

Traducteur : Jean-Pierre VIDAL

25.04.2001 - Version 0.2 :

- Mise en forme du code html (titres-hx[verdana], paragraphes-p[Georgia], code-blockquote

19.08.2000 - Version 0.1 :

- Dernière mise à jour de la version française

# Introduction

Tout comme n'importe quel langage humain, Java permet d'exprimer des concepts. S'il y parvient, il deviendra un moyen d'expression considérablement plus simple et plus souple que n'importe quelle alternative, alors même que les problèmes augmentent en taille et en complexité.

Il est impossible de considérer Java uniquement sous l'angle d'une collection de fonctionnalités — beaucoup de fonctionnalités perdent leur sens hors de leur contexte. On ne peut utiliser la somme des parties que si l'on pense en termes de *conception*, et non simplement en termes de code. Pour appréhender Java de cette manière, il faut comprendre les problèmes qui lui sont propres et ceux qui relèvent de la programmation en général. Ce livre traite de problèmes de programmation, en quoi ce sont des problèmes, et quelle est l'approche de Java pour les résoudre. Ainsi, l'ensemble de fonctionnalités que je présente dans chaque chapitre est basée sur ma manière de résoudre un certain type de problèmes au moyen de ce langage. Par cette démarche j'espère peu à peu amener le lecteur au point où « penser Java » lui deviendra naturel.

Je garderai constamment à l'esprit qu'il faut que chacun se construise un modèle de pensée permettant de développer une profonde connaissance du langage ; lorsqu'il rencontrera un problème ardu il devra être capable d'en alimenter ce modèle et d'en déduire la solution.

## Prérequis

Ce livre part du principe que le lecteur est un familier de la programmation : il sait qu'un programme est un ensemble d'instructions, il sait ce que sont un sous-programme, une fonction, une macro-instruction, un ordre de contrôle tel que « if » ainsi qu'une structure de boucle telle que « while », etc. Toutefois, il a certainement appris cela de différentes façons, par exemple en programmant avec un macro-langage ou bien en utilisant un outil tel que Perl. Si vous faites partie de ceux qui sont à l'aise avec les idées de base de la programmation, vous lirez ce livre sans problème. Bien entendu, le livre sera plus *facile* pour les programmeurs C et encore plus pour les programmeurs C++, mais n'abandonnez pas pour autant si vous n'avez aucune expérience de ces langages (en revanche, préparez-vous à travailler dur ; par ailleurs, le CD multimédia fourni avec ce livre vous amènera rapidement à comprendre la syntaxe de base du langage C nécessaire à l'apprentissage de Java). Je vais introduire les concepts de la programmation orientée objet (POO) et les mécanismes de contrôle de base de Java, ainsi le lecteur en aura connaissance, et rencontrera dans les premiers exercices les instructions de base du contrôle de flux de programme.

Bien qu'il soit souvent fait référence aux fonctionnalités des langages C et C++, il ne s'agit pas d'un aparté pour initiés, mais au contraire d'aider tous les programmeurs à mettre Java en perspective avec ces deux langages, qui, après tout, sont ses parents. Je vais essayer de simplifier ces références et d'expliquer à un programmeur ne connaissant ni C ni C++ tout ce que j'imagine être peu familier pour lui.

## Apprendre Java

J'ai commencé à enseigner C++ à l'époque où était édité mon premier livre *Using C++* (Osborne McGraw-Hill, 1989). Enseigner la programmation est devenu ma profession ; depuis 1989 j'ai vu bien des hochements de tête, de visages vides, ainsi que beaucoup d'expressions d'incompréhension chez maint auditoire à travers le monde. Lorsque je me mis à donner des cours chez moi, pour des groupes plus réduits, je découvris que même ceux qui souriaient et hochaient la tête étaient déconcertés face à de nombreux problèmes. J'ai découvert aussi, alors que je présidais le cursus C++ durant plusieurs années à la Software Development Conference (et plus tard le cursus Java), que moi-même ainsi que les autres conférenciers avions tendance à traiter trop de choses trop rapidement. Finalement, à cause des différences entre les niveaux de mes auditeurs tout autant que de la manière dont je présentais mon exposé, j'aurais fini par perdre une partie de mon auditoire. Je me suis posé beaucoup de questions, mais, faisant partie de ceux qui rechignent au cours magistral (et chez beaucoup de gens, je crois qu'une telle attitude ne peut provenir que du souvenir de l'ennui que distillent de tels cours), j'ai voulu faire en sorte que tout le monde reste éveillé.

À une certaine période, je terminais mes différents cours sous la pression des besoins. C'est ainsi que j'ai fini par enseigner par essais et erreurs (une technique qui marche bien également dans la conception des programmes Java), et finalement j'ai réalisé un cours qui utilise tout ce que j'ai appris grâce à mon expérience d'enseignant — un cours qu'il me serait agréable de donner durant longtemps. Il consiste à s'attaquer au problème de l'apprentissage par touches discrètes et faciles à intégrer, et lors d'un séminaire impromptu (la situation idéale pour enseigner) chaque courte leçon est suivie d'exercices. Je donne maintenant ce cours dans des séminaires Java publics, que l'on peut trouver sur le site <http://www.BruceEckel.com>. (Le séminaire d'introduction est également disponible sur le CD ROM, son contenu est disponible sur le même site Web.)

