

Б. Маклафлин Г. Поллайс Д. Уэст

# ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ АНАЛИЗ И ПРОЕКТИРОВАНИЕ



Научитесь передавать  
свои мысли при помощи UML  
и сценариев использования



Поломайте голову  
над десятками  
ОО-упражнений



Не портите настроение  
своим заказчикам

Преобразуйте  
требования  
и планы  
в серьезные  
программы



Загрузите принципы  
ОО-проектирования  
прямо в свой мозг



Узнайте, как абстракция,  
агрегирование и делегирование  
помогли Мэри стать душой  
общества в Объектвиле

# ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ АНАЛИЗ И ПРОЕКТИРОВАНИЕ

Как было бы здорово изучить  
JavaScript, не испытывая желания  
бросить все на половине пути и никогда  
больше не заходить в Интернет!  
Наверное, об этом можно  
только мечтать...

Б. Маклафлин  
Г. Поллайс  
Д. Уэст



 **ПИТЕР®**

Москва • Санкт-Петербург • Нижний Новгород • Воронеж  
Ростов-на-Дону • Екатеринбург • Самара • Новосибирск  
Киев • Харьков • Минск

**2013**

ББК 32.973.2-018.1  
УДК 004.43  
М15

**Маклафлин Б., Поллайс Г., Уэст Д.**

М15 Объектно-ориентированный анализ и проектирование. — СПб.: Питер, 2013. — 608 с.: ил.

ISBN 978-5-496-00144-1

Надоело читать книги по объектно-ориентированному анализу и проектированию, которые понятны только специалистам? Возможно, вы слышали, что ООАП помогает писать хорошие программы — программы, которыми будет довольно ваше начальство и заказчики. Но как это сделать?

Книга покажет вам, как организованы анализ, проектирование и написание серьезных объектно-ориентированных программ; программ, которые просты в повторном использовании, сопровождении и расширении; программ, от которых не болит голова; программ, в которые можно добавлять новые возможности, не нарушая работу старых.

Вы узнаете, какое место занимают ОО-принципы, паттерны проектирования и различные методы разработки в жизненном цикле ООАП-проектов. За счет использования особенностей работы мозга эта книга сокращает время усвоения и запоминания сложной информации. К тому времени, когда будет перевернута последняя страница, вы повеселитесь, узнаете много нового и научитесь писать хорошие программы!

ББК 32.973.2-018.1  
УДК 004.43

Права на издание получены по соглашению с O'Reilly. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

ISBN 978-0596008673 англ.

ISBN 978-5-496-00144-1

© Authorized Russian translation of the English edition of titled Head First Object-Oriented Analysis and Design, 1st Edition (ISBN 9780596008673) © 2007, Brett D. McLaughlin, Gary Pollice, and David West. This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.  
© Перевод на русский язык ООО Издательство «Питер», 2013  
© Издание на русском языке, оформление ООО Издательство «Питер», 2013

Содержание (сводка)

	Введение	23
1	С чего начинаются хорошие программы. <i>О пользе качественного проектирования</i>	35
2	Дайте им то, что они хотят. <i>Сбор требований</i>	87
3	Я тебя люблю, ты мой идеал... Теперь изменись. <i>Изменение требований</i>	141
4	Программы для реального мира. <i>Анализ</i>	173
5	Часть 1. Все течет, все меняется. <i>Качественное проектирование</i>	225
	Вставка: ОО-КАТАСТРОФА	249
	Часть 2. Зарядка для программ. <i>Гибкость программы</i>	261
6	«Меня зовут Арт... И я архитектор». <i>Решение больших задач</i>	305
7	Навести порядок в хаосе. <i>Архитектура</i>	347
8	Не стремитесь к оригинальности. <i>Принципы проектирования</i>	397
9	Программы пишутся для заказчика. <i>Итерации и тестирование</i>	443
10	Все вместе. <i>Жизненный цикл ООАП</i>	501
	Приложение I. <i>Остатки</i>	571
	Приложение II. <i>Добро пожаловать в Объективль</i>	589

Содержание (настоящее)

Введение

**Ваш мозг думает об ООАП.** Вы сидите за книгой и пытаетесь что-нибудь выучить, но ваш мозг считает, что вся эта писанина не нужна. Ваш мозг говорит: «Выгляни в окно! На свете есть более важные вещи, например сноуборд». Как заставить мозг думать, что ваша жизнь действительно зависит от объектно-ориентированного анализа и проектирования?

