEZ À PROGRAMMER EN

JAVA

3e édition

OPENCLASSROOMS

EYROLLES

APPRENEZ À PROGRAMMER EN

JAVA

3e édition

Vous aimeriez apprendre à programmer en Java, mais vous débutez dans la programmation ? Pas de panique ! Vous tenez dans vos mains un livre conçu pour les débutants qui souhaitent se former à Java, le langage de programmation incontournable des professionnels ! De quoi permettre d'apprendre pas à pas à développer vos premiers programmes.

QU'ALLEZ-VOUS APPRENDRE ?

Bien commencer en Java

- Installer les outils de développement
- Les variables et les opérateurs
- Lire les entrées clavier
- Les conditions
- Les boucles

Java orienté objet

- Votre première classe
- L'héritage
- Les classes abstraites et les interfaces
- Java 8 ou la révolution des interfaces
- Les exceptions
- Les collections d'objets
- La généricité en Java
- Les flux d'entrées sorties
- Java et la réflexivité
- Classes anonymes, interfaces fonctionnelles, lambdas et références de méthode
- Manipuler vos données avec les streams
- La nouvelle API de gestion des dates de Java 8
- Une JVM modulaire avec Java 9

Java et la programmation événementielle

- Votre première fenêtre
- Positionner des boutons
- Interagir avec des boutons
- Les champs de formulaire
- Les menus et boîtes de dialogue
- Conteneurs, sliders et barres de progression
- Les arbres et leur structure
- Les interfaces de tableaux
- Mieux structurer son code : le pattern MVC
- Le drag'n drop
- Mieux gérer les interactions avec les composants

Initiation à Java FX

- Introduction et installation des outils
- Lier un modèle à votre vue
- Interagir avec vos composants
- Java FX a du style !

Interaction avec les bases de données

- JDBC : la porte d'accès aux bases de données
- Fouiller dans sa base de données
- Limiter le nombre de connexions

À PROPOS DE L'AUTEUR

Auditeur en sécurité, Cyrille Herby travaille sur des projets Java depuis plusieurs années. Il a notamment été administrateur système et réseau, puis responsable de la sécurité des systèmes d'information au sein du groupe Cordon Electronics. Il a débuté dans la programmation en découvrant OpenClassrooms il y a plusieurs années déjà. Désormais, il y rédige à son tour des cours pour montrer qu'il est possible de s'initier à la programmation et ses concepts sans avoir un dictionnaire à la main… et surtout, sans migraine !

L'ESPRIT D'OPENCLASSROOMS

Des cours ouverts, riches et vivants, conçus pour tous les niveaux et accessibles à tous gratuitement sur notre plate-forme d'e-éducation : www.openclassrooms.com. Vous y vivrez une véritable expérience communautaire de l'apprentissage, permettant à chacun d'apprendre avec le soutien et l'aide des autres étudiants sur les forums. Vous profiterez des cours disponibles partout, tout le temps.

www.editions-eyrolles.com

APPRENEZ À PROGRAMMER EN

JAVA

DANS LA MÊME COLLECTION

É. Lalitte. – **Apprenez le fonctionnement des réseaux TCP/IP.**
N° 67477, 3e édition, 2018, 304 pages.

M. Nebra. – **Concevez votre site web avec PHP et MySQL.**
N° 67475, 3e édition, 2017, 392 pages.

M. Nebra. – **Réalisez votre site web avec HTML 5 et CSS 3.**
N° 67476, 2e édition, 2017, 362 pages.

V. Thuillier. – **Programmez en orienté objet en PHP.**
N° 14472, 2e édition, 2017, 474 pages.

J. Pardanaud, S. de la Marck. – **Découvrez le langage JavaScript.**
N° 14399, 2017, 478 pages.

A. Bacco. – **Développez votre site web avec le framework Symfony3.**
N° 14403, 2016, 536 pages.

M. Chavelli. – **Découvrez le framework PHP Laravel.**
N° 14398, 2016, 336 pages.

R. De Visscher. – **Découvrez le langage Swift.**
N° 14397, 2016, 128 pages.

M. Lorant. – **Développez votre site web avec le framework Django.**
N° 21626, 2015, 285 pages.

M. Nebra, M. Schaller. – **Programmez avec le langage C++.**
N° 21622, 2015, 674 pages.

SUR LE MÊME THÈME

C. Delannoy. – **Programmer en Java.**
N° 67536, 10e édition, 2017, 984 pages.

A. Tasso. – **Le livre de Java premier langage.**
N° 67486, 12e édition, 2017, 616 pages.

Cyrille Herby

APPRENEZ À PROGRAMMER EN

JAVA

3e édition

EYROLLES

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

ISBN : 978-2-212-67521-4

Avant-propos

Si vous lisez ces lignes, c'est que nous avons au moins deux choses en commun : l'informatique vous intéresse et vous avez envie d'apprendre à programmer. Enfin, quand je dis en commun, je voulais dire en commun avec moi au moment où je voulais apprendre la programmation.

Pour moi, tout a commencé sur un site maintenant très connu : le Site du Zéro (maintenant OpenClassrooms). Étant débutant et cherchant à tout prix des cours adaptés à mon niveau, je suis naturellement tombé amoureux de ce site qui propose des cours d'informatique accessibles au plus grand nombre. Vous l'aurez sans doute remarqué, trouver un cours d'informatique simple et clair (sur les réseaux, les machines, la programmation…) est habituellement un vrai parcours du combattant.

Je ne me suis pas découragé et je me suis professionnalisé, via une formation diplômante, tout en suivant l'actualité de mon site préféré… Au sein de cette formation, j'ai pu voir divers aspects de mon futur métier, notamment la programmation dans les langages PHP, C#, JavaScript et, bien sûr, Java. Très vite, j'ai aimé travailler avec ce dernier, d'une part parce qu'il est agréable à manipuler, souple à utiliser en demandant toutefois de la rigueur (ce qui oblige à structurer ses programmes), et d'autre part parce qu'il existe de nombreuses ressources disponibles sur Internet (mais pas toujours très claires pour un débutant).

J'ai depuis obtenu mon diplôme et trouvé un emploi, mais je n'ai jamais oublié la difficulté des premiers temps. Comme le Site du Zéro permettait d'écrire des tutoriels et de les partager avec la communauté, j'ai décidé d'employer les connaissances acquises durant ma formation et dans mon travail à rédiger un tutoriel permettant d'aborder mon langage de prédilection avec simplicité. J'ai donc pris mon courage à deux mains et j'ai commencé à écrire. Beaucoup de lecteurs se sont rapidement montrés intéressés, pour mon plus grand plaisir.

De ce fait, mon tutoriel a été mis en avant sur le site et, aujourd'hui, il est adapté dans la collection « Livre du Zéro ». Je suis heureux du chemin parcouru, heureux d'avoir pu aider tant de débutants et heureux de pouvoir vous aider à mon tour !

Et Java dans tout ça ?

Java est un langage de programmation très utilisé, notamment par un grand nombre de développeurs professionnels, ce qui en fait un langage incontournable actuellement.

Voici les caractéristiques de Java en quelques mots.

Java est un langage de programmation moderne développé par Sun Microsystems, aujourd'hui racheté par Oracle. Il ne faut surtout pas le confondre avec JavaScript (langage de script utilisé sur les sites web), car ils n'ont rien à voir.

- Une de ses plus grandes forces est son excellente portabilité : une fois votre programme créé, il fonctionnera automatiquement sous Windows, Mac, Linux, etc.
- On peut faire de nombreux types de programmes avec Java :
  - des applications, sous forme de fenêtre ou de console, des applets, qui sont des programmes Java incorporés à des pages web ;
  - des applications pour appareils mobiles, comme les smartphones, avec J2ME (Java 2 Micro Edition) ;
  - des sites web dynamiques, avec J2EE (Java 2 Enterprise Edition, maintenant JEE) ;
  - et bien d'autres : JMF (Java Media Framework), J3D pour la 3D…

Comme vous le voyez, Java permet de réaliser une très grande quantité d'applications différentes ! Mais… comment apprendre un langage si vaste qui offre tant de possibilités ? Heureusement, ce livre est là pour tout vous apprendre sur Java à partir de zéro.

Java est donc un langage de programmation, un langage dit compilé : il faut comprendre par là que ce que vous allez écrire n'est pas directement compréhensible et utilisable par votre ordinateur. Nous devrons donc passer par une étape de compilation (étape obscure où votre code source est entièrement transformé). En fait, on peut distinguer trois grandes phases dans la vie d'un code Java :

- la phase d'écriture du code source, en langage Java ;
- la phase de compilation de votre code ;
- la phase d'exécution.

Ces phases sont les mêmes pour la plupart des langages compilés (C, C++…). Par contre, ce qui fait la particularité de Java, c'est que le résultat de la compilation n'est pas directement utilisable par votre ordinateur. Les langages mentionnés ci-dessus permettent de faire des programmes directement compréhensibles par votre machine après compilation, mais avec Java, c'est légèrement différent. En C++ par exemple, si vous voulez faire en sorte que votre programme soit exploitable sur une machine utilisant Windows et sur une machine utilisant Linux, vous allez devoir prendre en compte

les spécificités de ces deux systèmes d'exploitation dans votre code source et compiler une version spéciale pour chacun d'eux.

Avec Java, c'est un programme appelé **la machine virtuelle** qui va se charger de retranscrire le résultat de la compilation en langage machine, interprétable par celle-ci. Vous n'avez pas à vous préoccuper des spécificités de la machine qui va exécuter votre programme : la machine virtuelle Java s'en charge pour vous !

Structure de l'ouvrage

Ce livre a été conçu en partant du principe que vous ne connaissez rien à la programmation. Voilà le plan en quatre parties que nous allons suivre tout au long de cet ouvrage.

1. Les bases de Java – Nous verrons ici ce qu'est Java et comment il fonctionne. Nous créerons notre premier programme, en utilisant des variables, des opérateurs, des conditions, des boucles… Nous apprendrons les bases du langage, qui vous seront nécessaires par la suite.

2. Java et la Programmation orientée objet – Après avoir dompté les bases du langage, vous allez devoir apprivoiser une notion capitale : l'objet. Vous apprendrez à encapsuler vos morceaux de code avant de les rendre modulables et réutilisables, mais il y aura du travail à fournir.

3. Les interfaces graphiques – Là, nous verrons comment créer des interfaces graphiques et comment les rendre interactives. C'est vrai que, jusqu'à présent, nous avons travaillé en mode console. Il faudra vous accrocher un peu car il y a beaucoup de composants utilisables, mais le jeu en vaut la chandelle ! Nous passerons en revue différents composants graphiques tels que les champs de texte, les cases à cocher, les tableaux, les arbres ainsi que quelques notions spécifiques comme le drag'n drop.

4. Interactions avec les bases de données – De nos jours, avec la course aux données, beaucoup de programmes doivent interagir avec ce qu'on appelle des bases de données. Dans cette partie, nous verrons comment s'y connecter, comment récupérer des informations et comment les exploiter.

Comment lire ce livre ?

Suivez l'ordre des chapitres

Lisez ce livre comme on lit un roman. Il a été conçu pour cela. Contrairement à beaucoup de livres techniques où il est courant de lire en diagonale et de sauter certains chapitres, il est ici fortement recommandé de suivre l'ordre du livre, à moins que vous ne soyez déjà, au moins un peu, expérimenté.

Pratiquez en même temps

Pratiquez régulièrement. N'attendez pas d'avoir fini de lire ce livre pour allumer votre ordinateur.

Les compléments web

Pour télécharger le code source des exemples de cet ouvrage, veuillez-vous rendre à cette adresse : https://www.editions-eyrolles.com/dl/0067521.