Le retour d'expérience que me procure chaque séminaire m'aide à modifier et recentrer mon discours jusqu'à ce qu'il devienne un bon moyen d'enseignement. Mais ce livre est plus qu'une simple compilation de notes de séminaires : j'ai tenté d'y intégrer autant d'informations que je le pouvais, et je l'ai structuré afin que chaque sujet mène droit au suivant. Enfin, plus que tout, le livre est conçu pour aider le lecteur solitaire qui se bat avec un nouveau langage de programmation.

## Buts

Comme mon précédent livre *Thinking in C++*, celui-ci a été structuré pour enseigner le langage. En particulier, ma motivation est de faire en sorte qu'il puisse me servir pour enseigner le langage dans mes propres séminaires. Lorsque je pense à un chapitre du livre, je me demande s'il pourrait faire l'objet d'une bonne leçon lors d'un séminaire. Mon but est d'avoir des chapitres courts pouvant être exposés en un temps raisonnable, suivis par des exercices réalisables dans une situation de salle de classe.

Dans ce livre je me suis donné comme buts :

1. présenter le cours pas à pas afin que le lecteur assimile chaque concept avant d'aller plus avant ;
2. utiliser des exemples qui soient aussi simples et courts que possible. De temps en temps, cela me détournera des problèmes « du monde réel », mais j'ai remarqué que les débutants sont généralement plus satisfaits de comprendre chaque détail d'un exemple qu'ils ne sont impressionnés par la portée du problème qu'ils cherchent à résoudre. Il y a également une limite à la taille du code qui peut être assimilé dans une situation de cours magistral, limite qu'il faut impérativement ne pas dépasser. A ce sujet je devrais recevoir quelques critiques pour avoir utilisé des « exemples jouets », et je les accepte volontiers, avec le prétexte que ce que je présente est utile, pédagogiquement parlant ;
3. enchaîner soigneusement la présentation des fonctionnalités afin que l'on ne rencontre jamais quoi que ce soit qui n'ait jamais été exposé. Bien entendu, ce n'est pas toujours possible, et, dans de telles situations,

je donne une brève description en introduction ;

4. montrer ce que je pense être important concernant la compréhension du langage, plutôt qu'exposer tout mon savoir. Je crois que l'information est fortement hiérarchisée, qu'il est avéré que 95 % des programmeurs n'ont pas besoin de tout connaître, et que cela dérouté tout le monde et ajoute à leur impression de complexité du langage. Pour prendre un exemple en C, connaissant par coeur le tableau de priorité des opérateurs (ce qui n'est pas mon cas), il est possible d'écrire un code astucieux. Mais en y réfléchissant un instant, ceci risque de dérouter le lecteur et/ou le mainteneur de ce code. Il est donc préférable d'oublier la priorité des opérateurs, et d'utiliser des parenthèses lorsque les choses ne sont pas claires ;
5. maintenir chaque section assez concentrée de telle manière que le temps de lecture - et le temps entre les exercices - soit court. Non seulement cela maintient l'attention et l'implication des auditeurs lors d'un séminaire, mais cela donne au lecteur une plus grande impression de travail bien fait ;
6. vous munir de bases solides afin que votre connaissance soit suffisante avant de suivre un cours ou lire un livre plus difficiles.

## Documentation en ligne

Le langage Java et les bibliothèques de Sun Microsystems (en téléchargement libre) sont accompagnés d'une documentation sous forme électronique, que l'on peut lire avec un navigateur Web, et en pratique chaque implémentation tierce de Java possède un système de documentation équivalent. La plupart des livres publiés à propos de Java dupliquent cette documentation. Soit vous l'avez déjà, soit vous pouvez la télécharger, et donc ce livre ne la reprendra pas, excepté lorsque c'est nécessaire, parce qu'il sera généralement plus rapide de trouver la description d'une classe au moyen d'un navigateur plutôt que dans le livre (de plus, la documentation en ligne sera probablement davantage à jour). Ce livre fournira certaines descriptions de classes supplémentaires lorsqu'il sera nécessaire de compléter la documentation afin de comprendre un exemple particulier.

## Les chapitres

Ce livre a été conçu en gardant une seule chose à l'esprit : la manière dont les gens apprennent le langage Java. Le retour d'information des auditeurs de séminaires m'a aidé à découvrir les parties difficiles qui justifient un autre éclairage. Dans les domaines où je fus ambitieux, où j'ai ajouté trop de fonctionnalités dans un même temps, j'ai fini par comprendre - au travers du processus d'enseignement - que si l'on ajoute de nouvelles fonctionnalités, on doit les expliquer complètement, et que cela peut dérouter les étudiants. Je me suis donc donné beaucoup de mal pour introduire aussi peu que possible de nouveaux concepts en un même temps.

Le but est donc d'enseigner une seule fonctionnalité par chapitre, ou à la rigueur un petit ensemble de fonctionnalités associées, en évitant les liaisons avec des fonctionnalités supplémentaires. De cette manière, il est possible d'assimiler chaque chose dans le contexte de la connaissance actuelle avant d'aller plus loin.

Voici une brève description des chapitres contenus dans le livre, qui correspondent aux leçons et exercices de mes séminaires.