Для кого написана эта книга?	24
Мы знаем, о чем вы думаете	25
Метапознание: наука о мышлении	27
Вот что сделали МЫ	28
Что можете сделать ВЫ	29
Примите к сведению	30
Благодарности	33

О пользе качественного проектирования

1

С чего начинаются хорошие программы

**Так как же на самом деле пишутся хорошие программы?** Всегда сложно понять, с чего следует начинать. Делает ли приложение то, что ему положено делать? И как насчет таких тонкостей, как дублирование кода, — ведь это всегда плохо, верно? Да, определить, **над чем следует работать в первую очередь**, бывает нелегко, к тому же еще нужно не запороть все остальное по ходу дела. Но пусть вас это не тревожит! К тому моменту, когда вы дочитаете эту главу, вы **будете знать, как пишутся хорошие программы**, а ваш подход к разработке приложений навсегда изменится. И наконец-то вы поймете, почему ООАП — это такое слово, которое не стыдно произносить в приличном обществе.

Откуда я знаю, с чего начинать? В каждом новом проекте, над которым я работаю, у всех разные мнения по поводу того, что делать в первую очередь. Иногда я угадываю верно, иногда приходится переделывать все приложение из-за того, что я начал не с того. А я просто хочу писать хорошие программы! Так с чего нужно начать в приложении Рика?

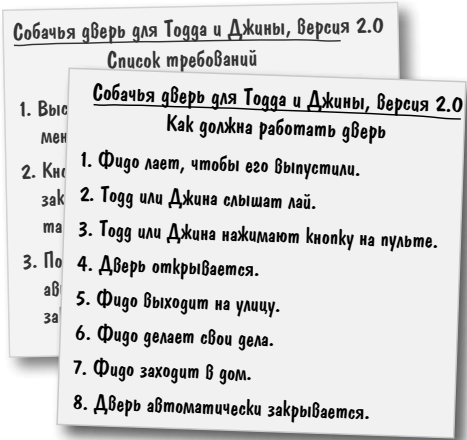


Рок-н-ролл навсегда!	36
Новенькое приложение Рика...	37
Что бы вы изменили В ПЕРВУЮ ОЧЕРЕДЬ?	42
Хорошая программа... сочетание нескольких факторов	46
Хорошая программа за 3 простых шага	47
Пробный запуск	57
Ищем проблемы	59
Анализируем метод search()	60
Теперь обновите свой код	64
Обновление класса Inventory	66
Готовимся к следующему пробному запуску	67
Возвращаемся к приложению Рика...	69
Продолжаем совершенствовать структуру кода	70
Убедимся в том, что класс Inventory хорошо спроектирован	71
Последнее тестирование (проверка готовности кода к повторному использованию)	80
Цель ООАП — написание хороших программ, а не формализация рабочего процесса!	83
Ключевые моменты	64

# 2 Сбор требований

## Дайте им то, что они хотят

**Хорошо, когда заказчик доволен.** Вы ведь уже знаете, что написание хороших программ начинается с того, что вы проверяете, работает ли программа так, как хочет заказчик. Но как узнать, чего на самом деле хочет заказчик? И как убедиться в том, что заказчик вообще знает, чего он хочет? На помощь приходят хорошо составленные требования. В этой главе вы узнаете, как выполнить требования заказчика и предоставить ему именно то, чего он хотел. После этого на всех ваших проектах можно будет ставить «знак качества», а вы сделаете еще один большой шаг к заветной цели — написанию хороших программ... раз за разом.



Вам поручена новая задача	88
Пробный запуск	91
Создание списка требований	96
Планирование возможных неудач	100
Альтернативные пути для решения проблем	102
(Снова) о вариантах использования	104
Один вариант, три части	106
Проверка требований по варианту использования	110
Все ли на месте?	110
Ну теперь-то можно заняться написанием кода?	113
Автоматическое закрывание двери	114
Нужен новый тест!	115
Пробный запуск, версия 2.0	116
Анализ альтернативного пути	118
Пробный запуск, версия 2.1	121
Приложение работает, заказчики довольны	123
Инструментарий ООАП	138

