
**ПРЕДМЕТНО-
ОРИЕНТИРОВАННЫЕ
ЯЗЫКИ
ПРОГРАММИРОВАНИЯ**

DOMAIN- SPECIFIC LANGUAGES

Martin Fowler
With Rebecca Parsons



ADDISON-WESLEY

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Cape Town • Sydney • Tokyo • Singapore • Mexico City

ПРЕДМЕТНО- ОРИЕНТИРОВАННЫЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Мартин Фаулер
При участии Ребекки Парсонс



Москва • Санкт-Петербург • Киев
2011

ББК 32.973.26-018.2.75

Ф28

УДК 681.3.07

Издательский дом “Вильямс”

Зав. редакцией *С.Н. Тригуб*

Перевод с английского и редакция канд. техн. наук *И.В. Красикова*

Научный консультант докт. физ.-мат. наук *Д.А. Ключин*

По общим вопросам обращайтесь в Издательский дом “Вильямс” по адресу:
info@williamspublishing.com, http://www.williamspublishing.com

Фаулер, Мартин.

Ф28 Предметно-ориентированные языки программирования. : Пер. с англ. — М. :
ООО “И.Д. Вильямс”, 2011. — 576 с. : ил. — Парал. тит. англ.

ISBN 978-5-8459-1738-6 (рус.)

ББК 32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Addison-Wesley Publishing Company, Inc.

Authorized translation from the English language edition published by Addison-Wesley Publishing Company, Inc, Copyright © 2011 by Martin Fowler

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise.

Russian language edition is published by Williams Publishing House according to the Agreement with R&I Enterprises International, Copyright © 2011.

Научно-популярное издание

Мартин Фаулер

Предметно-ориентированные языки программирования

Литературный редактор *Л.Н. Красножон*
Верстка *О.В. Романенко*
Художественный редактор *В.Г. Павлютин*
Корректор *Л.А. Гордиенко*

Подписано в печать 30.06.2011. Формат 70x100/16.

Гарнитура Times. Печать офсетная.

Усл. печ. л. 46,44. Уч.-изд. л. 33,4.

Тираж 1000 экз. Заказ № 0000.

Отпечатано с готовых диапозитивов в ГУП “Типография «Наука»”
199034, Санкт-Петербург, 9-я линия В. О., 12.

ООО “И. Д. Вильямс”, 127055, г. Москва, ул. Лесная, д. 43, стр. 1

ISBN 978-5-8459-1738-6 (рус.)
ISBN 978-0-321-71294-3 (англ.)

© Издательский дом “Вильямс”, 2011
© Martin Fowler, 2011

Оглавление

Предисловие	17
Благодарности	24
Часть I. Описание	27
Глава 1. Вводный пример	29
Глава 2. Использование предметно-ориентированных языков	49
Глава 3. Реализация предметно-ориентированных языков	63
Глава 4. Реализация внутреннего DSL	85
Глава 5. Реализация внешнего DSL	105
Глава 6. Выбор между внутренними и внешними DSL	119
Глава 7. Альтернативные вычислительные модели	127
Глава 8. Генерация кода	135
Глава 9. Языковые инструментальные средства	143
Часть II. Общие вопросы	157
Глава 10. Зоопарк DSL	159
Глава 11. Семантическая модель	171
Глава 12. Таблица символов	177
Глава 13. Переменная контекста	187
Глава 14. Построитель конструкции	191
Глава 15. Макрос	195
Глава 16. Уведомление	205
Часть III. Вопросы создания внешних DSL	211
Глава 17. Трансляция, управляемая разделителями	213
Глава 18. Синтаксически управляемая трансляция	229
Глава 19. Форма Бэкуса–Наура	237
Глава 20. Лексический анализатор на основе таблицы регулярных выражений	247
Глава 21. Синтаксический анализатор на основе рекурсивного спуска	253
Глава 22. Комбинатор синтаксических анализаторов	263
Глава 23. Генератор синтаксических анализаторов	277
Глава 24. Построение дерева	289
Глава 25. Встроенная трансляция	305
Глава 26. Встроенная интерпретация	311
Глава 27. Внешний код	315

