

Зміст

Присвята	7
Вступне слово	9
Передмова	11
Подяки	15
Про автора	16

Частина I. ФУНКЦІОНАЛЬНІ ОСНОВИ

1. НЕЗМІННІСТЬ	19
Що таке функціональне програмування?	20
Проблема із присвоюванням	23
То чому ж воно називається функціональним?	25
Без зміни стану?	26
Незмінність	30
2. ПОСТІЙНІ ДАНІ	31
Про шахрайство	33
Виготовлення копій	34
Структурний розподіл	36
3. РЕКУРСІЯ ТА ІТЕРАЦІЯ	39
Ітерація	40
Дуже короткий підручник із Clojure	41
Ітерація	43
TCO, Clojure та JVM	43
Рекурсія	43
4. ЛІНЬ	47
Ліниве накопичення	50
Гаразд, але чому?	50
Кода	52
5. СТАБІЛЬНИЙ СТАН	53
Коли ми мусимо мутувати	57
Програмна транзакційна пам'ять (STM)	58
Життя складне, програмне забезпечення іще складніше	60

Частина II. ПОРІВНЯЛЬНИЙ АНАЛІЗ	
6. «PRIME FACTORS»	63
Версія Java	64
Версія на Clojure	67
Висновок	70
7. «ГРА У БОУЛІНГ»	71
Версія Java	72
Версія на Clojure	76
Висновок	80
8. «ПЛІТКИ ВОДІЙ АВТОБУСІВ»	81
Рішення на Java	82
Driver	87
Route	88
Stop	88
Rumor	89
Simulation	90
Clojure	91
Висновок	95
9. ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ	97
Функціональна реалізація Payroll (Розрахунок заробітної платні)	100
Простори імен та вихідні файли	107
Висновок	108
10. ТИПИ	109
Висновок	114

Частина III. ФУНКЦІОНАЛЬНИЙ ДИЗАЙН	
11. ПОТІК ДАНИХ	117
12. SOLID	125
Принцип єдиної відповідальності (SRP)	126
Принцип відкритості-закритості (OCP)	130
Функції	132
Об'єкти з Vtables	133
Мультиметоди	133
Незалежне розгортання	135
Принцип заміщення Лісков (LSP)	136
Правило ISA	139
Hi!	142
Правило представлення	142

Принцип розділення інтерфейсів (ISP)	143
Не залежте від речей, що вам не потрібні	145
Чому?	146
Висновок	147
Принцип інверсії залежностей (DIP)	147
Вибух із минулого	150
Порушення DIP	159
Висновок	170

Частина IV. ФУНКЦІОНАЛЬНА ПРАГМАТИКА

13. ТЕСТИ	173
А як щодо REPL?	174
А що з моками?	174
Тестування на основі властивостей	175
Метод діагностики	179
Функціональний підхід	186
14. ГРАФІЧНИЙ ІНТЕРФЕЙС	187
Turtle-Graphics у Quil	188
15. КОНКУРЕНТНІСТЬ	199
Висновок	208

Частина V. ДИЗАЙН-ПАТЕРНИ

16. ОГЛЯД ДИЗАЙН-ПАТЕРНІВ	211
Патерни у функціональному програмуванні	214
Абстрактний сервер	215
Адаптер	218
Чи справді це об'єкт адаптера?	221
Command	223
Відміна	225
Composite	229
Функціональний?	233
Декоратор	238
Візитер	241
Close чи Clojure?	244
Проблема повороту на 90 градусів	247
Абстрактна фабрика	250
Знову поворот на 90 градусів	254
Безпека типів?	256
Висновок	257
Постскриптум: отруєне ОП?	257

Частина VI. ПРИКЛАД ІЗ ЖИТТЯ

17. WA-TOR	261
Позбавляємося сверблячки	280
Душ вирішує проблеми	282
Час дикого розмноження	291
А як щодо shark?	293
Висновок	303
Післямова від Джини Мартіні	304
Показчик	307

Передмова

Ця книжка для програмістів «в окопах», які хочуть навчитися використовувати функціональні мови програмування для виконання реальних завдань. Тому я не витрачатиму багато часу на теоретичні аспекти функціонального програмування, як-от монади, моноїди, функції, категорії тощо. Не тому, що ці ідеї не є дієвими, цінними або актуальними; скоріше тому, що вони не часто впливають на повсякденне життя розробника. І відбувається це через те, що вони вже «запечені в пиріг» поширених мов, бібліотек і фреймворків. Якщо вас цікавить функціональна теорія, рекомендую роботи Марка Сімана (*Mark Seemann*).

Ця книжка про те, як і навіщо використовувати функціональне програмування в повсякденній роботі зі створення реальних систем для реальних клієнтів. На наступних сторінках ми будемо порівнювати і зіставляти структури кодування, поширені в об'єктно-орієнтованих мовах, як-от Java, зі структурами, що використовуються у функціональних мовах — як Clojure.

Я обрав ці дві мови, зокрема, тому, що Java дуже широко відома і так само широко вживається, а Clojure надзвичайно проста у вивченні.

Коротка історія функціонального і процедурного програмування

У 1936 році двоє математиків, Алан Тюрінг та Алонзо Черч, незалежно один від одного розв'язали одну з відомих проблем Девіда Гільберта: проблему розв'язності (*The Decidability Problem*). Детальний опис цієї проблеми виходить за рамки цього вступу, за винятком того, що вона була пов'язана з пошуком загального розв'язання для формул цілих чисел¹. Для нас це актуально, оскільки кожна програма у цифровому комп'ютері є цілочисельною формулою.

