

ЗМІСТ

ПЕРЕДМОВА.....	13
ВСТУП	18
ПОДЯКИ.....	20
РОЗДІЛ 1. ЧИСТИЙ КОД.....	21
Хай живе код	22
Поганий код	22
Розплата за хаос	23
Грандіозне перероблення.....	24
Відносини.....	25
Головний парадокс.....	26
Мистецтво чистого коду?.....	26
Що таке «чистий код»?	26
Школи думки	32
Ми — автори	33
Правило бойскаута.....	34
Передісторія й принципи.....	34
Висновки	35
Література.....	35
РОЗДІЛ 2. ЗМІСТОВНІ ІМЕНА	36
Імена мають передавати наміри програміста.....	37
Уникайте дезінформації	38
Використовуйте осмислені відмінності	39
Використовуйте імена, що зручно вимовляти	41
Вибирайте імена, зручні для пошуку	42
Уникайте схем кодування імен.....	42
Угорський запис	43
Префікси членів класів.....	43
Інтерфейси та реалізації.....	44
Уникайте ментальних перетворень.....	44
Імена класів	45

Імена методів	45
Уникайте дотепів.	45
Виберіть одне слово для кожної концепції	46
Утримуйтеся від каламбурів	46
Надавайте імена в термінах рішення	47
Надавайте імена в термінах задачі	47
Додайте змістовний контекст	47
Не додавайте надлишкового контексту	49
Кілька слів наостанок	50
РОЗДІЛ 3. ФУНКЦІЇ.....	51
Компактність!.....	54
<i>Блоки й відступи</i>	55
Правило однієї операції	55
<i>Секції у функціях.</i>	56
Один рівень абстракції на функцію	56
<i>Читання коду згори вниз: правило зниження</i>	57
Команди switch	57
Використовуйте змістовні імена.....	59
Аргументи функцій	60
<i>Стандартні унарні форми.</i>	61
<i>Аргументи-прапори</i>	62
<i>Бінарні функції.</i>	62
<i>Тернарні функції</i>	63
<i>Об'єкти як аргументи.</i>	63
<i>Списки аргументів</i>	63
<i>Дієслова й ключові слова</i>	64
Позбавтеся побічних ефектів	64
<i>Вихідні аргументи</i>	65
Поділ команд і запитів	66
Використовуйте винятки замість повернення кодів помилок	67
<i>Виділіть блоки try/catch.</i>	67
<i>Оброблення помилок як одна операція</i>	68
<i>Магніт залежностей Error.Java.</i>	68
Не повторюйтеся.....	69
Структурне програмування.....	69
Як навчитися писати такі функції?	70
Висновок	70
Література.....	73

РОЗДІЛ 4. КОМЕНТАРІ	74
Коментарі не компенсують поганого коду	76
Поясніть свої наміри в коді	76
Гарні коментарі	76
Юридичні коментарі	77
Інформативні коментарі	77
Презентування намірів	77
Прояснення	78
Попередження про наслідки	79
Коментарі TODO	80
Посилення	80
Коментарі Javadoc у загальнодоступних API	81
Погані коментарі	81
Бурмотіння	81
Надлишкові коментарі	82
Недостовірні коментарі	84
Обов'язкові коментарі	85
Журнальні коментарі	85
Шум	86
Небезпечний шум	88
Не використовуйте коментарі там, де можна використати функцію або змінну	89
Позиційні маркери	89
Коментарі за закритою фігурною дужкою	89
Посилання на авторів	90
Закоментований код	90
HTML коментарі	91
Нелокальна інформація	92
Занадто багато інформації	92
Неочевидні коментарі	93
Заголовки функцій	93
Заголовки Javadoc у внутрішньому коді	93
Приклад	93
Література	97
РОЗДІЛ 5. ФОРМАТУВАННЯ	98
Мета форматування	99
Вертикальне форматування	99
Газетна метафора	100
Вертикальний розподіл концепцій	101
Вертикальне стиснення	102
Вертикальні відстані	103
Вертикальне впорядкування	108

