
Table des matières

Préface	XIII
Introduction	XIX
Sur la couverture	XXIII
1 Code propre	1
Il y aura toujours du code	2
Mauvais code	3
Coût total d'un désordre	4
L'utopie de la grande reprise à zéro	4
Attitude	5
L'énigme primitive	6
L'art du code propre	7
Qu'est-ce qu'un code propre ?	7
Écoles de pensée	14
Nous sommes des auteurs	15
La règle du boy-scout	16
Préquel et principes	17
Conclusion	17
2 Noms significatifs	19
Choisir des noms révélateurs des intentions	20
Éviter la désinformation	22
Faire des distinctions significatives	23
Choisir des noms prononçables	24
Choisir des noms compatibles avec une recherche	25
Éviter la codification	26
Notation hongroise	26
Préfixes des membres	27
Interfaces et implémentations	27
Éviter les associations mentales	28
Noms des classes	28
Noms des méthodes	29
Ne pas faire le malin	29
Choisir un mot par concept	30
Éviter les jeux de mots	30

Choisir des noms dans le domaine de la solution	31
Choisir des noms dans le domaine du problème	31
Ajouter un contexte significatif	31
Ne pas ajouter de contexte inutile	33
Mots de la fin	34
3 Fonctions	35
Faire court	38
Blocs et indentation	39
Faire une seule chose	39
Sections à l'intérieur des fonctions	41
Un niveau d'abstraction par fonction	41
Lire le code de haut en bas : la règle de décroissance	41
Instruction <i>switch</i>	42
Choisir des noms descriptifs	44
Arguments d'une fonction	45
Formes unaires classiques	46
Arguments indicateurs	46
Fonctions diadiques	47
Fonctions triadiques	47
Objets en argument	48
Listes d'arguments	48
Verbes et mots-clés	49
Éviter les effets secondaires	49
Arguments de sortie	50
Séparer commandes et demandes	51
Préférer les exceptions au retour de codes d'erreur	51
Extraire les blocs <i>try/catch</i>	52
Traiter les erreurs est une chose	53
L'aimant à dépendances <i>Error.java</i>	53
Ne vous répétez pas	54
Programmation structurée	54
Écrire les fonctions de la sorte	55
Conclusion	55
<i>SetupTeardownInclude</i>	56
4 Commentaires	59
Ne pas compenser le mauvais code par des commentaires	61
S'expliquer dans le code	61
Bons commentaires	62
Commentaires légaux	62
Commentaires informatifs	62
Expliquer les intentions	63

Clarifier	63
Avertir des conséquences	64
Commentaires <i>TODO</i>	65
Amplifier	66
Documentation Javadoc dans les API publiques	66
Mauvais commentaires	66
Marmonner	66
Commentaires redondants	67
Commentaires trompeurs	69
Commentaires obligés	70
Commentaires de journalisation	70
Commentaires parasites	71
Bruit effrayant	73
Ne pas remplacer une fonction ou une variable par un commentaire	73
Marqueurs de position	74
Commentaires d'accolade fermante	74
Attributions et signatures	75
Mettre du code en commentaire	75
Commentaires HTML	76
Information non locale	76
Trop d'informations	77
Lien non évident	77
En-têtes de fonctions	78
Documentation Javadoc dans du code non public	78
Exemple	78
5 Mise en forme	83
Objectif de la mise en forme	84
Mise en forme verticale	84
Métaphore du journal	86
Espacement vertical des concepts	86
Concentration verticale	87
Distance verticale	88
Rangement vertical	93
Mise en forme horizontale	93
Espacement horizontal et densité	94
Alignement horizontal	95
Indentation	97
Portées fictives	99
Règles d'une équipe	99
Règles de mise en forme de l'Oncle Bob	100