Глава 28. Альтернативная токенизация	325
Глава 29. Вложенные операторные выражения	333
Глава 30. Символ новой строки в качестве разделителя	339
Глава 31. Прочие вопросы	343
Часть IV. Вопросы создания внутренних DSL	347
Глава 32. Построитель выражений	349
Глава 33. Последовательность функций	357
Глава 34. Вложенные функции	361
Глава 35. Соединение методов в цепочки	375
Глава 36. Перенос области видимости в объект	387
Глава 37. Замыкание	397
Глава 38. Вложенные замыкания	403
Глава 39. Список литералов	415
Глава 40. Ассоциативные массивы литералов	417
Глава 41. Динамический отклик	423
Глава 42. Аннотации	439
Глава 43. Работа с синтаксическим деревом	449
Глава 44. Класс таблицы символов	461
Глава 45. Шлифовка текста	469
Глава 46. Расширение литералов	473
Часть V. Альтернативные вычислительные модели	477
Глава 47. Адаптивная модель	479
Глава 48. Таблицы принятия решений	485
Глава 49. Сеть зависимостей	495
Глава 50. Система правил вывода	503
Глава 51. Конечный автомат	517
Часть VI. Генерация кода	521
Глава 52. Генерация с помощью преобразователя	523
Глава 53. Шаблонная генерация	529
Глава 54. Встроенный помощник	537
Глава 55. Генерация, осведомленная о модели	545
Глава 56. Генерация, игнорирующая модель	557
Глава 57. Отделение генерируемого кода с помощью наследования	561
Список литературы	569
Предметный указатель	570

Содержание

Предисловие	17
Благодарности	24
Часть I. Описание	27
Глава 1. Вводный пример	29
1.1. Готическая безопасность	29
1.1.1. Контроллер мисс Грант	30
1.2. Модель конечного автомата	31
1.3. Программирование контроллера мисс Грант	34
1.4. Языки и семантическая модель	41
1.5. Использование генерации кода	43
1.6. Использование языковых инструментальных средств	46
1.7. Визуализация	48
Глава 2. Использование предметно-ориентированных языков	49
2.1. Определение предметно-ориентированных языков	49
2.1.1. Границы DSL	51
2.1.2. Фрагментарные и автономные DSL	53
2.2. Зачем используется DSL	54
2.2.1. Повышение производительности разработки	54
2.2.2. Общение с экспертами в предметной области	55
2.2.3. Изменения в контексте выполнения	56
2.2.4. Альтернативные вычислительные модели	57
2.3. Проблемы с DSL	57
2.3.1. Языковая какофония	58
2.3.2. Стоимость построения	58
2.3.3. Язык гетто	59
2.3.4. Ограниченная абстракция	59
2.4. Более широкая обработка языка	60
2.5. Жизненный цикл DSL	60
2.6. Что такое хороший дизайн DSL	62
Глава 3. Реализация предметно-ориентированных языков	63
3.1. Архитектура обработки DSL	63
3.2. Работа синтаксического анализатора	67
3.3. Грамматики, синтаксис и семантики	68
3.4. Анализ данных	69
3.5. Макросы	72
3.6. Тестирование DSL	72
3.6.1. Тестирование семантической модели	73
3.6.2. Тестирование синтаксического анализатора	76
3.6.3. Тестирование сценариев	80

3.7. Обработка ошибок	80
3.8. Миграция DSL	82
Глава 4. Реализация внутреннего DSL	85
4.1. Свободные API и API командных запросов	85
4.2. Необходимость в слое синтаксического анализа	88
4.3. Использование функций	89
4.4. Коллекции литералов	93
4.5. Использование грамматик для выбора внутренних элементов	95
4.6. Замыкания	96
4.7. Работа с синтаксическим деревом	98
4.8. Аннотации	100
4.9. Расширение литералов	101
4.10. Снижение синтаксического шума	101
4.11. Динамический отклик	102
4.12. Проверка типов	103
Глава 5. Реализация внешнего DSL	105
5.1. Стратегия синтаксического анализа	105
5.2. Стратегия получения вывода	108
5.3. Концепции синтаксического анализа	110
5.3.1. Лексический анализ	110
5.3.2. Грамматики и языки	111
5.3.3. Регулярные, контекстно-свободные и контекстно-зависимые грамматики	112
5.3.4. Нисходящий и восходящий синтаксический анализ	114
5.4. Смешивание с другим языком	115
5.5. XML DSL	117
Глава 6. Выбор между внутренними и внешними DSL	119
6.1. Обучение	119
6.2. Стоимость построения	120
6.3. Осведомленность программистов	121
6.4. Общение с экспертами предметной области	121
6.5. Смешивание в базовом языке	121
6.6. Границы строгой выразительности	122
6.7. Настройка времени выполнения	123
6.8. Скатывание в обобщенность	123
6.9. Соединение DSL	124
6.10. Подведение итогов	124
Глава 7. Альтернативные вычислительные модели	127
7.1. Несколько альтернативных моделей	130
7.1.1. Таблицы решений	130
7.1.2. Система правил вывода	130
7.1.3. Конечный автомат	132
7.1.4. Сеть зависимостей	132
7.1.5. Выбор модели	133