Remerciements

Comme pour la plupart des ouvrages, beaucoup de personnes ont participé de près ou de loin à l'élaboration de ce livre et j'en profite donc pour les en remercier.

Ma compagne, Manuela, qui me supporte et qui tolère mes heures passées à écrire les tutoriels pour OpenClassrooms. Un merci spécial à toi qui me prends dans tes bras lorsque ça ne va pas, qui m'embrasse lorsque je suis triste, qui me souris lorsque je te regarde, qui me donnes tant d'amour lorsque le temps est maussade : pour tout ça et plus encore, je t'aime.

Mes deux filles Jaana et Madie qui me donnent tant de joie et de bonheur au quotidien.

Agnès Haasser (Tûtie), Damien Smeets (Karl Yeurl), Mickaël Salamin (micky), François Glorieux (Nox), Christophe Tafani-Dereeper, Romain Campillo (Le Chapelier Toqué), Charles Dupré (Barbatos), Maxence Cordiez (Ziame), Philippe Lutun (ptipilou) et les relecteurs m'ayant accompagné dans la correction de cet ouvrage.

Mathieu Nebra (alias M@teo21), père fondateur d'OpenClassrooms, qui m'a fait confiance, soutenu dans mes démarches et qui m'a donné de précieux conseils.

Jessica Mautref, Marine Gallois et Laurène Gibaud qui m'ont accompagné chez OpenClassrooms pendant un bon nombre d'années et qui sont des perles de gentillesse et de compétences.

Tous les Zéros qui m'ont apporté leur soutien et leurs remarques.

Toutes les personnes qui m'ont contacté pour me faire des suggestions et m'apporter leur expertise.

Merci aussi à toutes celles et ceux qui m'ont apporté leur soutien et qui me permettent d'apprendre toujours plus au quotidien, mes collègues de travail :

- Thomas, qui a toujours des questions sur des sujets totalement délirants ;
- Angelo, mon chef adoré, qui est un puits de science en informatique ;
- Olivier, la force zen, qui n'a pas son pareil pour aller droit au but ;
- Dylan, discret mais d'une compétence plus que certaine dans des domaines aussi divers que variés ;
- Jérôme, que j'ai martyrisé mais qui, j'en suis persuadé, a adoré… :-)

Table des matières

Première partie

Bien commencer en Java

1 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.

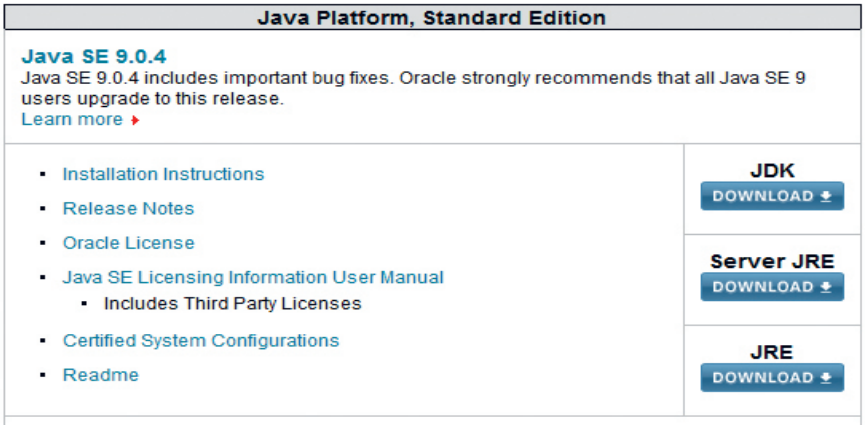
Afin de nous simplifier la vie, nous allons utiliser un outil de développement, ou **IDE** *(**I**ntegrated **D**evelopment **E**nvironment)*, pour nous aider à écrire nos futurs codes sources… Nous allons donc avoir besoin de différentes choses afin de pouvoir créer des programmes Java : la première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

Installer les outils nécessaires

JRE ou JDK

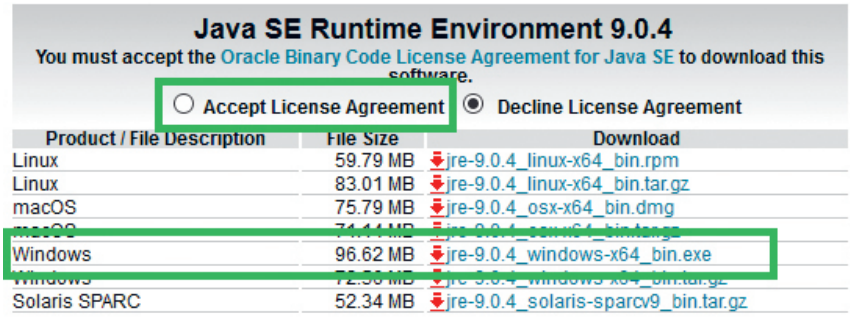
Commencez par télécharger l'environnement Java sur le site d'Oracle, comme le montre la figure page suivante. Choisissez la dernière version stable.

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 indiquée sur le site d'Oracle, 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…



Encart de téléchargement

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 fenêtre 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) : il permet de développer des applications dites « client lourd », par exemple Microsoft Word ou Excel, la suite OpenOffice.org… C'est ce que nous allons faire dans cet ouvrage.

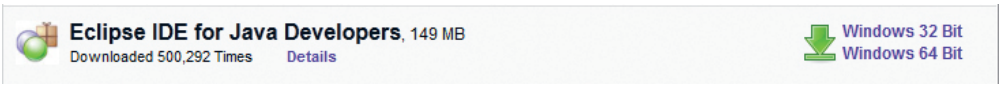
- **J2EE** *(Java 2 Enterprise Edition)* : il permet de développer des applications web en Java. On parle aussi de « clients légers ».
- **J2ME** *(Java 2 Micro Edition)* : il permet de développer des applications pour appareils mobiles, comme des téléphones portables, des PDA…

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 sur le site *https://www.eclipse.org/*. Une fois la page de téléchargement ouverte, choisissez ***Eclipse IDE for Java Developers***, en choisissant la version d'Eclipse correspondant à votre OS, 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.

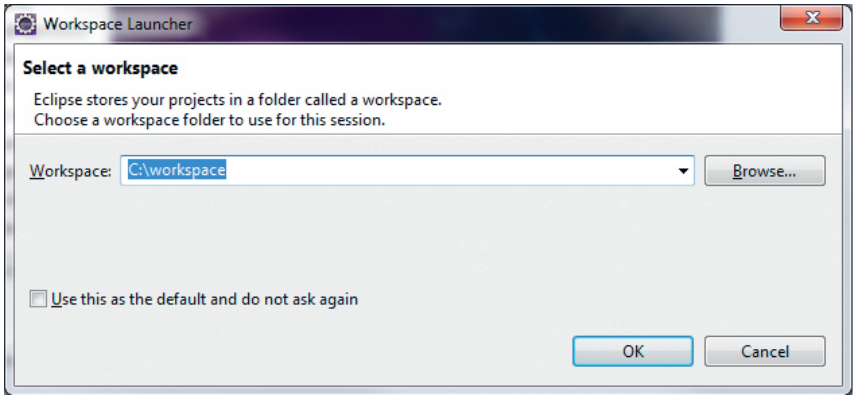
La spécificité d'Eclipse IDE vient du fait que son architecture est totalement développée autour de la notion de plug-in. Cela signifie que toutes ses fonctionnalités sont développées en tant que plug-ins. Pour faire court, si vous voulez ajouter des fonctionnalités à Eclipse, vous devez :

- télécharger le plug-in correspondant ;
- copier les fichiers dans les répertoires spécifiés ;
- démarrer Eclipse.

Et c'est tout !

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, 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 œil, allez-y !



Vous devez indiquer où enregistrer vos projets.

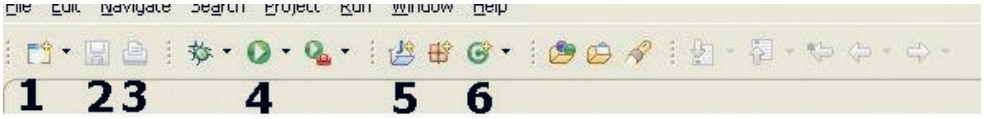
Présentation rapide de l'interface

Je vais maintenant vous présenter rapidement 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 clavier 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*** : ce menu nous permet de configurer Eclipse selon nos besoins.

La barre d'outils

La barre d'outils ressemble à celle de la figure ci-dessous.

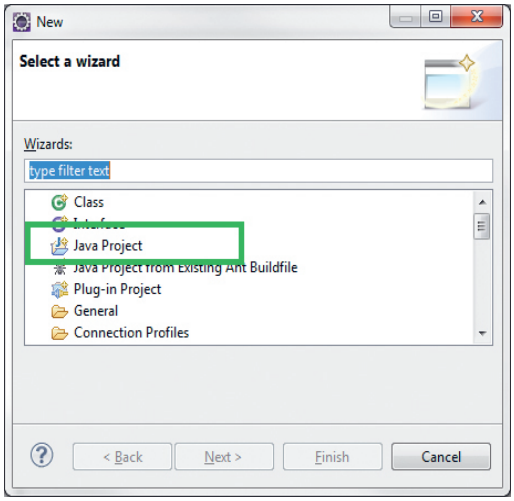


La barre d'outils d'Eclipse

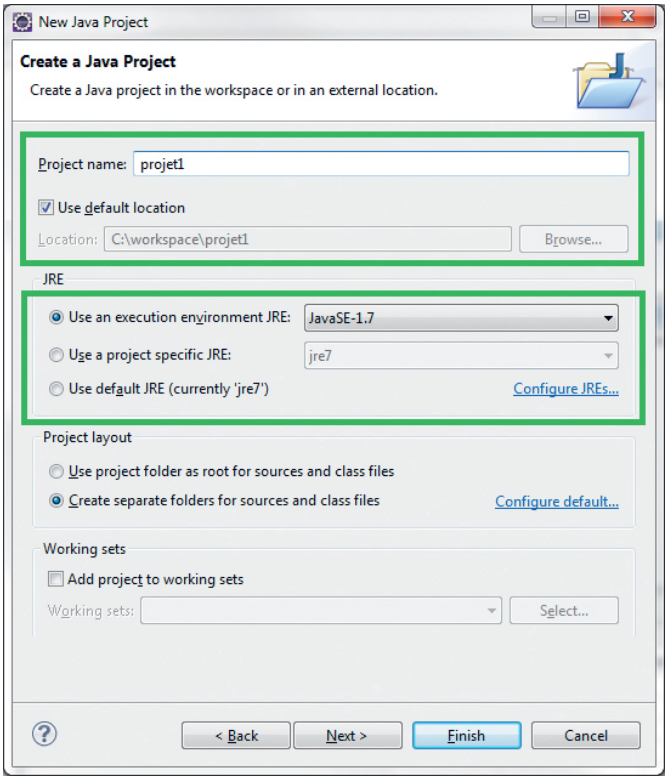
Voici les icônes de la barre d'outils, de gauche à droite :

1. ***Nouveau général*** : cliquer sur cette icône revient à sélectionner le menu ***Fichier>Nouveau*** ;
2. ***Enregistrer*** : cette icône à la même action que le raccourci ***Ctrl+S*** ;
3. ***Imprimer*** : ai-je besoin de préciser l'action de cette icône ?
4. ***Exécuter la classe ou le projet spécifié*** : nous verrons ceci plus en détail par la suite ;
5. ***Créer un nouveau projet*** : cliquer sur cette icône revient à sélectionner le menu ***Fichier>Nouveau>Java Project*** ;
6. ***Créer une nouvelle classe*** : cette icône permet de créer un nouveau fichier. Cela revient à sélectionner le menu ***Fichier>Nouveau>Classe***.