6 Objets et structures de données	103
Abstraction de données	104
Antisymétrie données/objet	105
Loi de Déméter	108
Catastrophe ferroviaire	108
Hybrides	109
Cacher la structure	110
Objets de transfert de données	110
Enregistrement actif	111
Conclusion	112
7 Gestion des erreurs	113
Utiliser des exceptions à la place des codes de retour	114
Commencer par écrire l'instruction <i>try-catch-finally</i>	115
Employer des exceptions non vérifiées	117
Fournir un contexte avec les exceptions	118
Définir les classes d'exceptions en fonction des besoins de l'appelant	118
Définir le flux normal	120
Ne pas retourner <i>null</i>	121
Ne pas passer <i>null</i>	122
Conclusion	123
8 Limites	125
Utiliser du code tiers	126
Explorer et apprendre les limites	128
Apprendre <i>log4j</i>	128
Les tests d'apprentissage sont plus que gratuits	130
Utiliser du code qui n'existe pas encore	131
Limites propres	132
9 Tests unitaires	133
Les trois lois du TDD	135
Garder des tests propres	135
Les tests rendent possibles les "-ilities"	136
Tests propres	137
Langage de test propre à un domaine	140
Deux standards	140
Une assertion par test	143
Un seul concept par test	144
F.I.R.S.T.	145
Conclusion	146

10 Classes	147
Organiser une classe	148
Encapsulation	148
De petites classes	148
Principe de responsabilité unique	150
Cohésion	152
Maintenir la cohésion mène à de nombreuses petites classes	153
Organiser en vue du changement	159
Cloisonner le changement	162
11 Systèmes	165
Construire une ville	166
Séparer la construction d'un système de son utilisation	166
Construire dans la fonction <i>main</i>	167
Fabriques	168
Injection de dépendance	169
Grandir	170
Préoccupations transversales	173
Proxies Java	174
Frameworks AOP en Java pur	176
Aspects d'AspectJ	179
Piloter l'architecture du système par les tests	179
Optimiser la prise de décision	181
Utiliser les standards judicieusement, lorsqu'ils apportent une valeur démontrable ...	181
Les systèmes ont besoin de langages propres à un domaine	182
Conclusion	182
12 Émergences	183
Obtenir la propreté par une conception émergente	183
Règle de conception simple n° 1 : le code passe tous les tests	184
Règles de conception simple n° 2 à 4 : remaniement	185
Pas de redondance	185
Expressivité	188
Un minimum de classes et de méthodes	189
Conclusion	189
13 Concurrency	191
Raisons de la concurrence	192
Mythes et idées fausses	193
Défis	194
Se prémunir des problèmes de concurrence	195
Principe de responsabilité unique	195
Corollaire : limiter la portée des données	195

Corollaire : utiliser des copies des données	196
Corollaire : les threads doivent être aussi indépendants que possible	196
Connaître la bibliothèque	197
Collections sûres vis-à-vis des threads	197
Connaître les modèles d'exécution	198
Producteur-consommateur	198
Lecteurs-rédacteurs	199
Dîner des philosophes	199
Attention aux dépendances entre des méthodes synchronisées	200
Garder des sections synchronisées courtes	200
Écrire du code d'arrêt est difficile	201
Tester du code multithread	202
Considérer les faux dysfonctionnements comme des problèmes potentiellement liés au multithread	202
Commencer par rendre le code normal opérationnel	203
Faire en sorte que le code multithread soit enfichable	203
Faire en sorte que le code multithread soit réglable	203
Exécuter le code avec plus de threads que de processeurs	204
Exécuter le code sur différentes plates-formes	204
Instrumenter le code pour essayer et forcer des échecs	204
Instrumentation manuelle	205
Instrumentation automatisée	206
Conclusion	207
14 Améliorations successives	209
Implémentation de <i>Args</i>	210
Comment ai-je procédé ?	216
<i>Args</i> : le brouillon initial	217
J'ai donc arrêté	228
De manière incrémentale	228
Arguments de type <i>String</i>	231
Conclusion	268
15 Au cœur de JUnit	269
Le framework JUnit	270
Conclusion	283
16 Remaniement de <i>SerialDate</i>	285
Premièrement, la rendre opérationnelle	286
Puis la remettre en ordre	288
Conclusion	303