Глава 8. Генерация кода	135
8.1. Выбор объекта генерации	136
8.2. Как генерировать код	138
8.3. Смешивание сгенерированного и рукописного кодов	139
8.4. Генерация удобочитаемого кода	140
8.5. Предварительный анализ генерации кода	141
8.6. Источники дополнительной информации	141
Глава 9. Языковые инструментальные средства	143
9.1. Элементы языковых инструментальных средств	143
9.2. Языки определения схем и метамодели	144
9.3. Редактирование исходного текста и проекционное редактирование	149
9.3.1. Множественные представления	151
9.4. Иллюстративное программирование	151
9.5. Тур по инструментам	153
9.6. Языковые инструментальные средства и CASE-инструменты	154
9.7. Следует ли использовать языковые инструментальные средства	155
Часть II. Общие вопросы	157
Глава 10. Зоопарк DSL	159
10.1. Graphviz	159
10.2. JMock	160
10.3. CSS	162
10.4. Hibernate Query Language (HQL)	163
10.5. XAML	164
10.6. FIT	166
10.7. Make и другие	167
Глава 11. Семантическая модель	171
11.1. Как это работает	171
11.2. Когда это использовать	173
11.3. Вводный пример (Java)	174
Глава 12. Таблица символов	177
12.1. Как это работает	177
12.1.1. Статически типизированные символы	179
12.2. Когда это использовать	180
12.3. Дополнительная литература	180
12.4. Сеть зависимостей во внешнем DSL (Java и ANTLR)	180
12.5. Использование символьных ключей во внутреннем DSL (Ruby)	182
12.6. Использование перечислений для статически типизированных символов (Java)	183
Глава 13. Переменная контекста	187
13.1. Как это работает	187
13.2. Когда это использовать	188
13.3. Чтение INI-файла (C#)	188

Глава 14. Построитель конструкции	191
14.1. Как это работает	191
14.2. Когда это использовать	192
14.3. Построение простых полетных данных (C#)	192
Глава 15. Макрос	195
15.1. Как это работает	195
15.1.1. Текстовые макросы	196
15.1.2. Синтаксические макросы	199
15.2. Когда это использовать	202
Глава 16. Уведомление	205
16.1. Как это работает	205
16.2. Когда это использовать	206
16.3. Очень простое уведомление (C#)	206
16.4. Уведомление анализа (Java)	207
Часть III. Вопросы создания внешних DSL	211
Глава 17. Трансляция, управляемая разделителями	213
17.1. Как это работает	213
17.2. Когда это использовать	216
17.3. Карты постоянных клиентов (C#)	216
17.3.1. Семантическая модель	217
17.3.2. Синтаксический анализатор	219
17.4. Синтаксический анализ неавтономных инструкций контроллера мисс Грант (Java)	221
Глава 18. Синтаксически управляемая трансляция	229
18.1. Как это работает	230
18.1.1. Лексический анализатор	231
18.1.2. Синтаксический анализатор	233
18.1.3. Генерация вывода	235
18.1.4. Семантические предикаты	236
18.2. Когда это использовать	236
18.3. Дополнительная литература	236
Глава 19. Форма Бэкуса–Наура	237
19.1. Как это работает	237
19.1.1. Символы множественности (операторы Клини)	239
19.1.2. Другие полезные операторы	240
19.1.3. Грамматики, разбирающие выражения	240
19.1.4. Преобразование РБНФ в БНФ	241
19.1.5. Действия	243
19.2. Когда это использовать	245