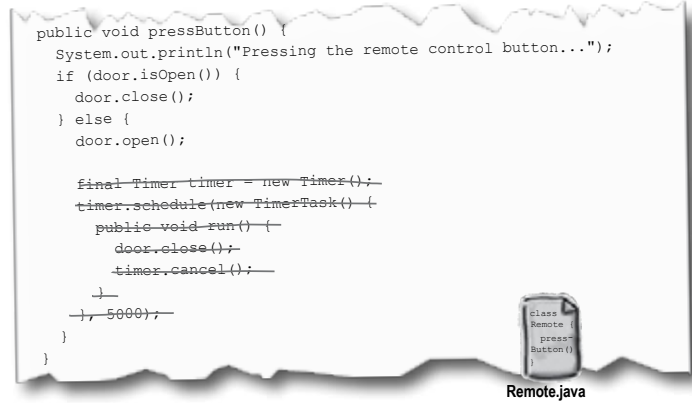


# 3 Изменение требований

## Я тебя люблю, ты мой идеал... Теперь изменись

**Думаете, вы сделали то, что нужно заказчику? Как бы не так...** Вы поговорили с заказчиком, собрали требования, записали варианты использования и выдали убойное приложение. Можно расслабиться, верно? Верно... Пока заказчик не решит, что ему нужно **что-то отличное от того, о чем он говорил вам**. То, что вы сделали, ему понравилось, честно, но **теперь это не совсем то, что нужно**. В реальном мире **требования всегда изменяются**; вы должны адаптироваться к этим изменениям и сделать так, чтобы заказчик был доволен.

Вы — герой!	142
А потом был телефонный звонок...	142
Снова беремся за карандаш	144
Единственный постоянный фактор в области анализа и проектирования программного обеспечения	145
Варианты использования должны быть понятными для вас	152
Готовимся к написанию кода...	159
Доработка списка требований	160
А теперь можно снова заняться программированием	161
Кто сказал «гав»?	162
Проверка новой двери	164
Обновление класса двери	169
Упрощение класса пульта	169
Последний пробный запуск	170
Новые инструменты ООАП	172





## Анализ

## 4

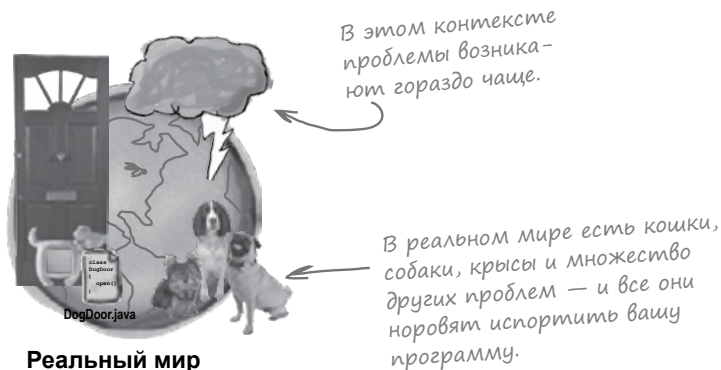
## Программы для реального мира

**Пора переходить к реальным приложениям.** Даже если ваше приложение идеально работает на машине разработки, этого недостаточно: приложения должны работать тогда, когда их используют реальные люди. В этой главе говорится о том, как заставить вашу программу работать в реальном контексте. Вы узнаете, как текстовый анализ преобразовывает вариант использования, над которым вы работали, в классы и методы, которые делают то, что нужно вашим заказчикам. А когда работа будет завершена, вы сможете сказать: «Да, у меня получилось! Моя программа готова к жизни в реальном мире!»

Потом, когда я поделилась с классами и операциями, я обновила свою диаграмму классов.



Одна собака, две собаки, три...	174
Программа работает в определенном контексте	175
Выявление проблемы	176
Планирование решения	177
История двух программистов	184
Делегирование в версии Сэма	188
Сила приложений с низкой связностью	190
Обращайте внимание на существительные в варианте использования	195
От качественного анализа к хорошим классам...	208
Подробнее о диаграммах классов	210
Диаграммы классов — это еще не все	215
И как теперь работает метод recognize()?	217





5 (часть I)

Качественное проектирование = Гибкость программного продукта

Все течет, все меняется

**Изменения неизбежны.** Какой бы замечательной ни была ваша программа сегодня, завтра ее, скорее всего, придется изменять. И чем труднее вносить поправки в программу, тем сложнее реагировать на изменяющиеся потребности заказчика. В этой главе мы вернемся к старому знакомому, попробуем улучшить существующую программу и увидим, как маленькие изменения создают большие трудности. А при этом мы обнаружим проблему настолько значительную, что для ее решения потребуется глава ИЗ ДВУХ ЧАСТЕЙ!