### ***Chapitre 1 : Introduction sur les Objets***

Ce chapitre est une vue d'ensemble de ce qu'est la programmation orientée objet, y compris la réponse à la question de base « Qu'est-ce qu'un objet », ce que sont une interface et une implémentation, l'abstraction et l'encapsulation, les messages et les fonctions, l'héritage et la composition, ainsi que le polymorphisme qui est d'une très haute importance. On y trouve également une vue d'ensemble de la manière dont les objets sont créés

avec les constructeurs, où se trouvent les objets, où les ranger une fois créés, ainsi que le magique ramasse-miettes (*garbage collector*) qui détruit tous les objets devenus inutiles. D'autres questions seront abordées, comme le traitement des erreurs par les exceptions, le multithreading pour des interfaces utilisateur réactives, la programmation réseau et l'Internet. On y apprendra ce qui rend Java spécial, pourquoi il a tant de succès, ainsi que l'analyse et la conception orientées objet.

## ***Chapitre 2 : Tout est Objet***

Avec ce chapitre on arrive au point où l'on peut écrire un premier programme Java. Il doit donc donner une vision d'ensemble des choses essentielles, entre autres : le concept de *référence* à un objet ; comment créer un objet ; une introduction aux types primitifs et aux tableaux ; comment ils sont détruits par le ramasse-miettes ; comment toute chose est en Java un nouveau type de données (class) et comment créer vos propres classes ; les fonctions, leurs arguments et leur valeur de retour ; la visibilité des noms et l'utilisation de composants provenant d'autres bibliothèques ; le mot clef **static** ; les commentaires et la documentation intégrée.

## ***Chapitre 3 : Contrôler le Déroulement du Programme***

Ce chapitre commence avec tous les opérateurs provenant de C et C++. On y découvre les pièges classiques des opérateurs, le changement de type, la promotion et la priorité. Suivent le classique contrôle de flux de programme, les instructions de rupture de séquence déjà connues pour avoir été rencontrées dans d'autres langages de programmation : le choix avec *if-else*, la boucle avec *for* et *while* ; comment sortir d'une boucle avec *break* et *continue* aussi bien qu'avec les *break étiquetés* et les *continue étiquetés* (qui représentent le « goto manquant » en java) ; la sélection avec *switch*. Bien que la majorité de ces fonctionnalités ressemblent au code C et C++, il existe certaines différences. De plus, tous les exemples sont écrits en pur Java, afin de mieux montrer à quoi ressemble le langage.

## ***Chapitre 4 : Initialisation et Nettoyage Mémoire***

Ce chapitre commence par décrire le constructeur, lequel garantit une initialisation correcte. La définition du constructeur débouche sur le concept de surcharge de fonction (dans la mesure où plusieurs constructeurs peuvent coexister). La suite est une discussion sur le processus de nettoyage mémoire, qui n'est pas toujours aussi simple qu'il semblerait. Normalement, il suffit simplement d'abandonner un objet lorsqu'on n'en a plus besoin, et le ramasse-miettes finira par libérer la mémoire. Cette partie explore le ramasse-miettes ainsi que quelques-unes de ses particularités. Le chapitre se termine par une vision plus centrée sur l'initialisation : initialisation automatique des membres, spécification de l'initialisation des membres, ordre d'initialisation, initialisation **static** et initialisation des tableaux.

## ***Chapitre 5 : Cacher l'Implémentation***

Ce chapitre traite de la manière dont le code est mis en paquetages, et pourquoi certaines parties d'une bibliothèque sont visibles alors que d'autres sont cachées. Il s'intéresse tout d'abord aux mots clefs **package** et **import**, qui sont en relation avec la gestion des paquetages au niveau fichier et permettent de construire des bibliothèques de classes. Il examine ensuite le problème sous l'angle des chemins de dossier et des noms de fichiers. Le reste du chapitre traite des mots clefs **public**, **private** et **protected**, du concept de l'accès « amical » (accès par défaut, NdT), et de ce que signifient les différents niveaux de contrôle d'accès utilisés dans divers contextes.

## ***Chapitre 6 : Réutilisation des Classes***

Le concept d'héritage se retrouve dans pratiquement tous les langages de POO. Il s'agit de prendre une classe existante et d'étendre ses fonctionnalités (ou tout aussi bien les modifier, c'est le sujet du chapitre 7). L'héritage consiste toujours à réutiliser du code en gardant la même « classe de base », et en modifiant simplement

certaines choses çà et là afin d'obtenir ce que l'on veut. Toutefois, l'héritage n'est pas la seule manière de créer de nouvelles classes à partir de classes existantes. Il est également possible d'encapsuler un objet dans une nouvelle classe au moyen de la *composition*. Ce chapitre explique ces deux méthodes de réutilisation du code en Java, et comment les utiliser.

## *Chapitre 7 : Le Polymorphisme*

Si vous appreniez par vous-même, il vous faudrait neuf mois pour découvrir et comprendre le polymorphisme, l'une des pierres angulaires de la POO. Des exemples simples et courts montreront comment créer une famille de types au moyen de l'héritage et comment manipuler les objets dans cette famille par l'intermédiaire de leur classe de base. Le polymorphisme de Java permet de traiter de manière générique tout objet d'une famille, ce qui signifie que la plus grande partie du code n'est pas liée à une information spécifique sur le type. Ceci rend les programmes extensibles, et donc leur développement et leur maintenance plus simples et moins onéreux.

## *Chapitre 8 : Interfaces & Classes Internes*

Java fournit une troisième voie pour la réutilisation du code, avec l'*interface*, qui est une pure abstraction de l'interface d'un objet. L'**interface** est bien plus qu'une simple classe abstraite dont on aurait poussé l'abstraction à l'extrême, puisqu'il vous permet de développer une variation sur l'« héritage multiple » du C++, en créant une classe qui peut être transtypée vers plus d'un type de base.