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

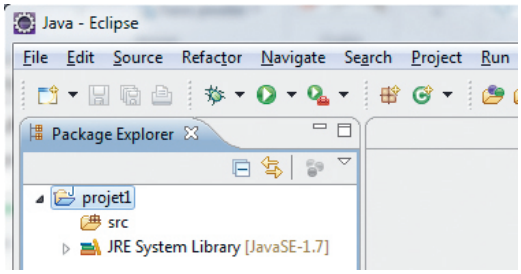


Création de projet Java – étape 1



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 seul environnement Java sur votre machine, mais si vous en aviez plusieurs, vous pourriez aussi spécifier à Eclipse quel JRE utiliser pour ce projet, comme sur le second encadré. Vous pourrez changer ceci à tout moment dans Eclipse via le menu ***Window>Preferences***, en dépliant l'arbre ***Java*** dans la fenêtre et en choisissant ***Installed JRE***.

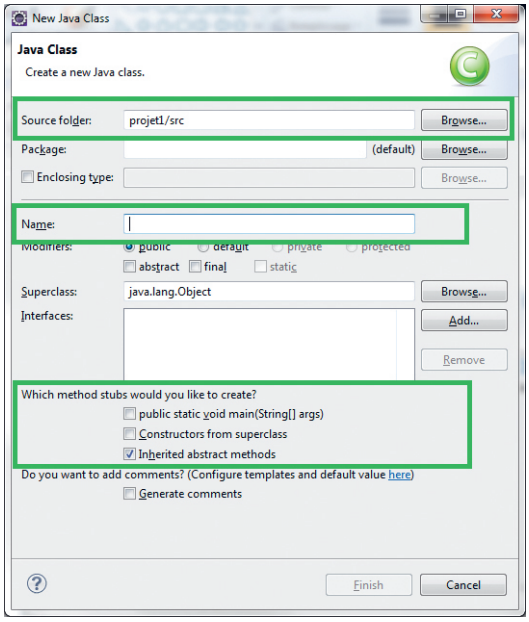


Explorateur de projets

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

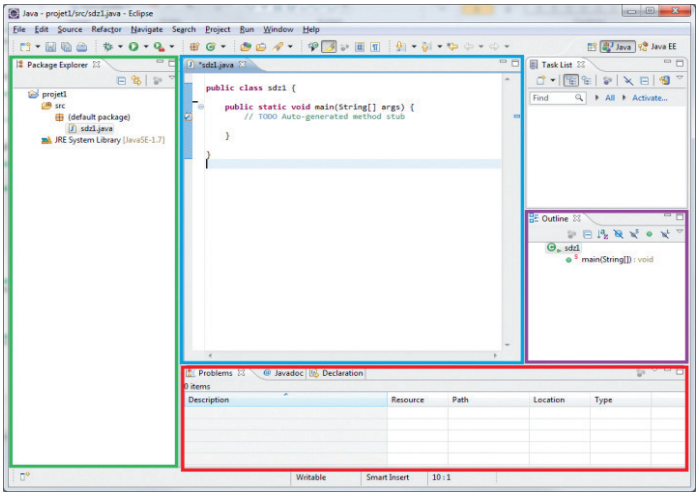
Pour boucler la boucle, ajoutons dès maintenant une nouvelle **classe** dans ce projet comme nous avons appris à le faire précédemment via la barre d'outils. La figure suivante représente la fenêtre que vous devriez obtenir.

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 le premier encadré de la figure, nous pouvons voir où seront enregistrés nos fichiers Java (paramètre ***Source folder***). Nommez votre classe Java grâce au champ ***Name***, ici j'ai choisi `sdz1`. Dans le troisième encadré, Eclipse vous demande si cette classe a quelque chose de particulier. Eh bien oui ! Cochez l'option ***public static void main(String[] args)*** (nous reviendrons plus tard sur ce point), puis cliquez sur le bouton ***Finish***. La fenêtre principale d'Eclipse s'ouvre alors comme à la figure page suivante.



Fenêtre principale d'Eclipse

Avant de commencer à coder, nous allons explorer l'espace de travail. Dans l'encadré de gauche (en vert sur la figure), vous trouverez le dossier de votre projet ainsi que son contenu. Vous pourrez ici gérer votre projet comme bon vous semble (ajout, suppression de données…). Pour l'encadré central (en bleu sur la figure), je pense que vous avez deviné : c'est ici que nous allons écrire nos codes sources. C'est dans l'encadré situé en bas (en rouge sur la figure) que vous verrez apparaître le contenu de vos programmes… ainsi que les erreurs éventuelles ! Et pour finir, c'est dans l'encadré de droite (en violet sur la figure), 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 indiqué précédemment, 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 le 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.

En effet, la machine virtuelle Java se présente différemment selon que l'on se trouve sous Mac OS, Linux ou encore Windows. En revanche, le byte code reste quant à lui le même quel que soit l'environnement avec lequel a été développé et précompilé votre programme Java. Par conséquent, 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 répéter 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 codes, 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 porte 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 !

En revanche, 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.

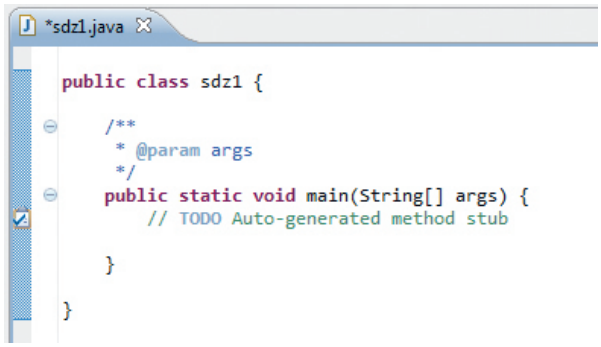
Une méthode est une suite d'instructions à exécuter. C'est un morceau de logique de notre programme. Une méthode contient :

- 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 les unes les autres. Mais pour le moment, nous n'allons travailler qu'avec une seule classe.

Je vous avais demandé précédemment de créer un projet Java, ouvrez-le. Vous voyez la fameuse classe dont je vous parlais ? Ici, elle s'appelle `sdz1` (figure suivante). 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 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`.



Méthode principale

La syntaxe de cette méthode est toujours la même :

```
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, il existe un moyen de forcer le compilateur à ignorer certaines instructions ! C'est ce qu'on appelle des **commentaires**, et deux syntaxes sont disponibles pour commenter son texte :

- les commentaires unilignes : ils sont introduits par les caractères `//` et mettent tout ce qui les suit en commentaire, du moment que le texte se trouve sur la même ligne ;
- les commentaires multilignes : ils sont introduits par les caractères `/*` et se terminent par `*/`.

```
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 content 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*). Nous aborderons ce sujet lorsque nous programmerons des objets mais pour les curieux, je vous conseille le très bon cours de dworkin sur ce sujet, disponible sur OpenClassrooms (*https://openclassrooms.com/courses/presentation-de-la-javadoc*).

À partir de maintenant, et jusqu'à ce que nous programmions des interfaces graphiques, nous allons faire ce que l'on appelle des « programmes procéduraux ». Cela signifie que le programme s'exécutera de façon procédurale, c'est-à-dire 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 est quant à elle basée sur des événements (clic de souris, choix dans un menu…).

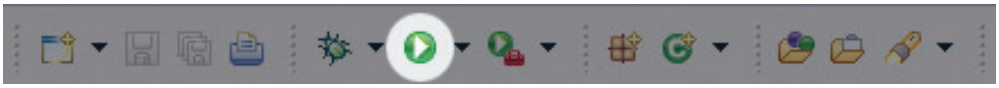
Hello World

Maintenant, essayons de taper le code suivant :

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

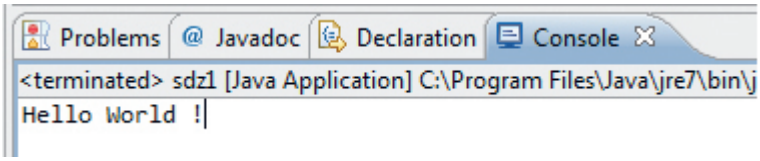
N'oubliez surtout pas le point-virgule (`;`) à la fin de la ligne ! Toutes les instructions en Java sont suivies d'un point-virgule.

Une fois que vous avez saisi cette ligne de code dans votre méthode `main`, il vous faut lancer le programme. Si vous vous souvenez de la présentation de la barre d'outils faite précédemment, vous devez cliquer sur l'icône représentant une flèche blanche dans un rond vert (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.



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` : il s'agit d'un objet de la classe System qui gère la sortie standard.
- `print` : cette méthode écrit dans la console le texte passé en paramètre (entre les parenthèses).

Prenons le code suivant :

```
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 :

```
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 ces solutions, nous obtenons :