Струнные инструменты	226
Давайте проверим структуру кода на прочность	226
Вы обратили внимание на абстрактный базовый класс?	229
Как устроены диаграммы классов (снова)	234
Напишем код новой поисковой программы Рика	236
Создание абстрактного класса для спецификаций инструментов	237
Завершение работы над поисковой программой	240

5 (вставка)

00-КАТАСТРОФА!  
Любимая телевикторина Объективля

Исключение риска	Знаменитые проектировщики	Программные конструкции	Сопровождение и повторное использование	Программные невроты
\$100	\$100	\$100	\$100	\$100
\$200	\$200	\$200	\$200	\$200
\$300	\$300	\$300	\$300	\$300
\$400	\$400	\$400	\$400	\$400

# 5

(часть 2)

Хорошая структура = Гибкость программы

## Зарядка для программ

**А вам когда-нибудь хотелось обрести большую гибкость?**

Если внесение изменений в приложение создает проблемы, скорее всего, это означает, что вашей программе не хватает гибкости и пластичности. Чтобы помочь приложению прийти в хорошую форму, необходимо провести анализ, основательно проработать структуру кода приложения и узнать, как ОО-принципы способствуют ослаблению связности. А в завершение вы увидите, как высокое сцепление способствует ослаблению связности. Звучит заманчиво? Переверните страницу, и мы продолжим наводить порядок в нашем негибком приложении.

Возвращаемся к приложению Рика	262
Присмотримся повнимательнее к методу search()	265
Результаты анализа	266
Проблемы с классами инструментов	269
Но ведь классы создаются ради поведения!	269
Смерть проектировочных решений	274
Субклассы конкретных инструментов	274
Преобразуем плохие структурные решения в хорошие	275
«Двойная инкапсуляция» в программе Рика	277
Динамические свойства инструментов	278
Смотрите: гибкое приложение Рика!	286
А приложение действительно работает?	287
Пробный запуск улучшенной программы Рика	289
Высокое сцепление и одна причина для изменения	298
Когда следует сказать: «Достаточно!»	303
Инструментарий ООАП	304

# 6

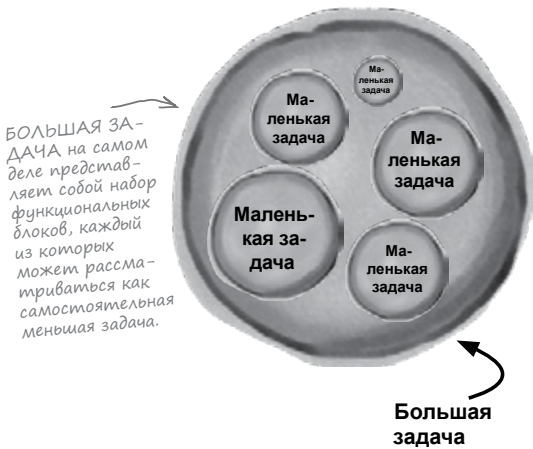
## Решение больших задач

### «Меня зовут Арт... И я архитектор»

**Пора построить нечто ДЕЙСТВИТЕЛЬНО БОЛЬШОЕ. Готовы?**

В вашем инструментарии ООАП скопилось множество инструментов, но как использовать эти инструменты при построении большого проекта? Возможно, вы этого и не осознали, но у вас имеется все необходимое для решения больших задач. В этой главе мы рассмотрим новые инструменты — анализ предметной области и диаграммы вариантов использования. Но даже эти инструменты основаны на том, что вам уже известно, — на необходимости прислушиваться к мнению заказчика и необходимости понимания того, что вы строите, до начала работы над кодом. Приготовьтесь... Пора поиграть в архитектора.

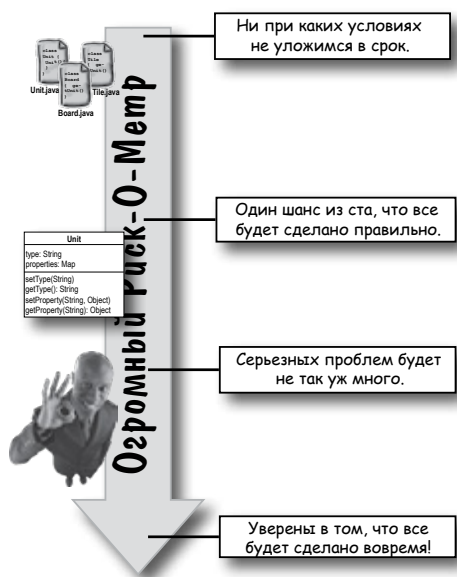
Все дело в подходе к решению большой задачи	307
Так давайте решим БОЛЬШУЮ задачу!	309
Необходимо больше информации	313
Определение функциональных возможностей	316
Используем диаграммы вариантов	322
Маленький субъект	327
Субъекты тоже люди (но не всегда)	328
Займемся анализом предметной области!	333
Разделяй и властвуй	335
Не забывайте, кто ваш заказчик	339
Что такое «паттерны проектирования»? И как их использовать?	341
Сила ООАП (и немного здравого смысла)	344
Инструментарий ООАП	346



## Архитектура

## Навести порядок в хаосе

**Любая работа с чего-то начинается, но вам стоит выбрать правильное что-то!** Вы уже знаете, как разбить приложение на множество мелких задач, но это означает, что мелких задач будет действительно МНОГО. В этой главе мы поможем вам разобраться, с чего начать и как избежать потерь времени на движение в ошибочном направлении. Пришло время взять все эти мелкие детали, разложенные на вашем рабочем месте, и понять, как собрать из них четко структурированное, хорошо спроектированное приложение. Попутно вы узнаете о трех главных вопросах архитектуры и поймете, что «Риск» — это не только интересная настольная игра из 1980-х годов.