Au premier abord, les classes internes ressemblent à un simple mécanisme permettant de cacher le code : on place des classes à l'intérieur d'autres classes. Vous apprendrez toutefois que la classe interne fait plus que cela - elle connaît la classe enveloppante et peut communiquer avec elle - et il est certain que le style de code que l'on écrit au moyen des classes internes est plus élégant et plus clair, bien que cela représente pour la plupart d'entre vous une nouvelle fonctionnalité nécessitant un certain temps d'apprentissage avant d'être maîtrisée.

## *Chapitre 9 : Stockage des Objets*

Un programme qui manipule un nombre fixe d'objets dont la durée de vie est connue ne peut être que clair et très simple. Mais généralement, les programmes créent de nouveaux objets à différents moments, qui ne seront connus que lors de l'exécution. De plus, avant l'exécution, on ne connaît ni le nombre ni parfois le type exact des objets qui seront nécessaires. Afin de résoudre ce problème général de la programmation, nous devons pouvoir créer n'importe quel nombre d'objets, à n'importe quel moment, n'importe où. Ce chapitre explore en profondeur la bibliothèque fournie par Java 2 pour ranger les objets durant leur existence : les tableaux simples et les conteneurs plus sophistiqués (structures de données) comme **ArrayList** et **HashMap**.

## *Chapitre 10 : Traitement des Erreurs au Moyen des Exceptions*

Java a pour philosophie de base qu'un code mal écrit ne sera jamais exécuté. Autant que possible, le compilateur repère les problèmes, mais parfois les problèmes - aussi bien une erreur de programmation qu'une condition d'erreur naturelle survenant lors de l'exécution normale du programme - ne peuvent être détectés et traités qu'au moment de l'exécution. Java possède un traitement des erreurs par les exceptions pour s'occuper de tout problème survenant pendant l'exécution. Ce chapitre examine comment fonctionnent en Java les mots clefs **try**, **catch**, **throw**, **throws**, et **finally** ; quand lancer des exceptions ; et ce que l'on doit faire si on les intercepte. Il expose aussi les exceptions standard de Java, comment créer vos propres exceptions, ce qu'il advient des exceptions dans les constructeurs, et comment sont localisés les codes de traitement d'exception.

## *Chapitre 11 : le Système d'E/S de Java*

En théorie, on peut diviser n'importe quel programme en trois parties : entrée, traitement, et sortie des données. Ceci suggère que les E/S (entrées/sorties) représentent une part importante de n'importe quel problème. Ce

chapitre étudie les différentes classes fournies par Java pour lire et écrire des fichiers, des blocs mémoire, ainsi que la console. Il montre la distinction entre E/S « vieux style » et E/S « nouveau style » Java. Il examine également le processus consistant à prendre un objet, le transformer en flux (de manière à pouvoir le ranger sur disque ou l'envoyer à travers un réseau) puis le reconstruire, ce qui est pris en charge par la *sérialisation des objets* de Java. Il présente également les bibliothèques de compression de Java, utilisées dans le format de fichier Java ARchive (JAR).

### ***Chapitre 12 : Identification Dynamique de Type***

L'identification dynamique de type de Java (Run-Time Type Identification, RTTI) permet de connaître le type exact d'un objet à partir d'une référence sur le type de base. Habituellement, on préfère ignorer intentionnellement le type exact d'un objet et laisser au mécanisme de liaison dynamique de Java (polymorphisme) le soin d'implémenter la signification correcte pour ce type. Mais de temps en temps il est très utile de connaître le type réel d'un objet pour lequel on n'a qu'une référence sur le type de base. Souvent cette information permet d'implémenter plus efficacement un traitement spécial. Ce chapitre explique à quoi sert la RTTI, comment l'utiliser, et comment s'en débarrasser lorsqu'on n'en a plus besoin. Enfin, il introduit le mécanisme de *réflexion* de Java.

### ***Chapitre 13 : Créer des Fenêtres et des Applets***

Java est livré avec la bibliothèque GUI « Swing », qui est un ensemble de classes traitant du fenêtrage d'une manière portable (NdT : sur différentes plates-formes). Ces programmes fenêtrés peuvent être soit des applets soit des applications autonomes. Ce chapitre est une introduction à Swing et à la création d'applets pour le World Wide Web. Il introduit aussi l'importante technologie des « JavaBeans », fondamentale pour la création d'outils de développement de programmes destinés au Développement Rapide d'Applications (RAD, Rapid-Application Development).

### ***Chapitre 14 : Les Threads Multiples***

Java fournit un moyen de créer de multiples sous-tâches concurrentes, appelées threads, s'exécutant dans le contexte d'un même programme (mis à part le cas où la machine possède plus d'un processeur, ceci n'a que l'apparence de sous-tâches multiples). Bien qu'on puisse les utiliser n'importe où, l'utilisation des threads est plus évidente lorsqu'il s'agit de créer une interface utilisateur réactive comme, par exemple, lorsqu'un certain processus gourmand en ressources système en cours d'exécution empêche un utilisateur d'utiliser un bouton ou d'entrer des données. Ce chapitre examine la syntaxe et la sémantique du multithreading en Java.