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

Avec pour résultat :

```
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 indiquer deux autres caractères d'échappement :

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

Vous avez sûrement remarqué que la chaîne de caractères que l'on affiche est entourée par des guillemets doubles ". 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 le signe `\`, 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, mais elle ne peut être qu'enrichissante.

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 sources 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 OS 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 aux instructions suivantes :
  - `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.

2 Les variables et les opérateurs

Nous allons maintenant commencer sérieusement la programmation. Dans ce chapitre, nous allons découvrir les variables. On les retrouve dans la quasi-totalité des langages de programmation. Une variable est un élément qui stocke des informations de toute sorte en mémoire : des chiffres, des résultats de calcul, des tableaux, des renseignements fournis par l'utilisateur…

Vous ne pourrez pas programmer sans variables. Il est donc indispensable que je vous les présente !

Petit rappel

Avant de commencer, je vous propose un petit rappel sur le fonctionnement d'un ordinateur et particulièrement sur la façon dont ce dernier interprète notre façon de voir le monde…

Vous n'êtes pas sans savoir que votre ordinateur ne parle qu'une seule langue : le **binaire**. Le langage binaire est une simple suite de 0 et de 1. Il nous serait très difficile, en tant qu'êtres humains, d'écrire des programmes informatiques pour expliquer à nos ordinateurs ce qu'ils doivent faire, entièrement en binaire… Vous imaginez, des millions de 0 et de 1 qui se suivent ! Non, ce n'était pas possible ! De ce fait, des langages de programmation ont été créés afin que nous ayons à disposition des instructions claires pour créer nos programmes. Ces programmes sont ensuite compilés pour que nos instructions humainement compréhensibles soient, après coup, compréhensibles par votre machine.

Le langage binaire est donc une suite de 0 et de 1 que l'on appelle **bit**. Si vous êtes habitué à la manipulation de fichiers (audio, vidéo, etc.), vous devez savoir qu'il existe plusieurs catégories de poids de programme (Ko, Mo, Go, etc.). Tous ces poids correspondent au système métrique informatique. Le tableau suivant présente les poids les plus fréquemment rencontrés :

Tableau récapitulatif des poids des données

Abréviation	Traduction	Correspondance
b	Bit	C'est la plus petite valeur informatique, soit 0 soit 1.
o	Octet	Regroupement de 8 bits, par exemple : 01011101.
Ko	Kilo-octet	Regroupement de 1 024 octets.
Mo	Mégaoctet	Regroupement de 1 024 Ko.
Go	Gigaoctet	Regroupement de 1 024 Mo.
To	Teraoctet	Regroupement de 1 024 Go.

Dans ce cours, j'utilise les termes de kilo-octet, mégaoctet, gigaoctet et teraoctet, mais ce n'est pas exact. Il faudrait utiliser kibioctet (Kio), mébioctet (Mio), gibioctet (Gio) et tébioctet (Tio). Le « vrai » kilo-octet représente 1 000 octets tandis que le kibioctet contient 1 024 octets. Cependant, la plupart du temps, le terme kilo-octet sera utilisé pour désigner le kibioctet.

Pourquoi 1 024 octets et pas 1 000 ?

Si vous vous posez cette question, c'est parce que vous ne savez pas encore compter comme un ordinateur et que vous êtes trop habitué à utiliser un système en base 10. Je sais, c'est un peu confus… Pour comprendre pourquoi ce découpage est ainsi fait, vous devez prendre conscience que votre façon de compter n'est pas identique à celle de votre machine. En effet, vous avez l'habitude de compter ainsi :

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15… 254, 255, 256… 12 345 678, 12 345 679, 12 345 680

Cette façon de compter repose sur une base 10, car elle se décompose en utilisant des puissances de 10. Ainsi, le nombre 1 024 peut se décomposer de cette façon :

$1 \times 1\,000 + 0 \times 100 + 2 \times 10 + 4 \times 1$.

Pour bien comprendre ce qui suit, vous devez aussi savoir que tout nombre élevé à la puissance 0 vaut 1, donc $100 = 1$.

Partant de ce postulat, nous pouvons donc réécrire la décomposition du nombre 1 024 ainsi : $1 \times 103 + 0 \times 102 + 2 \times 101 + 4 \times 100$. Nous multiplions donc la base utilisée, ordonnée par puissance, par un nombre compris entre 0 et cette base moins 1 (de 0 à 9).

Sauf que votre machine parle en binaire, elle compte donc en base 2. Cela revient donc à appliquer la décomposition précédente en remplaçant les 10 par des 2. En revanche, vous n'aurez que deux multiplicateurs possibles : 0 ou 1 (et oui, vous êtes en base 2). De ce fait, en base 2, nous pourrions avoir ce genre de chose :

$1 \times 23 + 1 \times 22 + 0 \times 21 + 1 \times 20$, qui peut se traduire de la sorte :

$1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1$ donc $8 + 4 + 0 + 1$ soit 13.

Donc, 1101 en base 2 s'écrit 13 en base 10. Et donc pourquoi des paquets de 1 024 comme délimiteur de poids ? Car ce nombre correspond à une puissance de 2 : 1 024 = 210.

Dans le monde de l'informatique, il existe une autre façon de compter très répandue : l'hexadécimal. Dans ce cas, nous comptons en base 16 :

1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, 1C… 5E, 5F, 60… A53A, A53B, A53C…

C'est tordu ! À quoi ça peut bien servir ?

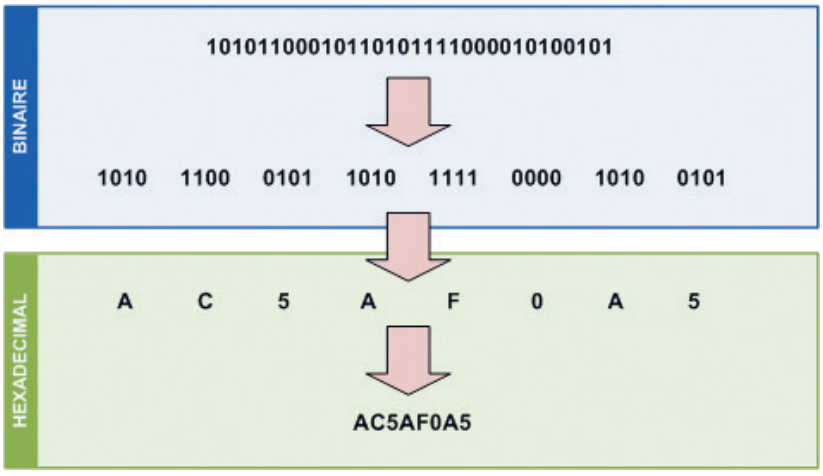
Le côté pratique de cette notation c'est qu'elle se base sur une subdivision d'un octet. Pour représenter un nombre de 0 à 15 (donc les seize premiers nombres), 4 bits sont nécessaires :

$0 \times 23 + 0 \times 22 + 0 \times 21 + 0 \times 20 = 0$ et $1 \times 23 + 1 \times 22 + 1 \times 21 + 1 \times 20 = 15$.

En fait, vous savez maintenant qu'un octet est un regroupement de 8 bits. Utiliser l'hexadécimal permet de simplifier la notation binaire car, si vous regroupez votre octet de bits en deux paquets de 4 bits, vous pouvez représenter chaque paquet avec un caractère hexadécimal. Voici un exemple :

```
10110100 -> 1011 0100
1011 (en base 2) = 11 (base 10) = B (en base 16)
0100 (en base 2) = 4 (base 10) = 4 (en base 16)
Donc 10110100 -> 1011 0100 -> B4
```

La figure représente un nombre binaire plus conséquent.



Un nombre binaire conséquent

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 ».

En Java, ce que l'on appelle des types simples ou types primitifs, correspond tout simplement aux nombres entiers, aux nombres réels, aux booléens ou encore aux caractères. Comme vous allez le voir, il y a plusieurs façons de déclarer certains de ces types.

Les variables de type numérique

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

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

Le type `short` (2 octets) contient les entiers compris entre -32 768 et +32 767.

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

Le type `int` (4 octets) va de -2×10^9 à 2×10^9 (2 suivi de 9… ce qui fait déjà un joli nombre).

```
int temperatureSoleil;
temperatureSoleil = 15600000; // La température est exprimée en kelvins
```

Le type `long` (8 octets) peut aller de $-9 \times 1\,018$ à $9 \times 1\,018$ (encore plus gros…).

```
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. Dans le cas contraire, 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;
```

Ou encore :

```
float nombre;
nombre = 2.0f;
```

Vous remarquerez que nous ne mettons pas une virgule, mais un point. Et vous noterez aussi que même si le nombre en question est rond, on écrit `.0` derrière celui-ci, 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 et qu'il n'a pas de suffixe.

```
double division;
division = 0.33333333333333333333333333333333333333333333334d;
```

Ici encore, vous devez utiliser une lettre, le `d`, pour parfaire la déclaration de votre variable.

Des variables stockant un caractère

Le type `char` contient *un* caractère stocké entre apostrophes (`' '`), comme ceci :

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

Des variables de type booléen

Le type `boolean`, quant à 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 :

```
// Première méthode de déclaration
String phrase;
phrase = "Titi et Grosminet";

// Deuxième méthode de déclaration
String str = new String();
str = "Une autre chaîne de caractères";

// Troisième méthode de déclaration
String string = "Une autre chaîne";

// Quatrième méthode de déclaration
String chaine = new String("Et une de plus !");
```

Attention, `String` commence par une majuscule ! Et lors de l'initialisation, on utilise des guillemets doubles (`" "`).

Comme mentionné plus haut, `String` n'est pas un type de variable, mais un objet. Notre variable est donc un objet, on parle aussi d'une instance : ici, une instance de la classe `String`. Nous y reviendrons lorsque nous aborderons les objets.

On te croit sur parole, mais pourquoi `String` commence par une majuscule et pas les autres ?

C'est simple, il s'agit d'une convention de nommage. En fait, c'est une façon d'appeler nos classes, nos variables, etc. Il faut que vous essayiez de la respecter au maximum. Voici les caractéristiques de cette convention :

- 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 espace ;
- tout ceci sans accentuation !

Je sais que la première classe que je vous ai demandé de créer ne respecte pas cette convention, mais il était trop tôt pour vous parler de cette convention… À présent, je vous demanderai de ne pas oublier ces règles.

Voici quelques exemples de noms de classes et de variables :

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

Pour en revenir au pourquoi du comment, je vous ai dit que les variables de type String sont des objets. Ceux-ci sont définis par une ossature (un squelette) qui est en fait une classe. Ici, nous utilisons un objet String défini par une classe qui s'appelle `String` également. C'est pourquoi String a une majuscule et pas int, float, etc., qui eux ne sont pas définis par une classe.

Veillez à bien respecter la casse (majuscules et minuscules) car une déclaration de CHAR à la place de char ou autre chose provoquera une erreur, tout comme une variable de type string à la place de String !

Faites donc bien attention lors de vos déclarations de variables… Une petite astuce quand même (enfin deux, plutôt) : on peut très bien compacter les phases de déclaration et d'initialisation en une seule phase, comme ceci :

```
int entier = 32;
float pi = 3.1416f;
char carac = 'z';
String mot = new String("Coucou");
```

Et lorsque nous avons plusieurs variables d'un même type, nous pouvons résumer tout ceci à une déclaration :

```
int nbre1 = 2, nbre2 = 3, nbre3 = 0;
```

Ici, toutes les variables sont des entiers et elles sont toutes initialisées.

Avant de nous lancer dans la programmation, nous allons faire un peu de mathématiques avec nos variables.

Les opérateurs arithmétiques

Les opérateurs arithmétiques sont ceux que l'on apprend à l'école primaire… ou presque :

- `+` : permet d'additionner deux variables numériques (mais aussi de concaténer des chaînes de caractères, nous y reviendrons) ;
- `-` : permet de soustraire deux variables numériques ;
- `*` : permet de multiplier deux variables numériques ;
- `/` : permet de diviser deux variables numériques ;
- `%` : permet de renvoyer le reste de la division entière de deux variables de type numérique ; cet opérateur s'appelle le « modulo ».

Voici quelques exemples de calculs :

```
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;             // ici, nbre3 vaut 0, car il n'y a
                           // pas de reste
```

Ici, nous voyons bien que nous pouvons affecter des résultats d'opérations sur des nombres à nos variables, mais aussi affecter des résultats d'opérations sur des variables de même type.

Je me doute bien que le modulo est assez difficile à assimiler. Voici une utilisation assez simple : pour vérifier qu'un entier est pair, il suffit de vérifier que son modulo 2 renvoie 0.

Maintenant, voici quelque chose que les personnes qui n'ont jamais programmé ont du mal à intégrer. La déclaration de variables est identique à celle de l'exemple précédent :

```
int nbre1, nbre2, nbre3;     // Déclaration des variables
nbre1 = nbre2 = nbre3 = 0;   // Initialisation

nbre1 = nbre1 + 1;          // nbre1 = lui-même, donc 0 + 1 => nbre1 = 1
nbre1 = nbre1 + 1;          // nbre1 = 1 (cf. exemple précédent),
                            // maintenant, nbre1 = 1 + 1 = 2
nbre2 = nbre1;              // nbre2 = nbre1 = 2
nbre2 = nbre2 * 2;          // nbre2 = 2 => nbre2 = 2 * 2 = 4
nbre3 = nbre2;              // nbre3 = nbre2 = 4
nbre3 = nbre3 / nbre3;      // nbre3 = 4 / 4 = 1
nbre1 = nbre3;              // nbre1 = nbre3 = 1
nbre1 = nbre1 - 1;          // nbre1 = 1 - 1 = 0
```

Et là aussi, il existe une syntaxe qui raccourcit l'écriture de ce genre d'opération :

```
nbre1 = nbre1 + 1;
nbre1 += 1;
nbre1++;
++nbre1;
```

Les trois premières syntaxes correspondent exactement à la même opération. La troisième sera certainement celle que vous utiliserez le plus, mais elle ne fonctionne que pour augmenter d'une unité la valeur de `nbre1`. Si vous voulez augmenter de 2 la valeur d'une variable, utilisez les deux syntaxes précédentes. On appelle cela l'**incrémentation**. La dernière fait la même chose que la troisième, mais il y a une subtilité dont nous reparlerons au chapitre 5 consacré aux boucles.

Pour la soustraction, la syntaxe est identique :