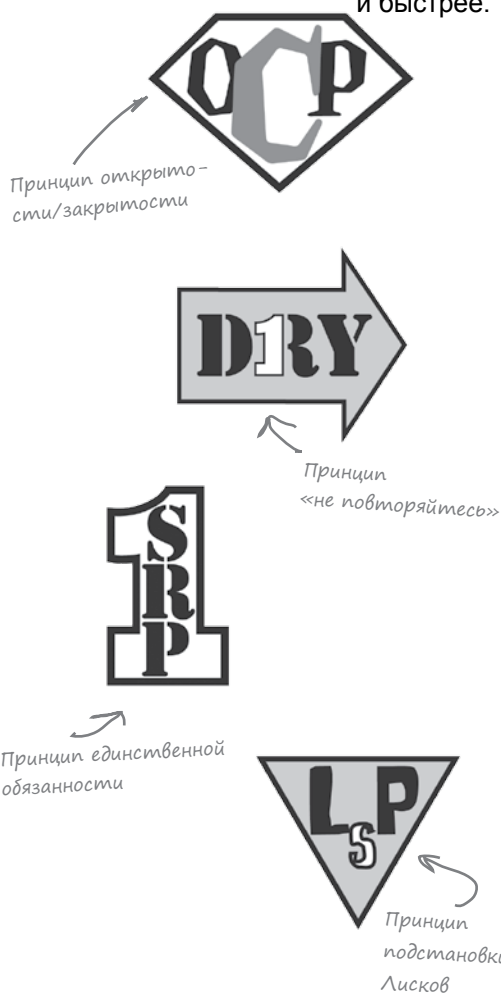


Слегка растерялись?	348
Нам нужна архитектура	350
Начнем с функциональности	353
Но что из этого считать самым важным?	353
Три основных вопроса архитектуры	356
Сценарии способствуют сокращению рисков	365
Классы Tile и Unit	372
Больше порядка, меньше хаоса	374
Чем займемся теперь?	375
Юниты для конкретной игры... что это означает?	376
Возвращаемся к общности	379
Анализ общности: путь к гибкости кода	385
Сокращение рисков помогает писать хорошие программы	395
Ключевые вопросы	396

# 8 Принципы проектирования

## Не стремитесь к оригинальности

**Имитация — самая искренняя форма проявления ума.** Ничто не доставляет столько удовлетворения, как создание совершенно нового, оригинального решения задачи, которая не давала вам покоя несколько дней. Пока не выяснится, что кто-то уже решил ту же задачу задолго до вас, да еще и сделал это лучше вас! В этой главе рассматриваются принципы проектирования, которые были созданы за прошедшие годы, и то, как они помогают вам стать более квалифицированным программистом. Откажитесь от мысли «сделать по-своему», в этой главе вы узнаете, как сделать умнее и быстрее.

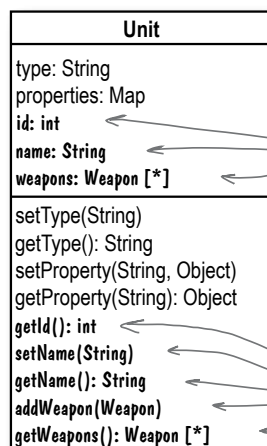


Кратко о принципах проектирования	398
Принцип № 1: принцип открытости/закрытости	399
ОСР шаг за шагом	401
Принцип № 2: не повторяйтесь	404
DRY: ОДНО требование в ОДНОМ месте	406
Принцип № 3: принцип единственной обязанности	412
Выявление множественных обязанностей	414
От множественных обязанностей к единственной	417
Принцип № 4: принцип подстановки Лисков	422
Пример неправильного использования наследования	423
Принцип LSP выявляет скрытые проблемы в структуре наследования structure	424
«Замена базового типа субтипом не должна отразиться на работе программы»	425
Нарушение LSP усложняет код	426
Делегирование функциональности другому классу	428
Использование композиции для объединения поведения других классов	430
Агрегирование: композиция с продолжением существования	434
Агрегирование и композиция	435
Наследование — всего лишь одна из возможностей	436
Ключевые моменты	439
Инструментарий ООАП	440