Глава 20. Лексический анализатор на основе таблицы регулярных выражений	247
20.1. Как это работает	248
20.2. Когда это использовать	249
20.3. Лексический анализатор контроллера мисс Грант (Java)	249
Глава 21. Синтаксический анализатор на основе рекурсивного спуска	253
21.1. Как это работает	254
21.2. Когда это использовать	257
21.3. Дополнительная литература	257
21.4. Рекурсивный спуск и контроллер мисс Грант (Java)	257
Глава 22. Комбинатор синтаксических анализаторов	263
22.1. Как это работает	264
22.1.1. Выполнение действий	267
22.1.2. Функциональный стиль комбинаторов	268
22.2. Когда это использовать	268
22.3. Комбинаторы синтаксических анализаторов и контроллер мисс Грант (Java)	269
Глава 23. Генератор синтаксических анализаторов	277
23.1. Как это работает	277
23.1.1. Встроенные действия	278
23.2. Когда это использовать	280
23.3. Hello World (Java и ANTLR)	280
23.3.1. Написание базовой грамматики	281
23.3.2. Построение синтаксического анализатора	282
23.3.3. Добавление кода действий в грамматику	284
23.3.4. Применение шаблона Generation Gap	286
Глава 24. Построение дерева	289
24.1. Как это работает	289
24.2. Когда это использовать	291
24.3. Использование синтаксиса построения дерева ANTLR (Java и ANTLR)	292
24.3.1. Токенизация	293
24.3.2. Синтаксический анализ	294
24.3.3. Наполнение семантической модели	296
24.4. Построение дерева с использованием кода действий (Java и ANTLR)	299
Глава 25. Встроенная трансляция	305
25.1. Как это работает	305
25.2. Когда это использовать	306
25.3. Контроллер мисс Грант (Java и ANTLR)	306
Глава 26. Встроенная интерпретация	311
26.1. Как это работает	311
26.2. Когда это использовать	311
26.3. Калькулятор (ANTLR и Java)	312

Глава 27. Внешний код	315
27.1. Как это работает	315
27.2. Когда это использовать	317
27.3. Встраивание динамического кода (ANTLR, Java и Javascript)	317
27.3.1. Семантическая модель	318
27.3.2. Синтаксический анализатор	320
Глава 28. Альтернативная токенизация	325
28.1. Как это работает	325
28.1.1. Кавычки	326
28.1.2. Лексическое состояние	328
28.1.3. Изменение типа токена	330
28.1.4. Игнорирование типов токенов	331
28.2. Когда это использовать	332
Глава 29. Вложенные операторные выражения	333
29.1. Как это работает	333
29.1.1. Применение восходящих синтаксических анализаторов	334
29.1.2. Нисходящие синтаксические анализаторы	335
29.2. Когда это использовать	337
Глава 30. Символ новой строки в качестве разделителя	339
30.1. Как это работает	339
30.2. Когда это использовать	341
Глава 31. Прочие вопросы	343
31.1. Синтаксические отступы	343
31.2. Модулярная грамматика	345
Часть IV. Вопросы создания внутренних DSL	347
Глава 32. Построитель выражений	349
32.1. Как это работает	350
32.2. Когда это использовать	350
32.3. Свободный календарь с строителем и без него (Java)	351
32.4. Использование для календаря нескольких строителей (Java)	353
Глава 33. Последовательность функций	357
33.1. Как это работает	357
33.2. Когда это использовать	358
33.3. Простая конфигурация компьютера (Java)	358
Глава 34. Вложенные функции	361
34.1. Как это работает	361
34.2. Когда это использовать	363
34.3. Простой пример конфигурации компьютера (Java)	363
34.4. Обработка различных аргументов с помощью токенов (C#)	365