```
nbre1 = nbre1 - 1;
nbre1 -= 1;
nbre1--;
--nbre1;
```

Même commentaire que pour l'addition, sauf qu'ici, la troisième syntaxe s'appelle la **décrémentation**.

Les raccourcis pour la multiplication fonctionnent de la même manière, regardez plutôt :

```
nbre1 = nbre1 * 2;
nbre1 *= 2;
nbre1 = nbre1 / 2;
nbre1 /= 2;
```

Pour aller plus loin

Point très important : on ne peut faire du traitement arithmétique que sur des variables de même type sous peine de perdre en précision lors du calcul. Ainsi, on ne s'amusera 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.

Voici les raisons de ma mise en garde : comme je vous l'ai dit précédemment, chaque type de variable a une capacité différente et, pour faire simple, nous allons comparer nos variables à différents récipients. Ainsi, une variable de type :

- `byte` correspondrait à un dé à coudre, elle ne peut pas contenir grand-chose ;
- `int` serait un verre, c'est déjà plus grand ;
- `double` serait un baril, donc qui peut contenir beaucoup.

À partir de là, ce n'est plus qu'une question de bon sens. Il est possible de mettre le contenu d'un dé à coudre dans un verre ou un baril. En revanche, si vous versez le contenu d'un baril dans un verre… il y en aura partout à côté !

Ainsi, si nous affectons le résultat d'une opération sur deux variables de type `double` dans une variable de type `int`, le résultat sera de type `int` et ne sera donc pas un réel mais un entier.

Pour afficher le contenu d'une variable dans la console, appelez l'instruction `System.out.println(maVariable);` ou encore `System.out.print(maVariable);`.

Je suppose que vous voudriez aussi mettre du texte en même temps que vos variables… Eh bien sachez que l'opérateur + sert aussi d'opérateur de concaténation, c'est-à-dire qu'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);
```

Sachez aussi que vous pouvez tout à fait mettre des opérations dans un affichage, comme ceci : `System.out.print("Résultat = " + nbre1/nbre2);` (l'opérateur + joue ici le rôle d'opérateur de concaténation). Ceci vous permet d'économiser une variable et par conséquent de la mémoire.

Cependant, nous n'allons pas utiliser cette méthode dans ce chapitre. Vous constaterez que le résultat affiché est 3 au lieu de 3.33333333333333… Cela vous intrigue sûrement :

```
int resultat = (int)(nbre1 / nbre2);
```

Avant de vous expliquer pourquoi, remplacez la ligne précédente par :

```
int resultat = nbre1 / nbre2;
```

Vous allez voir qu'Eclipse n'aime pas du tout ! Pour comprendre cela, nous allons parler à présent des **conversions**.

Les conversions ou « cast »

Comme expliqué précédemment, les variables de type `double` contiennent plus d'informations que les variables de type `int`. Ici, il va falloir écouter attentivement… enfin plutôt lire ! Nous allons aborder un point très important en Java. Ne vous en déplaise, vous serez amené à convertir des variables :

• d'un type `int` en type `float` :

```
int i = 123;
float j = (float)i;
```

• d'un type `int` en `double` :

```
int i = 123;
double j = (double)i;
```

• 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 le `int` en question ne prendra que la valeur entière du `double`, quelle que soit celle des décimales.

Pour en revenir à notre problème précédent, il est aussi possible de caster le résultat d'une opération mathématique en la mettant entre parenthèses et en la précédant du type de cast souhaité, comme ceci :

```
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 nous faisions l'inverse, c'est-à-dire si nous déclarions deux entiers et que nous mettions 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 obtiendrez `1` pour résultat. Je ne caste pas ici, car un `double` peut contenir un `int`.

Voici une autre déclaration :

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

Afin de comprendre pourquoi, vous devez savoir qu'en Java, comme dans d'autres langages d'ailleurs, il existe la notion de **priorité d'opération**. Nous en avons ici un très bon exemple !

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, en respectant 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`.

Vous vous demandez sûrement pourquoi vous n'obtenez 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
```

Je ne vais pas trop détailler ce qui suit (vous verrez cela plus en détail dans le chapitre 9). Pour l'heure, vous allez apprendre à transformer l'argument d'un type donné, par exemple `int`, 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`.

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

Depuis Java 7 : le formatage des nombres

Comme vous le savez sûrement, le langage Java est en perpétuelle évolution. Les concepteurs ne cessent d'ajouter de nouvelles fonctionnalités qui simplifient la vie des développeurs. Ainsi, dans la version 7 de Java, vous avez la possibilité de formater vos variables de types numériques avec un séparateur, l'underscore (_), ce qui peut s'avérer très pratique pour de grands nombres qui peuvent être difficiles à lire. Voici quelques exemples :

```
double nombre = 1000000000000d; // cast en d
// Peut s'écrire ainsi
double nombre = 1____000____000___000_000d; // cast en d
// Le nombre d'underscore n'a pas d'importance

// Voici quelques autres exemples d'utilisation
int entier = 32_000;
double monDouble = 12_34_56_78_89_10d; // cast en d
double monDouble2 = 1234_5678_8910d;    // cast en d
```

Les underscores doivent être placés entre deux caractères numériques : ils ne peuvent donc pas être utilisés en début ou en fin de déclaration, ni avant ou après un séparateur de décimal. Ainsi, les déclarations suivantes ne sont pas valides :

```
double d = 123_.159;
int entier = _123;
int entier2 = 123_;
```

Avant Java 7, il était possible de déclarer des expressions numériques en hexadécimal, en utilisant le préfixe `0x` :

```
int entier = 255;  // Peut s'écrire « int entier = 0xFF; »
int entier = 20;   // Peut s'écrire « int entier = 0x14; »
int entier = 5112; // Peut s'écrire « int entier = 0x13_F8; »
```

Depuis Java 7, vous avez aussi la possibilité d'utiliser la notation binaire, en utilisant le préfixe `0b` :

```
int entier = 0b1111_1111;
// Est équivalent à : « int entier = 255; »
int entier = 0b1000_0000_0000;
// Est équivalent à : « int entier = 2048; »
int entier = 0b100000000000;
// Est équivalent à : « int entier = 2048; »
```

Certains programmes Java travaillent directement sur les bits, il peut donc être plus pratique de les représenter ainsi avant de les manipuler.

En résumé

- Les variables sont essentielles dans la construction de programmes informatiques.
- On affecte une valeur dans une variable avec l'opérateur égal (=).
- Après avoir affecté une valeur à une variable, l'instruction doit se terminer par un point-virgule (`;`).
- Vos noms de variables ne doivent contenir ni caractères accentués ni espaces et doivent, dans la mesure du possible, respecter la convention de nommage Java.
- Lorsque vous effectuez des opérations sur des variables, prenez garde à leur type : vous pourriez perdre en précision.
- Vous pouvez caster un résultat en ajoutant un type devant celui-ci : `(int)`, `(double)`, etc.
- Prenez garde aux priorités lorsque vous castez le résultat d'opérations, faute de quoi ce dernier risque d'être incorrect.

3 Lire les entrées clavier

Dans ce chapitre, vous allez apprendre à saisir des informations et à les stocker dans des variables afin de pouvoir les utiliser a posteriori.

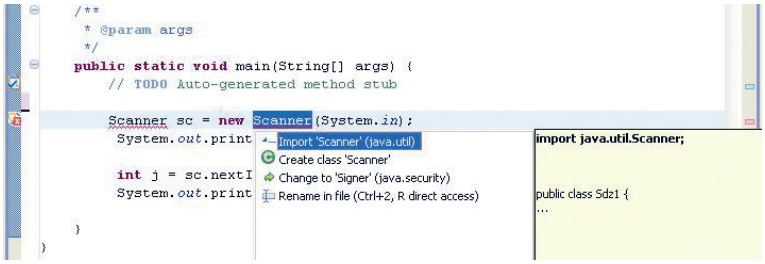
Jusqu'à ce que nous abordions les interfaces graphiques, nous travaillerons en mode console. Afin de rendre nos programmes plus ludiques, il est de bon ton de pouvoir interagir avec ceux-ci. Mais cela peut engendrer des erreurs (on parlera d'exceptions, lesquelles seront traitées plus loin). Afin de ne pas surcharger le chapitre, nous survolerons ce point sans s'attarder sur les différents cas d'erreurs que cela peut soulever.

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 présenter comporte des failles. Vous devrez taper des valeurs à récupérer dans votre programme !

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 nous n'en utiliserons qu'un ici, à savoir celui qui correspond à l'entrée standard en Java. Lorsque vous faites `System.out.println();`,vous appliquez la méthode `println()` sur la sortie standard. Ici, nous allons utiliser l'entrée standard `System.in`. Ainsi, avant d'indiquer à Java qu'il faut lire ce que nous allons taper au clavier, nous devrons instancier un objet `Scanner`. Avant de voir cela en détail, créez une nouvelle classe et tapez cette ligne de code dans votre méthode `main` :

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



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 cela, 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 manuellement 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 :

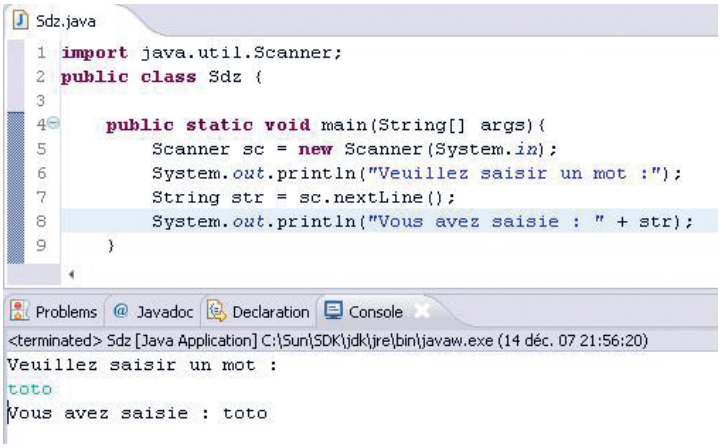
```
// Ceci importe la classe Scanner du package java.util
import java.util.Scanner;
// Ceci importe toutes les classes du package java.util
import java.util.*;
```

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.
```

Il existe un type de variable primitive 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 vous demandez peut-être pourquoi `charAt(0)` et pas `charAt(1)` ? Nous aborderons ce point lorsque nous verrons les tableaux… Jusqu'à ce que l'on traite des exceptions, je vous demanderai d'être rigoureux et de faire attention à ce que vous attendez comme type de donnée 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. En revanche, si vous avez invoqué une méthode comme `nextInt()` ou `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, le code suivant :