### ***Chapitre 15 : Informatique Distribuée***

Toutes les fonctionnalités et bibliothèques de Java semblent vraiment faites les unes pour les autres lorsqu'on commence à écrire des programmes qui travaillent en réseau. Ce chapitre explore la communication au travers des réseaux et sur l'Internet, ainsi que les classes fournies par Java pour faciliter cela. Il introduit les concepts très importants de *Servlets* et des *JSPs* (pour la programmation « côté serveur »), ainsi que la connectivité aux bases de données, *Java DataBase Connectivity* (JDBC), et l'invocation de méthodes distantes, *Remote Method Invocation* (RMI). Et, pour finir, une introduction aux nouvelles technologies *JINI*, *JavaSpaces*, et *Enterprise JavaBeans* (EJB).

### ***Annexe A : Passage & Retour d'Objets***

Etant donné qu'en Java seules les références permettent d'appréhender les objets, le concept de « passer un objet à une fonction » et celui de « retourner un objet depuis une fonction » ont quelques conséquences intéressantes. Cette annexe explique ce qu'il faut savoir afin de gérer les objets à l'entrée et à la sortie d'une fonction, et montre également la classe **String**, qui utilise une approche différente du problème.

## Annexe B : L'interface Java Natif (JNI)

Un programme Java entièrement portable a de sérieux inconvénients : la vitesse, et l'incapacité d'accéder à des services spécifiques de la plate-forme. Connaissant la plate-forme sur laquelle sera exécuté le programme, il est possible d'accélérer spectaculairement certaines opérations en les transformant en *méthodes natives*, qui sont des fonctions écrites dans un autre langage de programmation (actuellement, seuls C/C++ sont supportés). Cette annexe procure une courte introduction à cette fonctionnalité, suffisante pour qu'on puisse créer des exemples simples utilisant cette interface avec un code autre que Java.

## Annexe C : Conseils pour une Programmation Stylée en Java

Cet annexe est un ensemble de suggestions qui vous aideront dans la conception et le codage de bas niveau de votre application.

## Annexe D : Ressources

Une liste des livres sur Java que m'ont paru particulièrement utile.

# Exercices

Je me suis aperçu que des exercices simples sont très utiles pour consolider les connaissances des étudiants lors d'un séminaire, on en trouvera donc un ensemble à la fin de chaque chapitre.

La plupart d'entre eux sont conçus afin d'être assez simples pour être réalisés dans un temps raisonnable dans le contexte d'une salle de classe, pendant que l'instructeur vérifie que tous les étudiants ont assimilé le sujet de la leçon. Quelques exercices sont plus pointus, afin d'éviter l'ennui chez les étudiants expérimentés. La majorité est conçue pour être réalisés rapidement, ainsi que pour tester et perfectionner les connaissances. Quelques-uns présentent des difficultés, mais jamais de difficulté majeure. (Je présume que vous les découvrirez par vous-même — ou plutôt qu'ils vous trouveront).

Les solutions des exercices se trouvent dans le document électronique *The Thinking in Java Annotated Solution Guide*, disponible pour un faible coût sur <http://www.BruceEckel.com>.

# Le CD ROM Multimédia

Deux CD ROM multimédia sont associés à ce livre. Le premier est fourni avec le livre lui-même : *Thinking in C*, décrit à la fin de la préface, et consiste en une préparation à ce livre qui décrit la syntaxe C nécessaire à la compréhension de Java.

Il existe un deuxième CD ROM Multimédia, basé sur le contenu du livre. Ce CD ROM est un produit séparé et contient la **totalité** du séminaire d'une semaine de formation Java « Hands-On Java ». J'y ai enregistré plus de 15 heures de conférence, synchronisées avec des centaines de diapositives d'information. C'est un accompagnement idéal, dans la mesure où le séminaire est basé sur ce livre.

Le CD ROM contient toutes les conférences du séminaire de formation en immersion totale de cinq jours (il ne traite pas de l'attention portée aux cas particuliers !). Nous espérons que cela définira un nouveau standard de qualité.

Le CD ROM « Hands-On Java » est uniquement disponible en le commandant directement sur le site <http://www.BruceEckel.com>.

# Le Code Source

L'ensemble du code source de ce livre est disponible en freeware sous copyright, en une seule archive, en visitant le site Web <http://www.BruceEckel.com>. Afin que vous soyez certains d'obtenir la dernière version, cette adresse est celle du site officiel pour la distribution du code et de la version électronique du livre. Il existe des versions miroir du livre électronique et du code sur d'autres sites (dont certains sont référencés sur le site <http://www.BruceEckel.com>), mais il est préférable de rendre visite au site officiel afin de s'assurer que la version miroir est la plus récente. Vous êtes autorisés à distribuer le code à des fins d'enseignement ou d'éducation.

Le but essentiel du copyright est d'assurer que la source du code soit correctement citée, et d'éviter que le code soit utilisé sans autorisation dans un médium imprimé. (Tant que la source est citée, l'utilisation d'exemples provenant du livre ne pose généralement pas problème).

Chaque code source contient une référence à l'annonce suivante du copyright :

```
//:::CopyRight.txt

Copyright ©2000 Bruce Eckel

Source code file from the 2nd edition of the book

"Thinking in Java." All rights reserved EXCEPT as
allowed by the following statements:

You can freely use this file

for your own work (personal or commercial),
including modifications and distribution in
executable form only. Permission is granted to use
this file in classroom situations, including its
use in presentation materials, as long as the book
"Thinking in Java" is cited as the source.

Except in classroom situations, you cannot copy
and distribute this code; instead, the sole
distribution point is http://www.BruceEckel.com
(and official mirror sites) where it is
freely available. You cannot remove this
copyright and notice. You cannot distribute
modified versions of the source code in this
package. You cannot use this file in printed
media without the express permission of the
author. Bruce Eckel makes no representation about
```



the suitability of this software for any purpose.