# 9 Итерации и тестирование

## Программы пишутся для заказчика

Пришло время показать заказчику, что вы действительно **беспокоитесь о его интересах**. Назойливое начальство? Обеспокоенные заказчики? Люди, которые постоянно спрашивают: «Ну что, успеете к сроку?» Никакой хорошо структурированный код не порадует вашего заказчика; вы должны показать ему что-то работающее. А теперь, когда вы овладели солидным инструментарием ОО-программирования, пора узнать, как доказать заказчику, что его программа действительно работает. В этой главе мы рассмотрим два способа углубленного анализа функциональности вашего продукта, после которого заказчик с чувством скажет: *«Да, вы определенно идеально подходите для этой работы!»*



Инструментарий постепенно заполняется	444
Но программу-то вы пишете для ЗАКАЗЧИКА!	445
Углубление итераций: два основных варианта	447
Функционально-ориентированная разработка	448
Сценарно-ориентированная разработка	449
Два метода разработки	450
Использование функционально-ориентированной разработки	453
Анализ функциональной возможности	454
Написание тестовых сценариев	457
Воспользуемся решением, ориентированным на общность	470
Сопоставьте тесты с разработанной структурой	472
Изучаем тестовые примеры...	474
Представьте результат заказчику	480
До настоящего момента мы занимались контрактным программированием	482
Контрактное программирование основано на доверии	483
Перемещение юнитов	492
Ключевые моменты	495
Инструментарий ООАП	498

# 10

## Жизненный цикл ООАП

### Все вместе

**Далеко ли до цели?** Мы изучили множество способов совершенствования программных продуктов, но пришло время собрать все воедино. Да, это именно тот момент, которого вы так долго ждали: мы возьмем все, о чем вы узнали из книги, и покажем, что в действительности это были лишь части общего процесса, который можно использовать вновь и вновь для написания хороших программ.

Разработка в стиле ООАП	502
Жизненный цикл проекта в стиле ООАП	503
Задача	506
Начало итераций	521
Подробнее о представлении линий метро	523
Использовать класс Line или не использовать класс Line...	532
Достопримечательности (класса) метро Объектвиля	538
Защита ваших классов (и классов клиентов)	541
Возвращаемся к фазе требований...	551
Все внимание коду, потом все внимание заказчикам	553
Итеративный метод упрощает решение	557
Как выглядит маршрут?	562
Увидеть Объектвиль собственными глазами	566
Нужна ли итерация №3?	569
Путешествие продолжается...	570






Приложение I. Остатки

Десять главных тем (не рассмотренных в книге)

Хотите верьте, хотите нет, но это еще не все. Да, позади осталось более 550 страниц, и кое-что в них не уместилось. И хотя каждая из последних десяти тем заслуживают разве что краткого упоминания, мы решили выдать вам на дорогу чуть больше информации по каждой из них. Теперь вам будет о чем поговорить во время рекламных пауз... да и кто не любит время от времени поговорить о ООАП? А когда закончите читать это приложение, останется еще одно... И немного рекламы... А потом — действительно всё. Честное слово!



**Антипаттерны**

Антипаттерны являются противоположностью паттернов проектирования: это распространенные ПЛОХИЕ решения. Старайтесь своевременно распознавать ловушки и избегать их.

1. Отношения «является» и «содержит»	572
2. Форматы вариантов использования	574
3. Антипаттерны	577
4. Карты CRC	578
5. Метрики	580
6. Диаграммы последовательности	581
7. Диаграммы состояния	582
8. Модульное тестирование	584
9. Стандарты программирования и удобочитаемость кода	586
10. Рефакторинг	588

Здесь записываются как операции, выполняемые классом самостоятельно, так и операции, в выполнении которых участвуют другие классы.

**Класс: DogDoor**

Описание: представляет физическую собачью дверь. Предоставляет интерфейс к оборудованию, которое управляет дверью.

Обязанности:

Название	Сотрудник
Открыть дверь	
Закрыть дверь	

У этих операций классов-сотрудников нет.