Горизонтальне форматування.....	108
Горизонтальний розподіл і стиснення	109
Горизонтальне вирівнювання.....	110
Відступи	111
Вироджені ділянки видимості	113
Правила форматування в командах	113
Правила форматування від дядечка Боба.....	114
РОЗДІЛ 6. ОБ'ЄКТИ Й СТРУКТУРИ ДАНИХ	117
Абстракція даних	118
Антисиметрія даних/об'єктів	119
Закон Деметри.....	122
«Аварія потяга»	122
Гібриди	123
Приховування структури.....	123
Об'єкти передавання даних.....	124
Активні записи	125
Висновки	126
Література.....	126
РОЗДІЛ 7. ОБРОБЛЕННЯ ПОМИЛОК (Майкл Фізерс)	127
Використовуйте винятки замість кодів помилок	128
Почніть із написання команди try-catch-finally	129
Використовуйте неперевірні винятки.....	131
Передавайте контекст із винятками	132
Визначайте класи винятків у контексті потреб викличної сторони	132
Визначте нормальний шлях виконання.....	134
Не повертайте null	135
Не передавайте null	136
Висновки	138
Література.....	138
РОЗДІЛ 8. МЕЖІ	139
Використання стороннього коду	140
Дослідження та аналіз меж	142
Вивчення <i>log4j</i>	142
Навчальні тести: вигідніше, ніж безкоштовно.....	144
Використання неіснуючого коду	145

Чисті межі.....	146
Література.....	146
РОЗДІЛ 9. МОДУЛЬНІ ТЕСТИ.....	147
Три закони TDD	148
Про чистоту тестів.....	149
<i>Тести як засіб забезпечення змін</i>	150
Чисті тести	151
<i>Предметно-орієнтована мова тестування</i>	153
<i>Подвійний стандарт</i>	154
Одна перевірка на тест	156
<i>Одна концепція на тест</i>	157
FIRST	159
Висновки	160
Література.....	160
РОЗДІЛ 10. КЛАСИ (спільно з Джеффері Лангстром).....	161
Будова класу.....	161
<i>Інкапсуляція</i>	162
Класи мають бути компактними!.....	162
<i>Принцип єдиної відповідальності (SRP)</i>	164
<i>Зв'язність</i>	166
<i>Підтримання зв'язності призводить до зменшення класів</i>	167
Структурування з урахуванням змін	173
<i>Ізоляція змін</i>	176
Література.....	178
РОЗДІЛ 11. СИСТЕМИ (Кевін Дін Вомплер)	179
Як би ви будували місто?.....	179
Відокремлення конструювання системи від її використання	180
<i>Відокремлення таін</i>	181
<i>Фабрики</i>	182
<i>Впровадження залежностей</i>	182
Масштабування.....	183
<i>Перехресні ділянки відповідальності</i>	186
Посередники.....	187
АОП-інфраструктури «чистою» Java.....	189
Аспекти AspectJ	192
Випробування системної архітектури	193
Оптимізація прийняття рішень	194

Застосовуйте стандарти розумно, коли вони приносять очевидну користь.....	194
Системам необхідні предметно-орієнтовані мови	195
Висновки	195
Література.....	196
РОЗДІЛ 12. ФОРМУВАННЯ АРХІТЕКТУРИ (Джефф Лангр)	197
Чотири правила.....	197
Правило № 1: виконання всіх тестів.....	198
Правила № 2–4: перероблення коду	198
Відсутність дублювання	199
Виразність.....	201
Мінімум класів і методів	202
Висновки	203
Література.....	203
РОЗДІЛ 13. БАГАТОПОТОКОВІСТЬ (Бретт Л. Шухерт)	204
Навіщо потрібна багатопотоковість?	205
<i>Міфи й неправильні уявлення</i>	206
Труднощі	207
Захист від помилок багатопотоковості.....	207
<i>Принцип єдиної відповідальності</i>	208
<i>Наслідки: обмежуйте ділянку видимості даних</i>	208
<i>Наслідки: використовуйте копії даних</i>	208
<i>Наслідки: потоки мають бути якомога більш незалежними</i> ...	209
Знайте свою бібліотеку	209
<i>Потоково-безпечні колекції</i>	209
Знайте моделі виконання	210
<i>Модель «виробник / споживач»</i>	211
<i>Модель «читач / письменник»</i>	211
<i>Модель «філософи за обідом»</i>	211
Остерігайтеся залежностей між синхронізованими методами.....	212
Синхронізовані секції повинні мати мінімальний розмір	213
Про труднощі коректного завершення	213
Тестування багатопотокового коду	214
<i>Розглядайте неперіодичні збої як ознаки можливих проблем багатопотоковості</i>	214
<i>Почніть із налагодження основного коду, не пов'язаного з багатопотоковістю</i>	214
<i>Реалізуйте перемикання конфігурацій багатопотокового коду</i> ...	215