34.5. Использование токенов подтипов для поддержки интегрированной среды разработки (Java)	366
34.6. Использование инициализаторов объектов (C#)	368
34.7. Повторяющиеся события (C#)	369
34.7.1. Семантическая модель	369
34.7.2. DSL	372
Глава 35. Соединение методов в цепочки	375
35.1. Как это работает	375
35.1.1. Построители или значения	377
35.1.2. Проблема окончания	377
35.1.3. Иерархическая структура	378
35.1.4. Последовательные интерфейсы	378
35.2. Когда это использовать	379
35.3. Простой пример конфигурации компьютера (Java)	379
35.4. Соединение методов в цепочки со свойствами (C#)	383
35.5. Последовательные интерфейсы (C#)	383
Глава 36. Перенос области видимости в объект	387
36.1. Как это работает	388
36.2. Когда это использовать	388
36.3. Коды безопасности (C#)	389
36.3.1. Семантическая модель	389
36.3.2. DSL	391
36.4. Использование вычисления экземпляра (Ruby)	393
36.5. Использование инициализатора экземпляра (Java)	395
Глава 37. Замыкание	397
37.1. Как это работает	397
37.2. Когда это использовать	401
Глава 38. Вложенные замыкания	403
38.1. Как это работает	403
38.2. Когда это использовать	405
38.3. Заворачивание последовательности функций во вложенное замыкание (Ruby)	405
38.4. Простой пример (C#)	407
38.5. Применение соединения методов в цепочки (Ruby)	408
38.6. Последовательность функций с явными аргументами замыканий (Ruby)	410
38.7. Применение вычисления экземпляра (Ruby)	411
Глава 39. Список литералов	415
39.1. Как это работает	415
39.2. Когда это использовать	415
Глава 40. Ассоциативные массивы литералов	417
40.1. Как это работает	417
40.2. Когда это использовать	418

14 Содержание

40.3. Настройка конфигурации компьютера с помощью списков и отображений (Ruby)	418
40.4. Имитация Lisp (Ruby)	419
Глава 41. Динамический отклик	423
41.1. Как это работает	424
41.2. Когда это использовать	424
41.3. Расчет бонусов с помощью анализа имен методов (Ruby)	426
41.3.1. Модель	426
41.3.2. Построитель	428
41.4. Расчет бонусов с помощью цепочек вызовов (Ruby)	429
41.4.1. Модель	430
41.4.2. Построитель	430
41.5. Уменьшение шума в коде контроллера тайника (JRuby)	433
Глава 42. Аннотации	439
42.1. Как это работает	440
42.1.1. Определение аннотации	440
42.1.2. Обработка аннотаций	441
42.2. Когда это использовать	442
42.3. Пользовательский синтаксис с обработкой времени выполнения (Java)	443
42.4. Использование метода класса (Ruby)	445
42.5. Динамическая генерация кода (Ruby)	446
Глава 43. Работа с синтаксическим деревом	449
43.1. Как это работает	449
43.2. Когда это использовать	450
43.3. Генерация запросов IMAP из условий C# (C#)	451
43.3.1. Семантическая модель	452
43.3.2. Построение из исходного текста C#	454
43.3.3. Отступление	458
Глава 44. Класс таблицы символов	461
44.1. Как это работает	462
44.2. Когда это использовать	462
44.3. Статически типизированный класс таблицы символов (Java)	463
Глава 45. Шлифовка текста	469
45.1. Как это работает	469
45.2. Когда это использовать	470
45.3. Шлифовка правил дисконта (Ruby)	470
Глава 46. Расширение литералов	473
46.1. Как это работает	473
46.2. Когда это использовать	474
46.3. Ингредиенты рецепта (C#)	474

Часть V. Альтернативные вычислительные модели	477
Глава 47. Адаптивная модель	479
47.1. Как это работает	480
47.1.1. Внедрение императивного кода в адаптивную модель	481
47.1.2. Инструментарий	483
47.2. Когда это использовать	483
Глава 48. Таблицы принятия решений	485
48.1. Как это работает	485
48.2. Когда это использовать	487
48.3. Вычисление оплаты заказа (C#)	487
48.3.1. Модель	487
48.3.2. Синтаксический анализатор	491
Глава 49. Сеть зависимостей	495
49.1. Как это работает	496
49.2. Когда это использовать	498
49.3. Анализ зелий (C#)	498
49.3.1. Семантическая модель	499
49.3.2. Синтаксический анализатор	500
Глава 50. Система правил вывода	503
50.1. Как это работает	504
50.1.1. Цепочки выводов	505
50.1.2. Противоречивые выводы	505
50.1.3. Шаблоны в структуре правила	506
50.2. Когда это использовать	507
50.3. Проверка для членства в клубе (C#)	507
50.3.1. Модель	507
50.3.2. Синтаксический анализатор	508
50.3.3. Развитие DSL	509
50.4. Правила избрания: расширение членства в клубе (C#)	511
50.4.1. Модель	512
50.4.2. Синтаксический анализатор	514
Глава 51. Конечный автомат	517
51.1. Как это работает	517
51.2. Когда это использовать	519
51.3. Контроллер тайника (Java)	519
Часть VI. Генерация кода	521
Глава 52. Генерация с помощью преобразователя	523
52.1. Как это работает	523
52.2. Когда это использовать	524
52.3. Контроллер тайника (Java генерирует C)	525