```
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 ! ");
  }
}
```

En résumé

- La lecture des entrées clavier se fait via l'objet `Scanner`.
- L'objet `Scanner` se trouve dans le package `java.util` que vous devrez importer.
- Pour pouvoir récupérer ce que vous allez taper dans la console, vous devrez initialiser l'objet `Scanner` avec l'entrée standard, `System.in`.
- Il y a une méthode de récupération de données pour chaque type (sauf les `char`) : `nextLine()` pour les `String`, `nextInt()` pour les `int`, etc.

4 Les conditions

Nous abordons ici l'un des chapitres les plus importants : les conditions, qui sont une autre notion fondamentale de la programmation. En effet, ce qui va être développé ici s'applique à énormément de langages de programmation, et pas seulement à Java.

Dans une classe, la lecture et l'exécution se font de façon séquentielle, c'est-à-dire ligne par ligne. Avec les conditions, nous allons pouvoir gérer différents cas de figure sans pour autant lire tout le code. Vous vous rendrez vite compte que tous vos projets ne sont que des enchaînements et des imbrications de conditions et de boucles (notion que l'on abordera au chapitre suivant).

Assez de belles paroles, entrons tout de suite dans le vif du sujet.

La structure if… else

Avant de pouvoir créer et évaluer des conditions, vous devez savoir que pour y parvenir, nous allons utiliser ce que l'on appelle des « opérateurs logiques ». Ceux-ci sont surtout utilisés lors de conditions (*si [test] alors [faire ceci]*) pour évaluer différents cas possibles. Voici les différents opérateurs à connaître :

- `==` : permet de tester l'égalité ;
- `!=` : permet de tester l'inégalité ;
- `<` : strictement inférieur à ;
- `<=` : inférieur ou égal à ;
- `>` : strictement supérieur à ;
- `>=` : supérieur ou égal à ;
- `&&` : l'opérateur ET, il permet de préciser une condition ;
- `||` : l'opérateur OU, qui permet également de préciser une condition ;
- `?:` : l'opérateur ternaire, que vous comprendrez mieux avec un exemple.

Comme je vous l'ai dit dans le chapitre précédent, les opérations en Java sont soumises à des priorités. Tous ces opérateurs logiques se plient également à cette règle, de la même manière que les opérateurs arithmétiques…

Imaginons un programme qui demande à un utilisateur d'entrer un nombre entier relatif (qui peut être soit négatif, soit nul, soit positif). Les structures conditionnelles vont nous permettre de gérer ces trois cas de figure. La structure de ces conditions ressemble au code suivant :

```
if(// Condition)
{
    // Exécution des instructions si la condition est remplie
}
else
{
    // Exécution des instructions si la condition n'est pas remplie
}
```

Cela peut se traduire par « si… sinon… ».

Le résultat de l'expression évaluée par l'instruction `if` sera un `boolean`, donc soit `true`, soit `false`. La portion de code du bloc `if` ne sera exécutée que si la condition est remplie. Dans le cas contraire, c'est le bloc de l'instruction `else` qui le sera. Mettons notre petit exemple en pratique :

```
int i = 10;

if (i < 0)
   System.out.println("le nombre est négatif");
else
   System.out.println("le nombre est positif");
```

Essayez ce petit code et observez comment il fonctionne. Dans cet exemple, notre classe affiche « le nombre est positif ». Expliquons un peu ce qu'il se passe.

- Dans un premier temps, la condition du `if` est testée : elle dit « si `i` est strictement inférieur à 0, alors fais ça ».
- Dans un second temps, vu que la condition précédente est fausse, le programme exécute le `else`.

Attends un peu, lorsque tu nous as présenté la structure des conditions, tu as mis des accolades et là, tu n'en mets pas. Pourquoi ?

Bien observé. En fait, les accolades sont présentes dans la structure « normale » des conditions, mais lorsque le code à l'intérieur de l'une d'entre elles n'est composé que d'une seule ligne, les accolades deviennent facultatives.

Comme nous avons l'esprit perfectionniste, nous voulons que notre programme affiche « le nombre est nul » lorsque `i` est égal à 0. Nous allons donc ajouter une condition. Comment faire ? La condition du `if` est remplie si le nombre est strictement négatif, ce qui n'est pas le cas ici puisque nous allons le mettre à 0. Le code contenu dans la clause `else` est donc exécuté si le nombre est égal ou strictement supérieur à 0. Il nous suffit d'ajouter une condition à l'intérieur de la clause `else`, comme ceci :

```
int i = 0;
if (i < 0)
{
   System.out.println("Ce nombre est négatif !");
}
else
{
   if(i == 0)
      System.out.println("Ce nombre est nul !");

   else
      System.out.println("Ce nombre est positif !");

}
```

Maintenant que vous avez tout compris, je vais vous présenter une autre façon d'écrire ce code et qui permet d'obtenir le même résultat : on ajoute juste un petit « sinon si… ».

```
int i = 0;
if (i < 0)
   System.out.println("Ce nombre est négatif !");

else if(i > 0)
   System.out.println("Ce nombre est positif !");

else
   System.out.println("Ce nombre est nul !");
```

Alors ? Explicite, n'est-ce pas ?

- si `i` est strictement négatif alors le code de *cette* condition est exécuté ;
- sinon, si `i` est strictement positif alors le code de *cette* condition est exécuté ;
- sinon, si `i` est forcément nul alors le code de *cette* condition est exécuté.

Il faut absolument donner une condition au `else if` pour qu'il fonctionne.

Je vais très fortement insister sur un point : observez l'affichage du code et remarquez le petit décalage entre le test et le code à exécuter. On appelle cela « l'indentation » !

Pour vous repérer dans vos futurs programmes, l'indentation sera très utile. Imaginez un instant que vous avez un programme de 700 lignes avec 150 conditions, et que tout est écrit sans indentation. Il sera difficile de distinguer les tests du code. Vous n'êtes pas obligé de le faire, mais je vous assure que vous y viendrez un jour ou l'autre.

Avant de passer à la suite, vous devez savoir qu'on ne peut pas tester l'égalité de chaînes de caractères ! Du moins, pas comme je vous l'ai montré précédemment. Nous aborderons ce point plus tard.

Les conditions multiples

Derrière ce nom barbare se cachent simplement plusieurs tests dans une instruction `if` (ou `else if`). Nous allons maintenant utiliser les opérateurs logiques en vérifiant si un nombre donné appartient à un intervalle connu. Par exemple, on va vérifier si un entier est compris entre 50 et 100.

```
int i = 58;
if(i < 100 && i > 50)
   System.out.println("Le nombre est bien dans l'intervalle.");
else
   System.out.println("Le nombre n'est pas dans l'intervalle.");
```

Nous avons utilisé l'opérateur `&&`. La condition de notre `if` est devenue : « si `i` est inférieur à 100 `ET` supérieur à 50 ».

Avec l'opérateur `&&`, la clause est remplie *si et seulement si* les conditions la constituant sont toutes remplies. Si l'une des conditions n'est pas vérifiée, la clause sera considérée comme fausse.

Cet opérateur vous initie à la notion d'intersection d'ensembles. Ici, nous avons deux conditions qui définissent un ensemble chacune :

- `i < 100` définit l'ensemble des nombres inférieurs à 100 ;
- `i > 50` définit l'ensemble des nombres supérieurs à 50.

L'opérateur `&&` permet de faire l'intersection de ces ensembles. La condition regroupe donc les nombres qui appartiennent à ces deux ensembles, c'est-à-dire les nombres de 51 à 99 inclus. Réfléchissez bien à l'intervalle que vous voulez définir. Observez ce code :

```
int i = 58;
if(i < 100 && i > 100)
   System.out.println("Le nombre est bien dans l'intervalle.");
else
   System.out.println("Le nombre n'est pas dans l'intervalle.");
```

Ici, la condition ne sera jamais remplie, car je ne connais aucun nombre qui soit à la fois plus petit et plus grand que 100 ! Reprenez le code précédent et remplacez l'opérateur `&&` par l'opérateur `||` (le OU donc). À l'exécution du programme, et après plusieurs tests de valeurs pour `i`, vous pourrez vous apercevoir que tous les nombres remplissent cette condition, sauf 100.

La structure switch

Le `switch` est surtout utilisé lorsque nous voulons des conditions « à la carte ». Prenons l'exemple d'une interrogation comportant deux questions : pour chacune d'elles, on peut obtenir uniquement 0 ou 10 points, ce qui nous donne au final trois notes et donc trois appréciations possibles, comme ceci :

- 0/20 : tu peux revoir ce chapitre, petit Zéro !
- 10/20 : je crois que tu as compris l'essentiel ! Viens relire ce chapitre à l'occasion.
- 20/20 : bravo !

Dans ce genre de cas, on utilise un `switch` pour éviter des `else if` à répétition et pour alléger un peu le code. Je vais vous montrer comment se construit une instruction `switch`, puis nous allons l'utiliser.

Syntaxe

```
switch (/* Variable */)
{
  case /* Argument */:
    /* Action */;
    break;
  default:
    /* Action */;
}
```

Voici les opérations effectuées par cette expression :

- La classe évalue l'expression figurant après le `switch` (ici, `/* Variable */`).
- Si le premier cas (`case /* Valeur possible de la variable */:`) correspond à la valeur de `/* Variable */`, l'instruction figurant dans celle-ci sera exécutée.
- Sinon, on passe au cas suivant, et ainsi de suite.
- Si aucun des cas ne correspond, la classe va exécuter ce qui se trouve dans l'instruction `default:/* Action */;`. Voyez ceci comme une sécurité.

Notez bien la présence de l'instruction `break;`. Elle permet de sortir du `switch` si un cas correspond. Pour mieux juger de l'utilité de cette instruction, enlevez tous les `break;` et compilez votre programme. Vous verrez le résultat…

Voici un exemple de `switch` que vous pouvez essayer :

```
int note = 10; // On imagine que la note maximale est 20

switch (note)
{
   case 0:
      System.out.println("Ouch !");
      break;
   case 10:
      System.out.println("Vous avez juste la moyenne.");
      break;
   case 20:
      System.out.println("Parfait !");
      break;
   default:
      System.out.println("Il faut davantage travailler.");
}
```

Je n'ai écrit qu'une ligne de code par instruction `case`, mais rien ne vous empêche d'en mettre plusieurs.

Si vous avez essayé ce programme en supprimant l'instruction `break;`, vous avez dû vous rendre compte que le `switch` exécute le code contenu dans le `case 10:`, mais aussi dans tous ceux qui suivent ! L'instruction `break;` permet de sortir de l'opération en cours. Dans notre cas, on sort de l'instruction `switch`, mais nous verrons une autre utilité à `break;` dans le chapitre suivant.

Depuis la version 7 de Java, l'instruction `switch` accepte les objets de type `String` en paramètre. De ce fait, l'instruction suivante est donc valide :

```
String chaine = "Bonjour";

switch(chaine) {
   case "Bonjour":
      System.out.println("Bonjour monsieur !");
      break;
   case "Bonsoir":
      System.out.println("Bonsoir monsieur !");
      break;
   default:
      System.out.println("Bonjoir ! ");
}
```

La condition ternaire

Les conditions ternaires sont assez complexes et relativement peu utilisées. Je vous les présente ici à titre indicatif. La particularité de ces conditions réside dans le fait que

trois opérandes (c'est-à-dire des variables ou des constantes) sont mises en jeu, mais aussi que ces conditions sont employées pour affecter des données à une variable. Voici à quoi ressemble la structure de ce type de condition :

```
int x = 10, y = 20;
int max = (x < y) ? y : x ; // Maintenant, max vaut 20
```

Décortiquons ce qu'il se passe :

- Nous cherchons à affecter une valeur à notre variable `max`, mais de l'autre côté de l'opérateur d'affectation se trouve une condition ternaire…
- Ce qui se trouve entre les parenthèses est évalué : `x` est-il plus petit que `y` ? Ainsi, deux cas de figure se profilent :
  - si la condition renvoie `true` (vrai), qu'elle est vérifiée, la valeur qui se trouve après le `?` sera affectée ;
  - sinon, la valeur se trouvant après le caractère `:` sera affectée.
- L'affectation est effective : vous pouvez utiliser votre variable `max`.

Vous pouvez également faire des calculs, par exemple, avant d'affecter les valeurs :