It is provided "as is" without express or implied warranty of any kind, including any implied warranty of merchantability, fitness for a particular purpose or non-infringement. The entire risk as to the quality and performance of the software is with you. Bruce Eckel and the publisher shall not be liable for any damages suffered by you or any third party as a result of using or distributing software. In no event will Bruce Eckel or the publisher be liable for any lost revenue, profit, or data, or for direct, indirect, special, consequential, incidental, or punitive damages, however caused and regardless of the theory of liability, arising out of the use of or inability to use software, even if Bruce Eckel and the publisher have been advised of the possibility of such damages. Should the software prove defective, you assume the cost of all necessary servicing, repair, or correction. If you think you've found an error, please submit the correction using the form you will find at [www.BruceEckel.com](http://www.BruceEckel.com). (Please use the same form for non-code errors found in the book.)

///:~

Vous êtes autorisés à utiliser le code pour vos projets ainsi qu'à des fins d'éducation (ceci incluant vos cours) à la condition de conserver le copyright inclus dans chaque fichier source.

## Typographie et style de code

Dans ce livre, les identificateurs (de fonction, de variable, de nom de classe) sont écrits en **gras**. La plupart des mots clefs sont également en gras, à l'exception de ceux qui sont si souvent utilisés, tels que « class », que cela en deviendrait ennuyeux.

J'utilise un style de code particulier pour les exemples de ce livre. Ce style suit les règles que Sun utilise lui-même dans pratiquement tous les codes que l'on peut trouver sur son site (voir [java.sun.com/docs/codeconv/index.html](http://java.sun.com/docs/codeconv/index.html)), et semble être pris en compte par la plupart des environnements de

développement Java. Si vous avez lu mes autres livres, vous avez pu remarquer que le style de codage de Sun coïncide avec le mien — ce qui me fait évidemment plaisir, bien que je n'y sois pour rien. Le sujet du style de format demanderait des heures de débat assez chaud, aussi je vais simplement dire que je ne prétends pas imposer un style correct au travers de mes exemples, j'ai seulement mes propres motivations pour faire ainsi. Puisque Java est un langage de programmation indépendant de la forme, vous pouvez continuer à utiliser le style qui vous convient.

Les programmes de ce livre sont des fichiers directement inclus, au moyen du traitement de texte, depuis des fichiers ayant déjà subi une compilation. Par suite, le code imprimé dans le livre ne doit pas provoquer d'erreurs de compilation. Les erreurs qui *pourraient* entraîner des messages d'erreur lors de la compilation sont mis en commentaires au moyen de `//!` de manière à être facilement repérées et testées par des moyens automatiques. Les erreurs découvertes et rapportées à l'auteur feront d'abord l'objet d'une modification du code source distribué, puis, plus tard, d'une révision du livre (qui sera également disponible sur le site Web <http://www.BruceEckel.com>).

## Les versions de Java

Je me réfère généralement à l'implémentation Sun de Java pour déterminer la démarche correcte.

Depuis le début, Sun a fourni trois versions majeures de Java : 1.0, 1.1 et 2 (laquelle est appelée version 2 même si les versions du JDK de Sun continuent à être numérotées 1.2, 1.3, 1.4, etc.). La version 2 semble définitivement mettre Java en lumière, en particulier lorsqu'il est question des outils d'interface utilisateur. Ce livre en parle et a été testé avec Java 2, bien que je fasse de temps en temps des concessions aux fonctionnalités futures de Java 2 afin que le code soit compilable sous Linux (avec le JDK Linux disponible alors que j'écrivais ceci).

Si vous désirez apprendre les versions antérieures du langage non couvertes par cette édition, la première édition de ce livre est librement téléchargeable à l'adresse url : <http://www.BruceEckel.com>, vous la trouverez également dans le CD livré avec ce livre.

Attention : lorsqu'il m'a fallu mentionner les versions antérieures du langage, je n'utilise pas les numéros de sous-révision. Dans ce livre je fais uniquement référence à Java 1.0, Java 1.1, et Java 2, afin de me prémunir contre les erreurs typographiques qui pourraient résulter de futures sous-révisions de ces produits.

## Seminars and mentoring

*(non traduit, les personnes intéressées par les séminaires de Bruce Eckel devant à priori maîtriser l'anglais)*  
My company provides five-day, hands-on, public and in-house training seminars based on the material in this book. Selected material from each chapter represents a lesson, which is followed by a monitored exercise period so each student receives personal attention. The audio lectures and slides for the introductory seminar are also captured on CD ROM to provide at least some of the experience of the seminar without the travel and expense. For more information, go to <http://www.BruceEckel.com>.

My company also provides consulting, mentoring and walkthrough services to help guide your project through its development cycle — especially your company's first Java project.

## Errors

*(non traduit car cela concerne les erreurs relevées dans la version anglaise du livre de Bruce Eckel)*  
No matter how many tricks a writer uses to detect errors, some always creep in and these often leap off the page

for a fresh reader.

There is an error submission form linked from the beginning of each chapter in the HTML version of this book (and on the CD ROM bound into the back of this book, and downloadable from <http://www.BruceEckel.com>) and also on the Web site itself, on the page for this book. If you discover anything you believe to be an error, please use this form to submit the error along with your suggested correction. If necessary, include the original source file and note any suggested modifications. Your help is appreciated.