Глава 53. Шаблонная генерация	529
53.1. Как это работает	529
53.2. Когда это использовать	531
53.3. Генерация конечного автомата тайника с помощью вложенных условных конструкций (Velocity и Java, генерирующие C)	531
Глава 54. Встроенный помощник	537
54.1. Как это работает	538
54.2. Когда это использовать	538
54.3. Состояния тайника (Java и ANTLR)	539
54.4. Должен ли помощник генерировать HTML (Java и Velocity)	541
Глава 55. Генерация, осведомленная о модели	545
55.1. Как это работает	546
55.2. Когда это использовать	546
55.3. Конечный автомат тайника (C)	546
55.4. Динамическая загрузка конечного автомата (C)	553
Глава 56. Генерация, игнорирующая модель	557
56.1. Как это работает	557
56.2. Когда это использовать	558
56.3. Конечный автомат тайника как вложенные условные конструкции (C)	558
Глава 57. Отделение генерируемого кода с помощью наследования	561
57.1. Как это работает	562
57.2. Когда это использовать	563
57.3. Генерация классов из схемы данных (Java и немного Ruby)	563
Список литературы	569
Предметный указатель	570

Предисловие

Предметно-ориентированные языки (Domain-Specific Languages — DSL) были частью компьютерного мира еще до того, как я научился программировать. Спросите ветеранов Unix или Lisp, и они будут счастливы утомить вас до судорог рассказами о том, как предметно-ориентированные языки помогли им в их работе и какие трюки с их помощью они ухитрились выполнять. Тем не менее этим языкам не суждено было стать заметной частью упомянутого компьютерного мира. Большинство людей знают о предметно-ориентированных языках только по чьим-то рассказам, и часто эти рассказы ограничиваются описанием только одной из сторон имеющихся методов.

Я написал эту книгу в надежде изменить ситуацию и предоставить вам как можно больше информации о методах предметно-ориентированных языков, чтобы вы могли принять осознанное решение об их использовании в своей работе и о том, какие именно методы предметно-ориентированных языков применять.

Предметно-ориентированные языки популярны по нескольким причинам, но я остановлюсь только на двух из них: повышение производительности труда разработчиков и улучшение связи с экспертами в предметной области. Правильно выбранный язык может сделать сложный блок кода существенно проще для понимания, что повышает производительность работающих с ним. Он также упрощает общение разработчика программного обеспечения со специалистами в предметной области, по сути предоставляя текст, который одновременно действует и как выполняемое программное обеспечение, и как описание проблемы, которое узкие специалисты в данной области знаний могут прочесть, чтобы понять, как их идеи представлены в программной системе. Трудно переоценить возможность говорить со специалистами на одном языке — эта возможность разрушает самый высокий из барьеров на пути общения между программистами и их клиентами и устраняет массу узких мест в разработке специализированного программного обеспечения.

Однако не хотелось бы и преувеличивать значение предметно-ориентированных языков. Я часто говорю, что всякий раз, рассматривая преимущества применения предметно-ориентированного языка для решения той или иной проблемы, вы должны выполнить еще одно рассмотрение, подставив на этот раз вместо слов “предметно-ориентированный язык” слово “библиотека”. Многое из того, что вы получаете с помощью предметно-ориентированного языка, можно получить, создав соответствующую программную структуру. Большинство предметно-ориентированных языков на самом деле представляют собой просто косметический фасад над библиотеками или программной структурой. В результате выгоды от предметно-ориентированных языков часто оказываются меньшими, чем думают неискушенные в этих вопросах люди; однако в любом

случае для принятия решения нужно четко понимать, какие именно положительные и отрицательные стороны имеет каждое из решений. Знание хорошо себя зарекомендовавших технологий существенно снижает стоимость построения предметно-ориентированного языка (и я надеюсь, что моя книга поможет вам получить эти знания). Фасад может быть декоративной деталью, но часто он так же полезен, как и само здание.

Почему сейчас ?

