Зміст

Купити книгу на сайті kniga.biz.ua >>>

Частина II. ПОРІВНЯЛЬНИЙ АНАЛІЗ

Частина III. ФУНКЦІОНАЛЬНИЙ ДИЗАЙН

Купити книгу на сайті kniga.biz.ua >>>

Купити книгу на сайті kniga.biz.ua >>>

Частина VI. ПРИКЛАД ІЗ ЖИТТЯ

Купити книгу на сайті kniga.biz.ua >>>

Передмова

Це 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 чисел[1]. 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».

Купити книгу на сайті kniga.biz.ua >>>

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 року[1] Тюрінг винайшов цифровий комп'ютер, а потім показав, що існують числа, які неможливо обчислити — навіть за умови наявності нескінченного часу і простору[2].

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 проєкти 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 є, 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».

Купити книгу на сайті kniga.biz.ua >>>

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 Clojure[1]. Я поясню деякі основи в перших розділах і використаю деякі пояснювальні виноски 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 проєктування та архітектури систем, побудованих у функціональному стилі. Із цією метою я буду використовувати діаграми уніфікованої мови моделювання (UML) і посилатися на принципи проєктування програмного забезпечення SOLID[3], патерни проєктування[4] 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, проєкти та архітектури, що ви побачите тут, поєднуватимуть 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).

Купити книгу на сайті kniga.biz.ua >>>