

Peter Haggar

Mieux programmer en Java

**68 astuces pour
optimiser son code**

- ▶ Programmer « juste » du premier coup
- ▶ Les pièges à éviter

 **Eyrolles**

2-005-631-1

Mieux programmer en **Java**

68 astuces pour optimiser son code

Peter Haggar



Eyrolles

Table des matières

Sommaire détaillé	XI
-------------------------	----

Avant-propos	XXI
--------------------	-----

Remerciements	XXV
---------------------	-----

1. Techniques générales	1
-------------------------------	---

ATELIER 1 Le passage des paramètres se fait par valeur et non par référence	1
---	---

ATELIER 2 Emploi de final pour des données ou des références objet constantes	3
---	---

ATELIER 3 Toute méthode non statique peut être surchargée par défaut	5
--	---

ATELIER 4 Tableaux ou vecteurs : que choisir ?	6
--	---

ATELIER 5 Préférer le polymorphisme à l'instruction instanceof	10
--	----

ATELIER 6 Dans quels cas utiliser instanceof ?	13
--	----

ATELIER 7 Quand assigner la valeur null à des références objet ?	16
--	----

2. Objets et test d'égalité	21
-----------------------------------	----

ATELIER 8 Types références et types primitifs : les différences	21
---	----

ATELIER 9 Opérateur == et méthode equals : les différences	25
--	----

ATELIER 10 Implémentation par défaut de equals : danger	28
---	----

ATELIER 11 Implémenter judicieusement la méthode equals	37
---	----

ATELIER 12 Préférer getClass dans les implémentations de la méthode equals	37
--	----

ATELIER 13	Appel de <code>super.equals</code> des classes de base	40
ATELIER 14	Emploi de <code>instanceof</code> dans les implémentations de la méthode <code>equals</code>	44
ATELIER 15	Règles à suivre dans l'implémentation d'une méthode <code>equals</code>	51
3. Gestion des exceptions		53
ATELIER 16	Mécanismes de flux de contrôle des exceptions	54
ATELIER 17	Ne jamais ignorer une exception	57
ATELIER 18	Ne jamais masquer une exception	59
ATELIER 19	Un inconvénient de la clause <code>throws</code>	64
ATELIER 20	Utilisez la clause <code>throws</code> de manière explicite et exhaustive	65
ATELIER 21	Économisez des ressources grâce à <code>finally</code>	67
ATELIER 22	Ne pas revenir d'un bloc <code>try</code>	69
ATELIER 23	Placer les blocs <code>try/catch</code> en dehors des boucles	70
ATELIER 24	Ne pas utiliser les exceptions pour les contrôles de flux	74
ATELIER 25	Ne pas employer les exceptions pour toutes les situations d'erreur ...	74
ATELIER 26	Traitement des exceptions à partir des constructeurs	76
ATELIER 27	Retour des objets à un état valide avant la récupération d'une exception	77
4. Performances		85
ATELIER 28	Soigner l'architecture du programme, les structures de données et les algorithmes mis en œuvre	87
ATELIER 29	Optimisation du code source par le compilateur : méfiance	89
ATELIER 30	Optimisation du code d'exécution	93
ATELIER 31	Utiliser <code>StringBuffer</code> plutôt que <code>String</code> pour la concaténation	94
ATELIER 32	Minimiser le coût de la création d'objet en termes de ressources	96
ATELIER 33	Ne pas créer d'objets inutiles	100
ATELIER 34	Éviter la synchronisation	101
ATELIER 35	Utilisez les variables de pile le plus souvent possible	106
ATELIER 36	Employer les méthodes <code>static</code> , <code>final</code> et <code>private</code> pour l'inlining ...	109
ATELIER 37	N'initialiser qu'une fois les variables d'instance	111
ATELIER 38	Utiliser des types primitifs pour obtenir du code plus rapide et moins volumineux	113

ATELIER 39	Ne pas parcourir un objet Vector avec une Enumeration ou un Iterator	117
ATELIER 40	Utiliser System.arraycopy pour copier des tableaux	118
ATELIER 41	Préférer un objet tableau à un objet Vector ou ArrayList	120
ATELIER 42	Réutilisation des objets	122
ATELIER 43	Emploi de l'évaluation différée	125
ATELIER 44	Optimiser le code source manuellement	132
ATELIER 45	Compilation en code natif	138
5. Programmation multithread		141
ATELIER 46	Pour les méthodes d'instance, le mot-clé synchronized verrouille les objets, mais non les méthodes ni le code	142
ATELIER 47	Bien distinguer les méthodes statiques synchronized et les méthodes d'instance synchronized	145
ATELIER 48	Dans une méthode d'accès, utiliser des données de type private et non de type public ou protected	149
ATELIER 49	Toute synchronisation qui n'est pas indispensable doit être évitée ...	152
ATELIER 50	Utiliser l'instruction synchronized ou volatile pour accéder aux variables partagées	154
ATELIER 51	Verrouillage de tous les objets invoqués en une unique opération	157
ATELIER 52	Acquisition de verrous multiples dans un ordre fixe et global pour éviter l'interblocage	158
ATELIER 53	Préférer notifyAll à notify	162
ATELIER 54	Emploi de verrous tournants pour wait et notifyAll	163
ATELIER 55	Préférer wait et notifyAll aux boucles	167
ATELIER 56	Ne pas réaffecter la référence objet d'un objet verrouillé	169
ATELIER 57	Ne pas invoquer les méthodes stop ou suspend	171
ATELIER 58	Terminer les threads grâce à la coopération de thread	173
6. Classes et interfaces		175
ATELIER 59	Utilisation des interfaces pour supporter l'héritage multiple	176
ATELIER 60	Conflits de méthodes dans les interfaces	179
ATELIER 61	Utilisation des classes abstract pour fournir une implémentation partielle	182
ATELIER 62	Différences entre une interface, une classe abstract et une classe concrète	185

ATELIER 63	Définition et implémentation judicieuses de classes constantes	186
ATELIER 64	Utilisation de <code>clone</code> pour les objets constants lors du passage ou de la réception de références objet vers des objets modifiables	188
ATELIER 65	Utiliser l'héritage ou la délégation pour définir des classes constantes	196
ATELIER 66	Appel de <code>super.clone</code> lors de l'implémentation d'une méthode <code>clone</code>	203
ATELIER 67	Ne pas invoquer de méthode <code>finalize</code> pour nettoyer les ressources autres que la mémoire	205
ATELIER 68	Utiliser avec prudence les appels des méthodes non <code>final</code> depuis un constructeur	207
 Annexes		 211
 Apprendre Java		 213
 Bibliographie		 215
 Index		 217