Вони незалежно довели, що такого загального розв'язання не існує, продемонструвавши, що існують цілі числа, які ніколи не можна обчислити за цілочисельною формулою, меншою за саме ціле число.

¹ Так звані «Діофантові рівняння».

Інакше кажучи, існують числа, які не може обчислити жодна комп'ютерна програма. І дійсно, саме такого підходу дотримувався Алан Тюрінг. У своїй знаменитій статті 1936 року¹ Тюрінг винайшов цифровий комп'ютер, а потім показав, що існують числа, які неможливо обчислити — навіть за умови наявності нескінченного часу і простору².

Черч, з іншого боку, дійшов такого ж висновку завдяки своєму винаходу лямбда-числення, математичного формалізму для маніпулювання функціями. Використовуючи маніпуляції з логікою свого формалізму, він зміг довести, що існують логічні проблеми, які неможливо розв'язати.

Винахід Тюрінга став прабатьком усіх сучасних цифрових комп'ютерів. Кожен цифровий комп'ютер за всіма ознаками є машиною Тюрінга (скінченною). Кожна програма, що коли-небудь виконувалася на цифровому комп'ютері, за всіма ознаками є програмою для машини Тюрінга.

Пізніше Черч і Тюрінг співпрацювали, щоби показати, що їхні підходи еквівалентні. Кожна програма в машині Тюрінга може бути представлена в лямбда-численні — і навпаки.

Функціональне програмування — це, по суті, програмування в лямбда-численні.

Отже, ці два стилі програмування еквівалентні в математичному сенсі. Будь-яка програма може бути написана з використанням як процедурного (Тьюрінг), так і функціонального (Черч) стилю. У цій книжці ми розглянемо не цю еквівалентність, а те, як використання функціонального підходу впливає на структуру та дизайн наших програм. Ми спробуємо визначити, чи є ці різні структури і проекти в певному сенсі кращими або гіршими, ніж ті, що виникають при використанні підходу Тюрінга.

Про Clojure

Я обрав Clojure для цієї книжки тому, що вивчення нової мови і нової парадигми — це вдвічі складніше завдання. Тому я намагався спростити це завдання, обравши мову, яка є достатньо простою, щоби не заважати вивченню функціонального програмування і функціонального дизайну.

Clojure семантично багата, але синтаксично тривіальна. Це означає, що сама мова має дуже простий синтаксис, який не вимагає великих зусиль

¹ Тюрінг А. М. Про обчислювані числа із застосуванням до проблеми *Entscheidungsproblem* (травень 1936).

² Маючи нескінченний час і простір, комп'ютер може обчислити π або e , або будь-яке інше ірраціональне чи трансцендентне число, для якого існує формула. Тюрінг і Черч довели існування чисел, для яких такої формули не існує. Такі числа є «необчислюваними».

для вивчення. Крива навчання Clojure перебуває на семантичній стороні. Бібліотеки та ідіоми вимагають значних зусиль для засвоєння; але сама мова майже не вимагає зусиль. Я сподіваюся, що ця книжка допоможе вам вивчити й оцінити функціональне програмування, не відволікаючись на синтаксис нової мови.

Незважаючи на все це, моя книжка не є навчальним посібником із Clojure¹. Я поясню деякі основи в перших розділах і використаю деякі пояснювальні виноска по всьому тексту, але я також покладаюся на вас, шановний читачу, щоби ви виконували свої домашні завдання і шукали інформацію самостійно. Є кілька вебсайтів, що допоможуть вам у цьому. Наведу один із моїх улюблених: <https://clojure.org/api/cheatsheet>.

Тестовий фреймворк, який я використовував у цій книжці, називається *speclj*². У міру просування по розділах ви будете бачити його все частіше. Він дуже схожий на інші популярні фреймворки для тестування, тому вам буде легко ознайомитися з його різноманітними можливостями.

Про архітектуру та дизайн

Основна мета цієї книжки — описати принципи проектування та архітектури систем, побудованих у функціональному стилі. Із цією метою я буду використовувати діаграми уніфікованої мови моделювання (UML) і посилаюся на принципи проектування програмного забезпечення SOLID³, патерни проектування⁴ та концепцію чистої архітектури. Не хвилюйтеся, я буду пояснювати ці речі в процесі і посилатимуся на багато зовнішніх джерел, якщо вам знадобиться щось знайти.

Про об'єктну орієнтацію

Багато хто вважає, що об'єктно-орієнтоване програмування і функціональне програмування несумісні. Але всі наступні сторінки призначені для того, щоби довести протилежне. Програми, проекти та архітектури, що ви побачите тут, поєднують у собі як функціональні, так і об'єктно-орієнтовані концепції. Моє тверде переконання полягає в тому, що ці два стилі цілком сумісні, і що хороші розробники можуть і повинні застосовувати їх разом.

¹ Ближче до кінця книжки ви вважатимете мене брехуном.

² <https://github.com/slagyr/speclj>

³ Див.: Мартін, Роберт С. Чиста архітектура. С. 57.

⁴ Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software* (Addison-Wesley, 1994).