Предметно-ориентированные языки известны издавна, но в последние годы они породили значительный всплеск интереса. И именно в это время я решил потратить пару лет на эту книгу. Я не уверен, что смогу пояснить, чем вызван такой интерес к предметно-ориентированным языкам, но я поделюсь своей личной точкой зрения.

На рубеже нового тысячелетия возникло ощущение превосходящей разумные рамки стандартизации в языках программирования, по крайней мере в моем мире корпоративного программного обеспечения. Несколько лет Java рассматривался как Единственный Язык Будущего, и даже когда Microsoft выпустила свой C#, он был очень похож на Java. В новых разработках доминирующую позицию занимали компилируемые статические объектно-ориентированные языки программирования с C-образным синтаксисом (даже Visual Basic не избежал попыток приведения его к этому общему знаменателю).

Но вскоре стало ясно, что не все в порядке в королевстве Java/C#. В наличии имелась логика, не хотевшая укладываться в ложе этих языков программирования, что привело к возникновению конфигурационных файлов в формате XML. Программисты шутили, что в результате они пишут больше строк XML, чем на Java/C#. Отчасти это было связано с желанием изменить поведение программы во время выполнения, а отчасти — с желанием выразить аспекты поведения более дружественным и простым для пользователя способом. XML, несмотря на его “зашумленный” синтаксис, позволяет определять собственный словарь и предоставляет строгую иерархическую структуру.

Но “зашумленность” XML все же оказалась слишком большой. Люди жаловались на то, что от угловых скобок у них рябит в глазах, и возникло желание получить преимущества XML-файлов конфигурации без затрат, связанных с применением XML.

Наш рассказ постепенно достиг взрывоподобного явления Ruby On Rails. Независимо от области применения этот язык оказывал огромное влияние на работу программистов с библиотеками и структурными схемами. Большая часть методов работы сообщества Ruby представляет собой более гибкий подход, при котором взаимодействие с библиотекой должно было походить на программирование на специализированном языке. Этот способ мышления восходит к одному из старейших языков программирования — Lisp. При этом подходе и способе мышления можно увидеть цветение даже на каменистой почве Java/C#: в обоих языках все более популярными становятся интерфейсы, вероятно, из-за влияния таких продуктов, как JMock и Hamcrest.

Узнав обо всем этом, я почувствовал, что у меня имеется немалый пробел в знаниях. Я видел людей, использующих XML там, где более удобным и ничуть не более трудным было бы применение иного синтаксиса. Я видел людей, пытавшихся загнать Ruby в тесные рамки сложных выражений, в которых гораздо легче было бы применить пользовательский синтаксис. Я видел программистов, играющихся с синтаксическими анализаторами там, где можно было бы обойтись существенно меньшим количеством работы с гибкими интерфейсами в их обычных языках программирования.

Мне кажется, что причина всех этих перегибов — в недостатке знаний. Квалифицированные программисты недостаточно знакомы с технологиями предметно-ориентирован-

ных языков, чтобы принять обоснованное решение о том, какие из них использовать. Этот пробел в знаниях я бы и хотел восполнить.

Значение предметно-ориентированных языков

Более подробно на эту тему мы поговорим в главе 2, “Использование предметно-ориентированных языков”, а пока что я вижу две основные причины, по которым вы должны быть заинтересованы в предметно-ориентированных языках (а значит, и в методах, описанных в этой книге).

Первая причина заключается в повышении производительности труда программиста. Рассмотрим фрагмент кода

```
input =~ /\d{3}-\d{3}-\d{4}/
```

Вы можете распознать в нем регулярное выражение, и, возможно, вы знаете, что оно означает. Регулярные выражения часто критикуют за излишнюю загадочность, но представьте, как бы выглядел обычный код сопоставления шаблону без регулярных выражений. Насколько легко было бы понять и модифицировать такой код в сравнении с регулярным выражением?

Предметно-ориентированные языки позволяют сделать некоторые частные задачи программирования более легкими для понимания, а значит, соответствующие программы можно будет быстрее писать, легче изменять, и они будут менее подвержены ошибкам.

Вторая причина заинтересованности в предметно-ориентированных языках выходит за рамки программирования. Поскольку такие языки меньше и их легче понимать, они позволяют специалистам в предметной области, не являющимся программистами, понимать код, управляющий важными аспектами их профессии. Анализ реального кода со специалистами в предметной области позволит значительно обогатить и сделать более полезным общение между программистами и их клиентами.