17 Indicateurs et heuristiques	305
Commentaires	306
C1 : informations inappropriées	306
C2 : commentaires obsolètes	306
C3 : commentaires redondants	306
C4 : commentaires mal rédigés	307
C5 : code mis en commentaire	307
Environnement	307
E1 : la construction exige plusieurs étapes	307
E2 : les tests exigent plusieurs étapes	308
Fonctions	308
F1 : trop grand nombre d'arguments	308
F2 : arguments de sortie	308
F3 : arguments indicateurs	308
F4 : fonction morte	308
Général	309
G1 : multiples langages dans un même fichier source	309
G2 : comportement évident non implémenté	309
G3 : comportement incorrect aux limites	309
G4 : sécurités neutralisées	310
G5 : redondance	310
G6 : code au mauvais niveau d'abstraction	311
G7 : classes de base qui dépendent de leurs classes dérivées	312
G8 : beaucoup trop d'informations	312
G9 : code mort	313
G10 : séparation verticale	313
G11 : incohérence	314
G12 : désordre	314
G13 : couplage artificiel	314
G14 : envie de fonctionnalité	314
G15 : arguments sélecteurs	316
G16 : intentions obscures	316
G17 : responsabilité mal placée	317
G18 : méthodes statiques inappropriées	317
G19 : utiliser des variables explicatives	318
G20 : les noms des fonctions doivent indiquer leur rôle	319
G21 : comprendre l'algorithme	319
G22 : rendre physiques les dépendances logiques	320
G23 : préférer le polymorphisme aux instructions <i>if/else</i> ou <i>switch/case</i>	321
G24 : respecter des conventions standard	322
G25 : remplacer les nombres magiques par des constantes nommées	322
G26 : être précis	323
G27 : privilégier la structure à une convention	324

G28 : encapsuler les expressions conditionnelles	324
G29 : éviter les expressions conditionnelles négatives	324
G30 : les fonctions doivent faire une seule chose	324
G31 : couplages temporels cachés	325
G32 : ne pas être arbitraire	326
G33 : encapsuler les conditions aux limites	327
G34 : les fonctions doivent descendre d'un seul niveau d'abstraction	327
G35 : conserver les données configurables à des niveaux élevés	329
G36 : éviter la navigation transitive	329
Java	330
J1 : éviter les longues listes d'importations grâce aux caractères génériques ...	330
J2 : ne pas hériter des constantes	331
J3 : constantes contre énumérations	332
Noms	333
N1 : choisir des noms descriptifs	333
N2 : choisir des noms au niveau d'abstraction adéquat	334
N3 : employer si possible une nomenclature standard	335
N4 : noms non ambigus	335
N5 : employer des noms longs pour les portées longues	336
N6 : éviter la codification	336
N7 : les noms doivent décrire les effets secondaires	337
Tests	337
T1 : tests insuffisants	337
T2 : utiliser un outil d'analyse de couverture	337
T3 : ne pas omettre les tests triviaux	337
T4 : un test ignoré est une interrogation sur une ambiguïté	337
T5 : tester aux conditions limites	338
T6 : tester de manière exhaustive autour des bogues	338
T7 : les motifs d'échec sont révélateurs	338
T8 : les motifs dans la couverture des tests sont révélateurs	338
T9 : les tests doivent être rapides	338
Conclusion	338
Annexe A Concurrency II	339
Exemple client/serveur	339
Le serveur	339
Ajouter des threads	341
Observations concernant le serveur	341
Conclusion	343
Chemins d'exécution possibles	344
Nombre de chemins	344
Examen plus approfondi	346
Conclusion	349

Connaître sa bibliothèque	349
Framework <i>Executor</i>	349
Solutions non bloquantes	350
Classes non sûres vis-à-vis des threads	351
Impact des dépendances entre méthodes sur le code concurrent	352
Tolérer la panne	354
Verrouillage côté client	354
Verrouillage côté serveur	356
Augmenter le débit	357
Calculer le débit en mode monothread	358
Calculer le débit en mode multithread	358
Interblocage	359
Exclusion mutuelle	361
Détention et attente	361
Pas de préemption	361
Attente circulaire	361
Briser l'exclusion mutuelle	362
Briser la détention et l'attente	362
Briser la préemption	362
Briser l'attente circulaire	363
Tester du code multithread	363
Outils de test du code multithread	367
Conclusion	367
Code complet des exemples	368
Client/serveur monothread	368
Client/serveur multithread	371
Annexe B org.jfree.date.SerialDate	373
Annexe C Référence des heuristiques	431
Bibliographie	435
Épilogue	437
Index	439