```
int x = 10, y = 20;
int max = (x < y) ? y * 2 : x * 2 ; // Ici, max vaut 2 * 20 donc 40
```

N'oubliez pas que la valeur que vous allez affecter à votre variable doit être du même type que votre variable. Sachez aussi que rien ne vous empêche d'insérer une condition ternaire dans une autre condition ternaire :

```
int x = 10, y = 20;

int max = (x < y) ? (y < 10) ? y % 10 : y * 2 : x ; // Max vaut 40

// Pas très facile à lire…
// Vous pouvez entourer votre deuxième instruction ternaire
// par des parenthèses pour mieux voir

max = (x < y) ? ((y < 10) ? y % 10 : y * 2) : x ; // Max vaut 40
```

En résumé

- Les conditions vous permettent de n'exécuter que certains morceaux de code.
- Il existe plusieurs sortes de structures conditionnelles :
  - la structure `if... elseif... else` ;
  - la structure `switch... case... default` ;
  - la structure `?:`.

- Si un bloc d'instructions contient plus d'une ligne, vous devez l'entourer d'accolades afin de bien en délimiter le début et la fin.
- Pour pouvoir mettre une condition en place, vous devez comparer des variables à l'aide d'opérateurs logiques.
- Vous pouvez mettre autant de comparaisons renvoyant un `boolean` que vous le souhaitez dans une condition.
- Pour la structure `switch`, pensez à mettre les instructions `break;` si vous ne souhaitez exécuter qu'un seul bloc `case`.

5 Les boucles

Le rôle des boucles est de répéter un certain nombre de fois les mêmes opérations. Tous les programmes, ou presque, ont besoin de ce type de fonctionnalité. Nous utiliserons les boucles pour permettre à un programme de recommencer depuis le début, pour attendre une action précise de l'utilisateur, parcourir une série de données, etc.

Une boucle s'exécute tant qu'une condition est remplie. Nous réutiliserons donc des notions du chapitre précédent.

La boucle while

Pour décortiquer précisément ce qui se passe dans une boucle, nous allons voir comment elle se construit. Une boucle commence par une déclaration, ici `while`, qui signifie, à peu de chose près, « tant que ». Nous avons ensuite une condition : c'est elle qui permet à la boucle de s'arrêter. Une boucle n'est utile que lorsque nous pouvons la contrôler, et donc lui faire répéter une instruction un certain nombre de fois. C'est à ça que servent les conditions. Viennent ensuite une ou plusieurs instructions : c'est ce que va répéter notre boucle (il peut même y avoir des boucles dans une boucle !).

```
while (/* Condition */)
{
   // Instructions à répéter
}
```

Nous allons travailler sur un exemple concret mais d'abord, réfléchissons à comment notre boucle va travailler. Pour cela, il faut préciser notre exemple. Nous allons afficher « Bonjour, <un prénom> », prénom qu'il faudra taper au clavier. Nous demanderons ensuite si l'on veut recommencer. Pour cela, il nous faut une variable qui va recevoir le

prénom, dont le type sera donc `String`, ainsi qu'une variable pour récupérer la réponse. Plusieurs choix s'offrent alors à nous : soit un caractère, soit une chaîne de caractères, soit un entier. Ici, nous opterons pour une variable de type `char`. C'est parti !

```
// Une variable vide
String prenom;
   // On initialise celle-ci à O pour oui
char reponse = 'O';
   // Notre objet Scanner, n'oubliez pas l'import
   // de java.util.Scanner !
Scanner sc = new Scanner(System.in);
   // Tant que la réponse donnée est égale à oui…
while (reponse == 'O')
{
   // On affiche une instruction
   System.out.println("Donnez un prénom : ");
   // On récupère le prénom saisi
   prenom = sc.nextLine();
   // On affiche notre phrase avec le prénom
   System.out.println("Bonjour " +prenom+ ", comment vas-tu ?");
   // On demande si la personne veut faire un autre essai
   System.out.println("Voulez-vous réessayer ? (O/N)");
   // On récupère la réponse de l'utilisateur
   reponse = sc.nextLine().charAt(0);
}

System.out.println("Au revoir…");
// Fin de la boucle
```

Vous avez dû cligner des yeux en lisant `reponse = sc.nextLine().charAt(0);`. Rappelez-vous comment on récupère un `char` avec l'objet `Scanner` : nous devons récupérer un objet `String` et ensuite prendre le premier caractère de celui-ci. Cette syntaxe vient contredire ce que je vous avais exposé auparavant.

Détaillons un peu ce qu'il se passe. Dans un premier temps, nous avons déclaré et initialisé nos variables. Ensuite, la boucle évalue la condition qui nous dit : « tant que la variable `reponse` contient `O`, on exécute la boucle ». Celle-ci contient bien le caractère `O`, donc nous entrons dans la boucle. Puis, les instructions sont exécutées suivant l'ordre dans lequel elles apparaissent dans la boucle. À la fin, c'est-à-dire à l'accolade fermante de la boucle, le compilateur nous ramène au début de la boucle.

Cette boucle n'est exécutée que lorsque la condition est remplie. Ici, nous avons initialisé la variable `reponse` à `O` pour que la boucle s'exécute. Si nous ne l'avions pas fait, nous n'y serions jamais entrés. Normal, puisque nous testons la condition avant d'entrer dans la boucle !

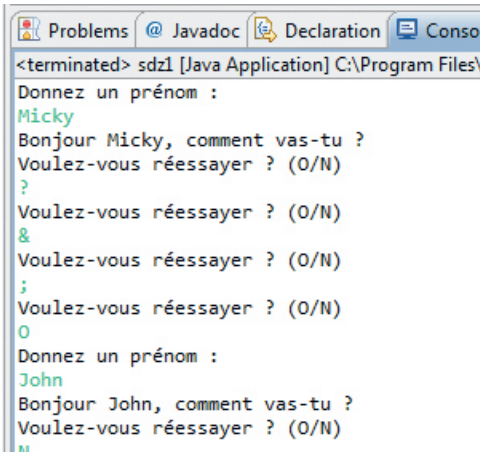
C'est pas mal, mais il faudrait forcer l'utilisateur à ne taper que « O » ou « N ». Comment faire ? C'est très simple, avec une boucle ! Il suffit de forcer l'utilisateur à entrer soit « N », soit « O » avec un `while`. Attention, il nous faudra réinitialiser la variable `reponse` à `''` (caractère vide). Il faudra donc répéter la phase « Voulez-vous réessayer ? » tant que la réponse donnée n'est pas « O » ou « N ».

Voici notre programme dans son intégralité :

```
String prenom;
char reponse = 'O';
Scanner sc = new Scanner(System.in);
while (reponse == 'O')
{
   System.out.println("Donnez un prénom : ");
   prenom = sc.nextLine();
   System.out.println("Bonjour " +prenom+ ", comment vas-tu ?");
   // Sans ça, nous n'entrerions pas dans la deuxième boucle
   reponse = ' ';

   // Tant que la réponse n'est pas O ou N, on repose la question
   while(reponse != 'O' && reponse != 'N')
   {
      // On demande si la personne veut faire un autre essai
      System.out.println("Voulez-vous réessayer ? (O/N)");
      reponse = sc.nextLine().charAt(0);
   }
}
System.out.println("Au revoir…");
```

Vous pouvez tester ce code (c'est d'ailleurs vivement conseillé). Vous verrez que si vous n'entrez pas la bonne lettre, le programme vous posera sans cesse sa question (figure suivante) !



Les instructions dans la boucle se répètent.

Attention à écrire correctement vos conditions et à bien vérifier vos variables dans vos `while`, et dans toutes vos boucles en général. Sinon, c'est le drame ! Essayez d'exécuter le programme précédent sans la réinitialisation de la variable `reponse`, et vous verrez le résultat ! On n'entre jamais dans la deuxième boucle, car `reponse = 'O'`

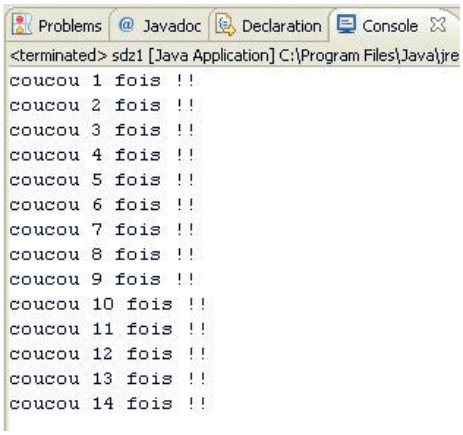
(puisque initialisée ainsi au début du programme). Là, vous ne pourrez jamais changer sa valeur… Le programme ne s'arrêtera donc jamais ! On appelle cela une « boucle infinie ». En voici un autre exemple.

```
int a = 1, b = 15;
while (a < b)
{
   System.out.println("coucou " +a+ " fois !!");
}
```

Si vous lancez ce programme, vous allez voir une quantité astronomique de « coucou 1 fois !! ». Nous aurions dû ajouter une instruction dans le bloc d'instructions de notre `while` pour changer la valeur de `a` à chaque tour de boucle, comme ceci :

```
int a = 1, b = 15;
while (a < b)
{
   System.out.println("coucou " +a+ " fois !!");
   a++;
}
```

La figure suivante présente le résultat de ce code.



Correction de la boucle infinie

Une petite astuce : lorsque vous n'avez qu'une instruction dans votre boucle, vous pouvez enlever les accolades, car elles deviennent superflues, tout comme pour les instructions `if`, `else if` ou `else`.

Vous auriez aussi pu utiliser cette syntaxe :

```
int a = 1, b = 15;
while (a++ < b)
    System.out.println("coucou " +a+ " fois !!");
```

Souvenez-vous de ce dont je vous parlais au chapitre précédent sur la priorité des opérateurs. Ici, l'opérateur < a la priorité sur l'opérateur d'incrémentation ++. Pour faire court, la boucle `while` teste la condition et ensuite incrémente la variable `a`. En revanche, essayez ce code :

```
int a = 1, b = 15;
while (++a < b)
   System.out.println("coucou " +a+ " fois !!");
```

Vous devez remarquer qu'il y a un tour de boucle en moins ! Eh bien avec cette syntaxe, l'opérateur d'incrémentation est prioritaire sur l'opérateur d'inégalité (ou d'égalité), c'est-à-dire que la boucle incrémente la variable `a`, et ce n'est qu'après l'avoir fait qu'elle teste la condition.

La boucle do… while

Puisque je viens de vous expliquer comment fonctionne une boucle `while`, je ne vais pas vraiment m'attarder sur la `boucle do… while`. En effet, ces deux boucles ne sont pas cousines, mais plutôt sœurs. Leur fonctionnement est identique à deux détails près.

```
do{
   // Instructions
}while(a < b);
```

- **Première différence**
  La boucle `do… while` s'exécutera *au moins une fois*, contrairement à sa sœur `while`. C'est-à-dire que la phase de test de la condition se fait à la fin, car la condition se met après le `while`.
- **Deuxième différence**
  Il s'agit d'une différence de syntaxe, qui se situe après la condition du `while`. Vous voyez la différence ? Oui ? Non ? Il y a un point-virgule `;` après le `while`. C'est tout ! Ne l'oubliez pas cependant, sinon le programme ne compilera pas.

Mis à part ces deux éléments, ces boucles fonctionnent exactement de la même manière. D'ailleurs, refaisons notre programme précédent avec une boucle `do… while`.