Обсуждая подобные темы, люди часто говорят, что предметно-ориентированные языки позволят полностью избавиться от программистов. Я очень скептически отношусь к этому аргументу; в конце концов, то же самое в свое время говорили и о COBOL. Хотя, конечно, есть языки (такие, как CSS), написанные людьми, которые не называют себя программистами. Однако для предметно-ориентированных языков чтение написанного кода имеет большее значение, чем написание. Если эксперты в предметной области читают и в основном понимают написанный программистами код, то им будет гораздо легче общаться с программистом, который этот код создавал.

Вторая причина использования предметно-ориентированных языков не так проста, но награда стоит затраченных усилий. Общение между программистами и их клиентами — самое узкое место в разработке программного обеспечения, так что все, что может облегчить такое общение, стоит затраченных усилий.

Не бойтесь объема этой книги

Толщина этой книги может немного испугать. Я сам всегда настороженно отношусь к толстым книгам, потому что у всех нас не так уж много времени, которое мы можем посвятить изучению чего-то нового (и которое гораздо важнее при выборе книги, чем ее цена). Поэтому я воспользовался форматом, который предпочитаю в таких случаях — двойной книги.

Двойная книга — это две книги под одной обложкой. Первая — обычная повествовательная книга, предназначенная для чтения от корки до корки. Моя цель в этой книге — предоставить краткий обзор темы, достаточный для понимания, но не для практической работы. Объем этой части — не более пары сотен страниц, что вполне можно позволить себе прочесть полностью.

Вторая (большая по объему) книга представляет собой справочный материал, который предназначен не для подробного чтения (хотя некоторые люди так и делают), а для того, чтобы при необходимости обратиться к ней, как к справочнику. Одни читатели полностью прорабатывают первую книгу, чтобы получить общее представление о теме, а затем обращаются ко второй части через предметный указатель — в поисках только того материала, который их интересует. Другие читают справочный раздел так же, как и описательный. При разделении книги на две я хотел помочь вам в выборе материала, который можно пропустить и в который следует погрузиться как следует. Решение за вами.

Я также попытался сделать справочную часть в достаточной степени самостоятельной, так что если вы захотите узнать, например, о построении деревьев, то можете прочесть только соответствующую часть книги и получить вполне приличное представление о том, что следует делать, даже если вы подзабыли материал первой части. Таким образом, как только вы прочтете первую часть книги, она станет справочником, что очень удобно при практической работе, когда вам нужно найти информацию о конкретных деталях.

Основная причина, по которой книга получилась такой большой, — я просто не придумал, как сделать ее покороче. Одна из главных моих целей в этой книге заключается в предоставлении читателю обзора большого количества различных методов предметно-ориентированных языков. Есть книги о генерации кода, о метапрограммировании в Ruby и об использовании генераторов анализаторов. Я же стремился охватить все эти методы, чтобы вы могли лучше понять их сходства и различия. Все они играют определенную роль в более широком ландшафте, и моя цель — так организовать путешествие по этому ландшафту, чтобы вы ознакомились не только с отдельными пейзажами, но и с каждой из его частей достаточно подробно, чтобы иметь возможность тут же начать работу с описанными методами.

О чем вы узнаете из этой книги

Я задумал эту книгу как руководство по различным видам предметно-ориентированных языков и подходам к их построению. Часто, начиная экспериментировать с предметно-ориентированными языками, люди используют какую-то одну технологию. Смысл этой книги — показать вам широкий спектр методов, чтобы вы могли оценить, какие из них лучше всего подходят в ваших обстоятельствах. Я представил подробную информацию и примеры реализации многих из этих методов. Естественно, я не могу показать вам все, что можно сделать, но для принятия первоначальных решений этого должно быть достаточно.

Из первых глав вы получите представление о том, что такое предметно-ориентированные языки, когда они могут пригодиться и какова их роль в сравнении с библиотеками. Вы узнаете, с чего начать при построении внешних и внутренних предметно-ориентированных языков. В главах о внешних предметно-ориентированных языках рассказывается о роли синтаксического анализатора, полезности генератора анализаторов и о способах применения анализаторов для синтаксического анализа внешнего предметно-ориентированного языка. В главах, посвященных внутренним предметно-ориентированным языкам, показано, как использовать языковые конструкции в стиле таких язы-