## À propos de la conception de la couverture du livre

La couverture de *Thinking in Java* est inspirée par le Mouvement des Arts et Métiers Américain (American Arts & Crafts Movement), qui commença peu avant le XX<sup>e</sup> siècle et atteignit son zénith entre 1900 et 1920. Il vit le jour en Angleterre en réaction à la fois contre la production des machines de la Révolution Industrielle et contre le style hautement ornemental de l'ère Victorienne. Arts & Crafts mit l'accent sur la sobriété, les formes de la nature telles que les voyait le mouvement « art nouveau » (en français dans le texte, NdT), le travail manuel, et l'importance des travailleurs et artisans particuliers, et encore n'ont-ils pas dédaigné l'utilisation des outils modernes. Il y a beaucoup de ressemblances avec la situation actuelle : le tournant du siècle, l'évolution des débuts inexpérimentés de la révolution informatique vers quelque chose de plus raffiné et significatif pour les individus, et l'engouement pour la connaissance du métier de programmeur face à la fabrication industrielle de code.

Je considère Java de la même manière : une tentative pour élever le programmeur au-dessus de la mécanique du système d'exploitation afin de l'amener à devenir un « artisan du logiciel ».

L'auteur, tout autant que le concepteur du livre et de sa couverture (qui sont amis depuis l'enfance) ont trouvé leur inspiration dans ce mouvement, et tous deux possèdent des meubles, lampes etc. soit originaux, soit inspirés par cette période.

L'autre thème de cette couverture suggère une boîte servant à la présentation des spécimens d'insectes recueillis par un naturaliste. Ces insectes sont des objets, qui sont placés dans des boîtes-objets. Les boîtes-objets sont elles-mêmes placées dans la « couverture-objet », ce qui illustre le concept fondamental d'agrégation en programmation orientée objet. Bien entendu, ceci n'est d'aucune utilité pour un programmeur, mais crée une association avec les « punaises » (« bugs »), ici les punaises ont été capturées, probablement tuées dans un bocal à spécimen, et pour finir confinées dans une petite boîte pour être exposées, comme pour montrer la capacité de Java à trouver, montrer, et soumettre les bugs (ce qui est vraiment l'un de ses attributs les plus puissants).

## Remerciements

En premier, merci à tous les associés qui travaillèrent avec moi pour encadrer des séminaires, faire du consulting, et développer des projets éducatifs : Andrea Provaglio, Dave Bartlett (qui a par ailleurs fortement contribué au Chapitre 15), Bill Venners, et Larry O'Brien. J'ai apprécié leur patience alors que je continuais de développer le meilleur modèle permettant à des personnes indépendantes comme nous de travailler ensemble. Merci à Rolf André Klaedtke (Suisse) ; Martin Vlcek, Martin Byer, Vlada & Pavel Lahoda, Martin the Bear, et Hanka (Prague) ; ainsi que Marco Cantu (Italie) pour leur hébergement lors de ma première tournée de conférences improvisée en Europe.

Merci aussi à la « Doyle Street Cohousing Community » pour m'avoir supporté durant les deux années nécessaires à la rédaction de cette première édition (ainsi que pour m'avoir supporté dans l'absolu). Bien des remerciements à Kevin et Sonda Donovan pour m'avoir accueilli dans leur magnifique Crested Butte, Colorado, l'été où je travaillais à la première édition du livre. Merci aussi à tous les amis résidents de « Crested Butte » et au « Rocky Mountain Biological Laboratory » qui m'ont si bien accueilli.

Merci également à Claudette Moore de « Moore Literary Agency » pour son énorme patience et sa constance à m'obtenir exactement ce je désirais.

Mes deux premiers livres ont été publiés sous la houlette de l'éditeur Jeff Pepper aux éditions Osborne/McGraw-Hill. Jeff est arrivé au bon endroit au bon moment à Prentice-Hall, il a débroussaillé le chemin et fait tout ce qu'il fallait pour rendre cette expérience de publication très agréable. Merci, Jeff — cela a compté pour moi.

J'ai une dette spéciale envers Gen Kiyooka et sa compagnie Digigami, qui m'ont gracieusement fourni un serveur Web les premières années. Cela a représenté pour moi une aide inestimable.

Merci à Cay Horstmann (co-auteur de *Core Java*, Prentice-Hall, 2000), D'Arcy Smith (Symantec), et Paul Tyma (co-auteur de *Java Primer Plus*, The Waite Group, 1996), pour m'avoir aidé à clarifier les concepts du langage.

Merci aux personnes qui ont pris la parole dans mon cursus Java à la Software Development Conference, aux étudiants de mes séminaires, pour m'avoir posé les bonnes questions qui m'ont permis de clarifier mes cours.

Je remercie spécialement Larry et Tina O'Brien, qui m'ont aidé à mettre mon séminaire sur le CD ROM original *Hands-On Java* (vous en saurez davantage sur <http://www.BruceEckel.com>).