Забезпечте логічну ізоляцію конфігурацій багатопотокового коду.....	215
Протестуйте програму з кількістю потоків, що перевищує кількість процесорів	215
Протестуйте програму на різних платформах	216
Застосовуйте інструментування коду для підвищення ймовірності виявлення збоїв	216
Ручне інструментування	216
Автоматизоване інструментування	217
Висновки	218
Література.....	219
РОЗДІЛ 14. ПОСЛІДОВНЕ ОЧИЩЕННЯ	
(Справа про розбирання аргументів командного рядка)	220
Реалізація Args	221
Як я це зробив?.....	227
Args: чернетка	228
На цьому я зупинився	240
Про поступове вдосконалення.....	240
Аргументи String.....	242
Висновки	281
РОЗДІЛ 15. ВНУТРІШНЯ БУДОВА JUNIT	282
Фреймворк JUnit	282
Висновки	297
РОЗДІЛ 16. ПЕРЕРОБЛЕННЯ SERIALDATE.....	298
Перш за все — змусити працювати	299
...Потім очистити код.....	301
Висновки	315
Література.....	315
РОЗДІЛ 17. «ЗАПАХИ» ТА ЕВРИСТИЧНІ ПРАВИЛА.....	316
Коментарі	317
C1: Недоречна інформація	317
C2: Застарілий коментар.....	317
C3: Надмірний коментар	317
C4: Погано написаний коментар	318
C5: Закоментований код.....	318
Робоче середовище.....	318
E1: Збирання складається з декількох етапів.....	318
E2: Тестування складається з декількох етапів	318

Функції.....	319
F1: Забагато аргументів.....	319
F2: Вихідні аргументи.....	319
F3: Прапори в аргументах.....	319
F4: Мертві функції.....	319
Різне.....	319
G1: Кілька мов в одному вихідному файлі.....	319
G2: Очевидна поведінка не реалізована.....	319
G3: Некоректна межова поведінка.....	320
G4: Відімкнені засоби безпеки.....	320
G5: Дублювання.....	320
G6: Код на неправильному рівні абстракції.....	321
G7: Базові класи, залежні від похідних.....	322
G8: Забагато інформації.....	322
G9: Мертвий код.....	323
G10: Вертикальний розподіл.....	323
G11: Непослідовність.....	323
G12: Баласт.....	324
G13: Штучні прив'язки.....	324
G14: Функціональна заздрість.....	324
G15: Аргументи-селектори.....	325
G16: Незрозумілі наміри.....	326
G17: Неправильне розміщення.....	326
G18: Недоречні статичні методи.....	327
G19: Використовуйте пояснювальні змінні.....	327
G20: Імена функцій повинні описувати виконувану операцію.....	328
G21: Розуміння алгоритму.....	328
G22: Перетворення логічних залежностей на фізичні.....	329
G23: Використовуйте поліморфізм замість if/else або switch/case.....	330
G24: Дотримуйте стандартних конвенцій.....	331
G25: Заміняйте «чарівні числа» іменованими константами....	331
G26: Будьте точні.....	332
G27: Структура важливіша за конвенції.....	333
G28: Інкапсулюйте умовні конструкції.....	333
G29: Уникайте негативних умов.....	333
G30: Функції мусять виконувати одну операцію.....	333
G31: Приховані темпоральні прив'язки.....	334
G32: Структура коду має бути обґрунтована.....	335
G33: Інкапсулюйте межові умови.....	336
G34: Функції мають бути написані на одному рівні абстракції...	336
G35: Зберігайте конфігураційні дані на високих рівнях.....	338
G36: Уникайте транзитивних звернень.....	338

Java	339
J1: Використовуйте узагальнені директиви імпорту	339
J2: Не успадковуйте від констант	340
J3: Константи проти перерахувань	341
Імена	342
N1: Використовуйте змістовні імена	342
N2: Вибирайте імена на відповідному рівні абстракції	343
N3: За можливості використовуйте стандартну номенклатуру	344
N4: Недвозначні імена	344
N5: Користуйтеся довгими іменами для довгих зон видимості	345
N6: Уникайте кодування	345
N7: Імена повинні описувати побічні ефекти	345
Тести	346
T1: Брак тестів	346
T2: Використовуйте засоби аналізу покриття коду	346
T3: Не пропускайте тривіальні тести	346
T4: Відімкнений тест як питання	346
T5: Тестуйте межові умови	346
T6: Ретельно тестуйте код поруч із помилками	346
T7: Закономірності збоїв часто несуть корисну інформацію ...	347
T8: Закономірності покриття коду часто несуть корисну інформацію	347
T9: Тести повинні працювати швидко	347
Висновки	347
Література	347