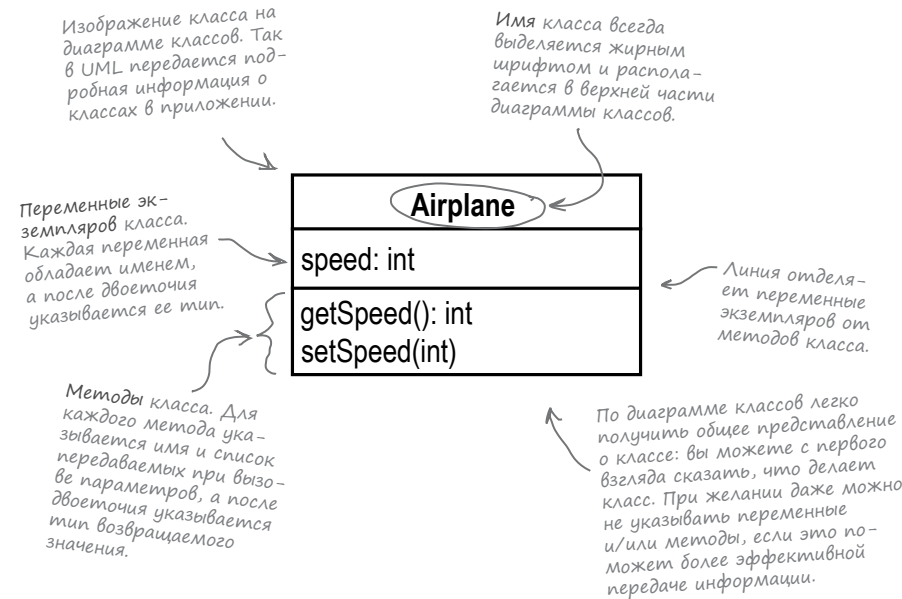
# II

## Приложение II. Добро пожаловать в ОбъектВиль

### Говорим на языке ООП

**Приготовьтесь к путешествию в другую страну.** Пришло время навестить ОбъектВиль — место, где объекты делают то, что им положено, все приложения хорошо инкапсулированы (вскоре вы узнаете, что это означает), а программные структуры обеспечивают простоту расширения и повторного использования. Но прежде чем браться за дело, необходимо кое-что знать заранее — в частности, овладеть кое-какими языковыми навыками. Не беспокойтесь, это не займет много времени. Вы и не заметите, как начнете говорить на языке ООП так, словно вы уже давно живете в элитном районе Объектвиля.

Добро пожаловать в ОбъектВиль	590
UML и диаграммы классов	591
Наследование	593
И еще полиморфизм...	595
И наконец, инкапсуляция	596
Теперь кто угодно может задать значение speed напрямую	596
И из-за чего столько шума?	597



# С чего начинаются хорошие программы

Дорогая, с тех пор, как я начал использовать ООАТ, я стал другим человеком... Клянусь, совершенно другим человеком!



**Так как же на самом деле пишутся хорошие программы?** Всегда сложно понять, с чего следует начинать. Делает ли приложение то, что ему положено делать? И как насчет таких тонкостей, как дублирование кода, — ведь это всегда плохо, верно? Да, определить, **над чем следует работать в первую очередь**, бывает нелегко, к тому же еще нужно не запороть все остальное по ходу дела. Но пусть вас это не тревожит! К тому моменту, когда вы дочитаете эту главу, вы **будете знать, как пишутся хорошие программы**, а ваш подход к разработке приложений навсегда изменится. И наконец-то вы поймете, почему ООАП — это такое слово, которое не стыдно произносить в приличном обществе.

## Рок-н-ролл навсегда!

Нет ничего лучше звука классной гитары в руках великого музыканта. Фирма Рика специализируется на поиске идеальных инструментов для своих требовательных покупателей.



Вы не поверите, какой у нас тут выбор. Только скажите, какая гитара вам нужна, и мы подберем вам идеальный инструмент, не сомневайтесь!