Beaucoup de gens m'ont envoyé des correctifs et je reste en dette avec eux, mais je dois remercier particulièrement (pour la première édition) : Kevin Raulerson (qui repéra des tonnes d'énormes bugs), Bob Resendes (tout simplement incroyable), John Pinto, Joe Dante, Joe Sharp (les trois, fabuleux), David Combs (beaucoup de corrections grammaticales et d'éclaircissements), Dr. Robert Stephenson, John Cook, Franklin Chen, Zev Griner, David Karr, Leander A. Stroschein, Steve Clark, Charles A. Lee, Austin Maher, Dennis P. Roth, Roque Oliveira, Douglas Dunn, Dejan Ristic, Neil Galarneau, David B. Malkovsky, Steve Wilkinson, ainsi qu'une foule d'autres. Prof. Ir. Marc Meurrens déploya beaucoup d'efforts de publication et réalisa la version électronique de la première édition du livre disponible en Europe.

J'ai rencontré dans ma vie une avalanche de personnes très techniques et très intelligentes qui sont devenues des amis mais qui étaient également peu communes et qui m'ont influencé en ce qu'elles pratiquaient le yoga ainsi que d'autres formes de spiritualité, ce qui m'a instruit et inspiré. Ce sont Kraig Brockschmidt, Gen Kiyooka, et Andrea Provaglio (qui aida à la compréhension de Java et de la programmation en général en Italie, et qui est maintenant aux Etats-Unis associé à l'équipe MindView).

Ce ne fut pas une grande surprise pour moi de découvrir que ma connaissance de Delphi m'a aidé à comprendre Java, car ces deux langages ont beaucoup de concepts et de décisions de conception de langage en commun. Des amis fanatiques de Delphi m'ont aidé à devenir plus perspicace à propos de ce merveilleux environnement de programmation. Ce sont Marco Cantu (un autre italien — il se pourrait qu'être imprégné de culture latine donne certaines aptitudes pour la programmation ?), Neil Rubenking (qui se nourrissait de culture yoga/végétarienne/Zen avant de découvrir les ordinateurs), et bien entendu Zack Urlocker, un vieux copain avec qui j'ai parcouru le monde.

La perspicacité et l'aide de mon ami Richard Hale Shaw m'ont été fort utiles (celles de Kim également). Richard et moi avons passé de concert beaucoup de mois à donner des séminaires et à tenter de trouver l'enseignement parfait pour les auditeurs. Merci également à KoAnn Vikoren, Eric Faurot, Marco Pardi, ainsi qu'à toute l'équipe du MFI. Merci particulièrement à Tara Arrowood, qui me rendit confiance à propos des possibilités de conférences.

La conception du livre, de la couverture, ainsi que la photo de couverture sont dûs à mon ami Will-Harris, auteur et concepteur connu (<http://www.Will-Harris.com>), qui jouait au collège avec des lettres transfert en

attendant l'invention de la publication assistée par ordinateur, tout en se plaignant de mes grognements à propos de mes problèmes d'algèbre. Toutefois, j'ai produit moi-même mes pages prêtes à imprimer, et donc j'assume mes erreurs de frappe. Microsoft® Word 97 for Windows a été utilisé pour écrire le livre et Adobe Acrobat pour créer les pages offset ; le livre est sorti directement des fichiers PDF d'Acrobat (pour rendre hommage à l'ère électronique, je me trouvais outre-mer les deux fois où la version finale du livre fut produite — la première édition fut envoyée depuis Le Cap en Afrique du Sud et la seconde depuis Prague). Les polices de caractères sont *Georgia* pour le corps du texte et *Verdana* pour les titres. La police de couverture est *ITC Rennie Mackintosh*.

Merci aux groupes qui ont créé les compilateurs : Borland, le Blackdown group (pour Linux), et bien entendu, Sun.

Je remercie particulièrement tous mes maîtres et tous mes étudiants (qui furent en même temps mes maîtres). Le plus plaisant de mes maîtres en écriture fut Gabrielle Rico (auteur de *Writing the Natural Way*, Putnam, 1983). Je garderai précieusement le souvenir d'une formidable semaine à Esalen.

Liste non exhaustive de mes collaborateurs : Andrew Binstock, Steve Sinofsky, JD Hildebrandt, Tom Keffer, Brian McElhinney, Brinkley Barr, Bill Gates au *Midnight Engineering Magazine*, Larry Constantine et Lucy Lockwood, Greg Perry, Dan Putterman, Christi Westphal, Gene Wang, Dave Mayer, David Intersimone, Andrea Rosenfield, Claire Sawyers, d'autres italiens (Laura Fallai, Corrado, Ilsa, et Cristina Giustozzi), Chris et Laura Strand, les Almquists, Brad Jerbic, Marilyn Cvitanic, les Mabrys, les Haflingers, les Pollocks, Peter Vinci, la famille Robbins, la famille Moelter (ainsi que les McMillans), Michael Wilk, Dave Stoner, Laurie Adams, les Cranstons, Larry Fogg, Mike et Karen Sequeira, Gary Entsminger et Allison Brody, Kevin Donovan et Sonda Eastlack, Chester et Shannon Andersen, Joe Lordi, Dave et Brenda Bartlett, David Lee, les Rentschlers, les Sudeks, Dick, Patty, et Lee Eckel, Lynn et Todd, et leurs familles. Et, bien entendu, Papa et Maman.

## Collaborateurs Internet

Merci à tous ceux qui m'ont aidé à réécrire les exemples au moyen de la bibliothèque Swing, ou pour d'autres choses : Jon Shvarts, Thomas Kirsch, Rahim Adatia, Rajesh Jain, Ravi Manthena, Banu Rajamani, Jens Brandt, Nitin Shivaram, Malcolm Davis, ainsi qu'à tous ceux qui se sont exprimés. Cela m'a réellement aidé à mettre le projet à jour.