ДОДАТОК А. БАГАТОПОТОКОВІСТЬ II (Бретт Л. Шухерт) 348

Приклад застосування «клієнт/сервер»	348
Сервер	348
Реалізація багатопотоковості	350
Аналіз серверного коду	350
Висновки	352
Можливі шляхи виконання	353
Кількість шляхів	353
Обчислення можливих варіантів упорядкування	354
Копаємо глибше	355
Висновки	358
Знайте свої бібліотеки	358
Executor Framework	358
Неблокові рішення	359
Потоково-небезпечні класи	360

Залежності між методами можуть порушити роботу	
багатопотокового коду	361
Перенесення збоїв.	362
Клієнтське блокування	362
Серверне блокування.	364
Підвищення продуктивності.	365
Обчислення продуктивності в однопотоковій моделі	367
Обчислення продуктивності в багатопотоковій моделі.	367
Взаємне блокування.	368
Взаємне виключення	369
Блокування з очікуванням.	369
Відсутність витіснення	369
Циклічне очікування.	369
Порушення взаємного виключення	370
Порушення блокування з очікуванням.	370
Порушення відсутності витіснення	370
Порушення циклічного очікування	371
Тестування багатопотокового коду	371
Засоби тестування багатопотокового коду	375
Висновки	375
Повні приклади коду	376
Однопотокова реалізація архітектури «клієнт/сервер».	376
Архітектура «клієнт/сервер» з використанням потоків.	379
ДОДАТОК В. ORG.JFREE.DATE.SERIALDATE	381
ДОДАТОК С. ПЕРЕХРЕСНІ ПОСИЛАННЯ	439
ЕПІЛОГ	440
ПОКАЖЧИК	441

РОЗДІЛ 2

ЗМІСТОВНІ ІМЕНА

Тім Оттінгер



Імена трапляються в програмуванні повсюдно. Ми присвоюємо імена своїм змінним, функціям, аргументам, класам і пакетам. Ми присвоюємо імена вихідним файлам і каталогам, у яких вони зберігаються. Ми присвоюємо імена файлам `jar`, `war` та `ear`. Імена, імена, імена... Але те, що роблять так часто, слід робити добре. Далі наводяться деякі прості правила створення хороших імен.

ІМЕНА МАЮТЬ ПЕРЕДАВАТИ НАМІРИ ПРОГРАМІСТА

Легко сказати: імена мають передавати наміри програміста. І все ж до вибору імен слід ставитися *серйозно*. Щоб вибрати гарне ім'я, знадобиться час, але економія окупить витрати. Отже, стежте за іменами у своїх програмах і змінюйте їх, якщо знайдете більш вдалі варіанти. Цим ви спрощуєте життя кожному, хто читатиме ваш код (і собі самому також).

Ім'я змінної, функції або класу має відповідати на всі головні питання. Воно має повідомити, чому ця змінна (або щось інше) існує, що вона робить і як використовується. Якщо ім'я вимагає додаткових коментарів, воно зазвичай не передає намірів програміста.

```
int d; // elapsed time in days
```

Ім'я `d` не передає зовсім нічого. Воно не асоціюється ні з часовими інтервалами, ні з днями. Його слід замінити іншим ім'ям, яке вказує, що саме вимірюють і в яких одиницях:

```
int elapsedTimeInDays;  
int daysSinceCreation;  
int daysSinceModification;  
int fileAgeInDays;
```

Змістовні імена істотно спрощують розуміння і модифікацію коду. Наприклад, що робить нижченаведений фрагмент?

```
public List<int[]> getThem() {  
    List<int[]> list1 = new ArrayList<int[]>();  
    for (int[] x : theList)  
        if (x[0] == 4)  
            list1.add(x);  
    return list1;  
}
```

Чому ми не можемо негайно сказати, що робить цей код? У ньому немає складних виразів. Пробіли й відступи розставлені грамотно. У коді задіяні тільки три змінні й дві константи. У ньому немає жодних хитромудрих класів або поліморфних методів, тільки список масивів (принаймні позір).

Проблема криється не в складності коду, а в його *неочевидності*, тобто в ступені, у якому контекст не походить із власне коду. Код неявно вимагає, щоби ми знали відповіді на такі запитання:

1. Які дані зберігаються в `theList`?
2. Чим так важливий елемент `theList` із нульовим індексом?
3. Який особливий сенс має значення 4?
4. Як будуть використовувати список, що повертається?