```
String prenom = new String();
// Pas besoin d'initialiser : on entre au moins une fois
// dans la boucle !
char reponse = ' ';

Scanner sc = new Scanner(System.in);

do{
   System.out.println("Donnez un prénom : ");
   prenom = sc.nextLine();
   System.out.println("Bonjour " +prenom+ ", comment vas-tu ?");

   do{
      System.out.println("Voulez-vous réessayer ? (O/N)");
      reponse = sc.nextLine().charAt(0);
   }while(reponse != 'O' && reponse != 'N');

}while (reponse == 'O');

System.out.println("Au revoir…");
```

Vous voyez donc que ce code ressemble beaucoup à celui utilisé avec la boucle `while`, mais il comporte une petite subtilité. Ici, plus besoin de réinitialiser la variable `reponse`, puisque de toute manière, la boucle s'exécutera au moins une fois !

La boucle for

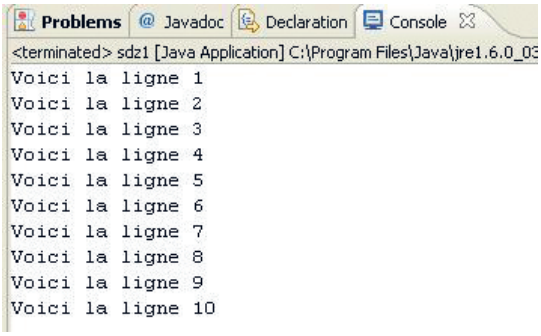
Cette boucle est un peu particulière puisqu'elle prend tous ses attributs dans sa condition et agit en conséquence. Je m'explique. Jusqu'ici, nous avions créé des boucles avec :

- la déclaration d'une variable avant la boucle ;
- l'initialisation de la variable ;
- l'incrémentation de la variable dans la boucle.

Toutes ces étapes seront rassemblées dans la condition de la boucle `for`. Il existe une autre syntaxe pour la boucle `for` depuis le JDK 1.5. Nous la verrons lorsque nous aborderons les tableaux au chapitre 7. Mais je sais bien qu'un long discours ne vaut pas un exemple, alors voici une boucle `for` sous vos yeux ébahis :

```
for(int i = 1; i <= 10; i++)
{
   System.out.println("Voici la ligne "+i);
}
```

La figure suivante présente le résultat obtenu.



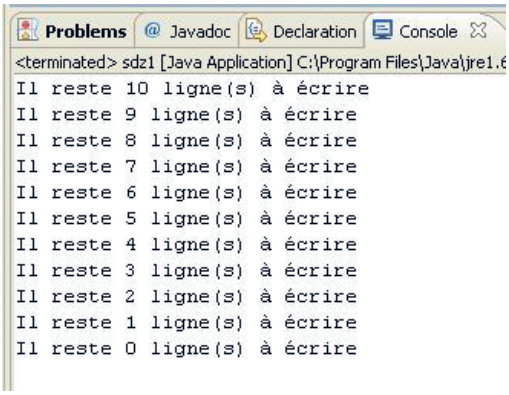
Test de boucle for

Vous aurez sûrement remarqué la présence des points-virgules `;` dans la condition pour la séparation des champs. Ne les oubliez surtout pas, sinon le programme ne compilera pas.

Nous pouvons aussi inverser le sens de la boucle, c'est-à-dire qu'au lieu de partir de 0 pour aller à 10, nous allons commencer à 10 pour atteindre 0 :

```
for(int i = 10; i >= 0; i--)
   System.out.println("Il reste "+i+" ligne(s) à écrire");
```

Le résultat de ce code est présenté à la figure suivante.



Boucle for avec décrémentation

Pour simplifier, la boucle `for` est un peu le condensé d'une boucle `while` dont le nombre de tours se détermine via un incrément. Nous avons un nombre de départ, une condition qui doit être remplie pour exécuter une nouvelle fois la boucle et une instruction de fin de boucle qui incrémente notre nombre de départ. Remarquez que

rien ne nous empêche de cumuler les déclarations, les conditions et les instructions de fin de boucle :

```
for(int i = 0, j = 2; (i < 10 && j < 6); i++, j+=2){
   System.out.println("i = " + i + ", j = " + j);
}
```

Ici, cette boucle n'effectuera que deux tours puisque la condition (i < 10 && j < 6) est remplie dès le deuxième tour, la variable `j` commençant à 2 et étant incrémentée de deux à chaque tour de boucle.

En résumé

- Les boucles vous permettent simplement d'effectuer des tâches répétitives.
- Il existe plusieurs sortes de boucles :
  - la boucle `while(condition){…}` évalue la condition puis exécute éventuellement un tour de boucle (ou plus) ;
  - la boucle `do{…}while(condition);` fonctionne exactement comme la précédente, mais effectue un tour de boucle quoi qu'il arrive ;
  - la boucle `for` permet d'initialiser un compteur, une condition et un incrément dans sa déclaration afin de répéter un morceau de code un nombre limité de fois.
- Tout comme les conditions, si une boucle contient plus d'une ligne de code à exécuter, vous devez l'entourer d'accolades afin de bien en délimiter le début et la fin.

6 TP : conversion Celsius–Fahrenheit

Voici un petit TP qui va vous permettre de mettre en œuvre toutes les notions que vous avez vues jusqu'ici :

- les variables ;
- les conditions ;
- les boucles ;
- votre génial cerveau…

Accrochez-vous, car je vais vous demander de penser à beaucoup de choses et vous serez tout seul. Lâché dans la nature… Mais non, je plaisante, je vais vous guider un peu.

Élaboration

Voici les caractéristiques du programme que nous allons devoir réaliser :

- le programme demande quelle conversion nous souhaitons effectuer, à savoir Celsius vers Fahrenheit ou l'inverse ;
- on n'autorise que les modes de conversion définis dans le programme (un simple contrôle sur la saisie fera l'affaire) ;
- on demande à la fin à l'utilisateur s'il veut faire une nouvelle conversion, ce qui signifie que l'on doit pouvoir revenir au début du programme.

Avant de vous lancer dans la programmation à proprement parler, je vous conseille fortement de réfléchir à votre code… sur papier. Réfléchissez à ce qu'il vous faut comme nombre de variables, les types de variables, comment va se dérouler le programme, les conditions et les boucles utilisées.

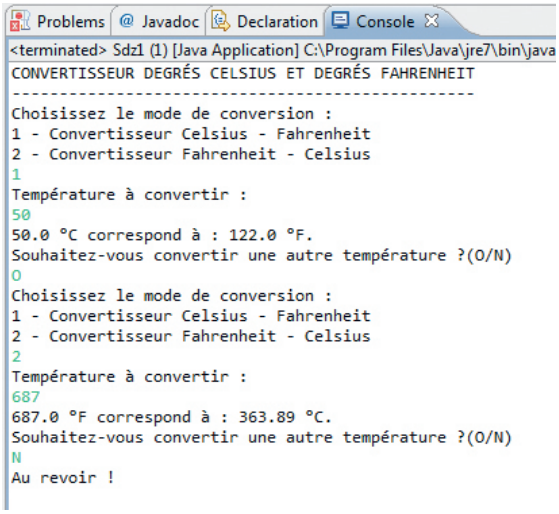
À toutes fins utiles, voici la formule de conversion pour passer des degrés Celsius en degrés Fahrenheit :

$$F = \frac{9}{5} \times C + 32$$

Pour l'opération inverse, la formule est la suivante :

$$C = \frac{(F - 32) \times 5}{9}$$

La figure suivante présente un aperçu de ce que je vous demande.

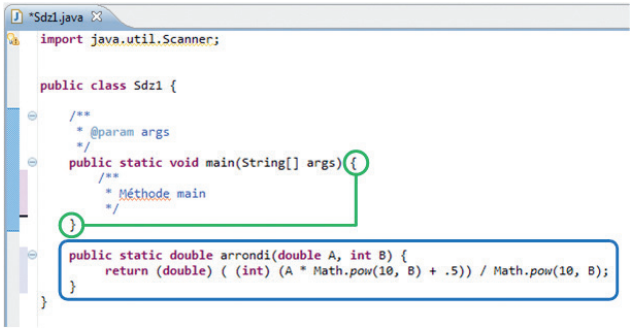


Rendu du TP

Je vais également vous donner une fonction toute faite qui vous permettra éventuellement d'arrondir vos résultats. Je vous expliquerai comment marchent les fonctions au chapitre 8. Vous pouvez très bien ne pas vous en servir. Pour ceux qui souhaitent tout de même utiliser cette fonction d'arrondi, la voici :

```
public static double arrondi(double A, int B) {
   return (double) ( (int) (A * Math.pow(10, B) + .5)) / Math.pow(10, B);
}
```

Elle est à placer entre les deux accolades de votre classe (figure suivante).



Emplacement de la fonction

Voici comment utiliser cette fonction : imaginez que vous avez la variable `faren` à arrondir, et que le résultat obtenu est enregistré dans une variable `arrondFaren`. Vous procéderez alors comme suit :

```
arrondFaren = arrondi(faren,1);
// Pour un chiffre après la virgule
arrondFaren = arrondi(faren, 2);
// Pour deux chiffres après la virgule, etc.
```

Une dernière recommandation : essayez de bien indenter votre code. Prenez votre temps. Essayez de penser à tous les cas de figure. Maintenant à vos papiers, crayons, neurones, claviers… et bon courage !

Correction

STOP ! C'est fini ! Il est temps de passer à la correction de ce premier TP. Pas trop mal à la tête ? Je me doute qu'il a dû y avoir quelques tubes d'aspirine vidés. Mais vous allez voir qu'en définitive, ce TP n'était pas si compliqué que ça.

Surtout, n'allez pas croire que ma correction est parole d'évangile. Il y avait différentes manières d'obtenir le même résultat. Voici l'une des solutions possibles :

```
import java.util.Scanner;

class Sdz1 {
  public static void main(String[] args) {
    // Notre objet Scanner
    Scanner sc = new Scanner(System.in);

    // Initialisation des variables
    double aConvertir, convertit=0;
    char reponse=' ', mode = ' ';
```

```
     System.out.println("CONVERTISSEUR DEGRÉS CELSIUS ET DEGRÉS
FAHRENHEIT");
     System.out.println("-----------------------------------------------");

     do{// tant que reponse = O // boucle principale

       do{// tant que reponse n'est pas O ou N
          mode = ' ';
          System.out.println("Choisissez le mode de conversion : ");
          System.out.println("1 - Convertisseur Celsius - Fahrenheit");
          System.out.println("2 - Convertisseur Fahrenheit - Celsius ");
          mode = sc.nextLine().charAt(0);

          if(mode != '1' && mode != '2')
            System.out.println("Mode inconnu, veuillez réitérer votre
choix.");

       }while (mode != '1' && mode != '2');

       // Saisie de la température à convertir
       System.out.println("Température à convertir :");
       aConvertir = sc.nextDouble();
       // Pensez à vider la ligne lue
       sc.nextLine();

       // Selon le mode, on calcule différemment et on affiche le résultat
       if(mode == '1'){
          convertit = ((9.0/5.0) * aConvertir) + 32.0;
          System.out.print(aConvertir + " °C correspond à : ");
          System.out.println(arrondi(convertit, 2) + " °F.");
       }
       else{
          convertit = ((aConvertir - 32) * 5) / 9;
          System.out.print(aConvertir + " °F correspond à : ");
          System.out.println(arrondi(convertit, 2) + " °C.");
       }

       // On invite l'utilisateur à recommencer ou à quitter
       do{
          System.out.println("Souhaitez-vous convertir une autre température
?(O/N)");
          reponse = sc.nextLine().charAt(0);

       }while(reponse != 'O' && reponse != 'N');

     }while(reponse == 'O');

   System.out.println("Au revoir !");

   // Fin du programme
   }

   public static double arrondi(double A, int B) {
     return (double) ( (int) (A * Math.pow(10, B) + .5)) / Math.pow(10,
B);
   }
}
```