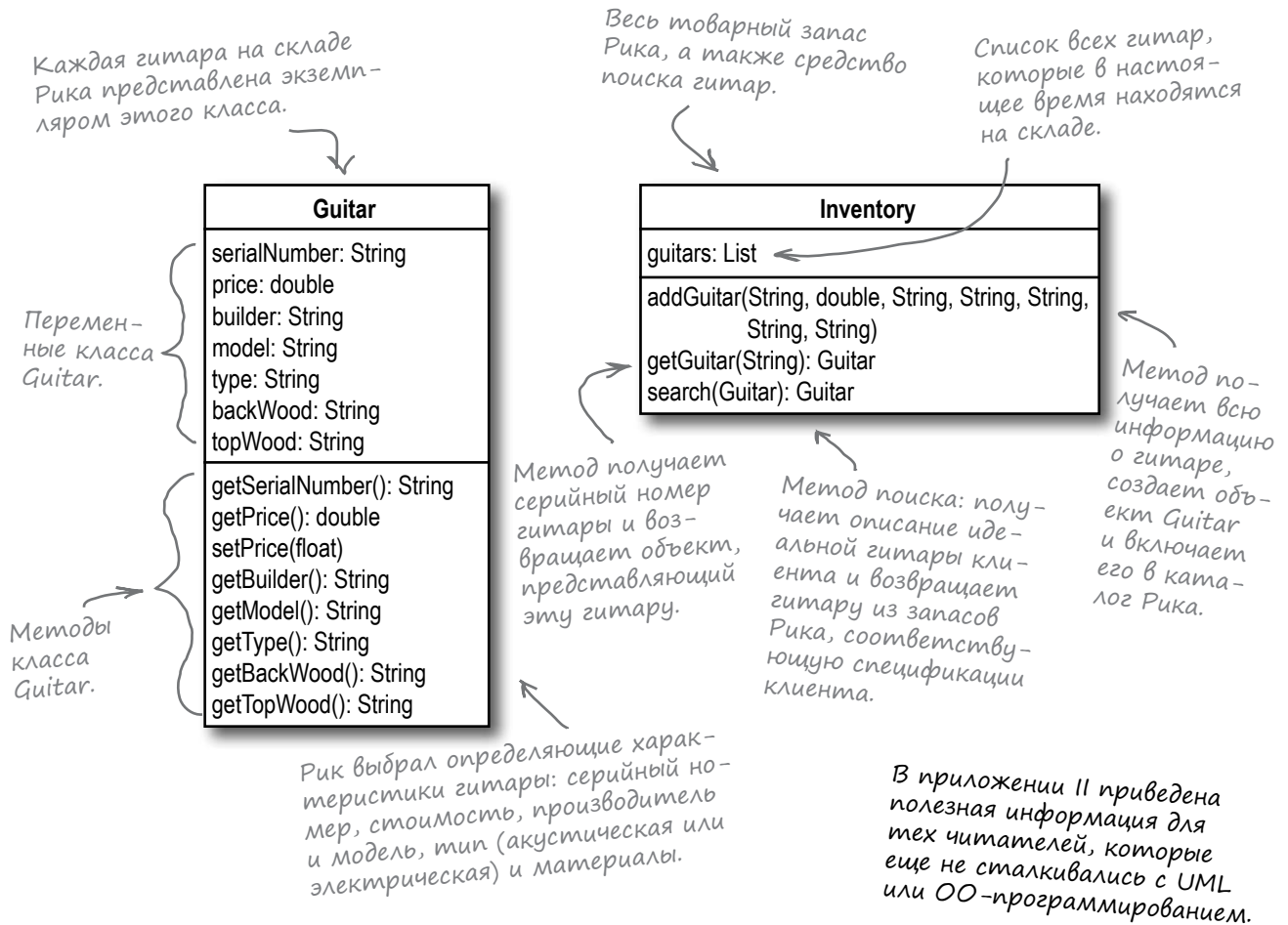
Знакомьтесь: Рик, страстный любитель гитары и владелец магазина гитар экстра-класса.



Несколько месяцев назад Рик решил выбросить свою картотеку и начал хранить информацию о текущем ассортименте на компьютере. Он обратился с заказом в популярную фирму «Дешево и сердито», и там ему уже написали систему складского учета. Рик даже вытребовал средство поиска, которое поможет ему подобрать для каждого клиента «инструмент мечты».

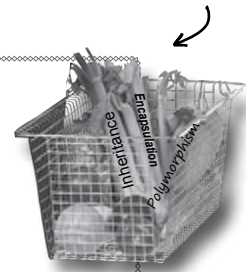
## Новенькое приложение Рика...

А вот и приложение, которое Рик получил от разработчиков из фирмы «Дешево и сердито». Построенная ими система полностью заменяет все рукописные заметки Рика и помогает ему подобрать идеальную гитару для каждого клиента. Чтобы показать, что было сделано, разработчики вручили Рiku следующую диаграмму классов UML:



### Недавно в Объектвиле?

Если вы не разбираетесь в объектно-ориентированном программировании, никогда не слышали о UML или не уверены в том, что означает приведенная диаграмма, — ничего страшного! Мы подготовили специальный краткий курс для новичков. Загляните в конец книги и прочитайте приложение II — обещаем, вы не пожалеете. А потом возвращайтесь сюда, и все происходящее покажется вам куда более осмысленным.



## А вот как выглядит код Guitar.java

Диаграмма классов приложения Рика была приведена на предыдущей странице; а теперь давайте посмотрим, как выглядит реальный код **Guitar.java** и **Inventory.java**.

```
public class Guitar {

    private String serialNumber, builder, model, type, backWood, topWood;
    private double price;

    public Guitar(String serialNumber, double price,
                  String builder, String model, String type,
                  String backWood, String topWood) {
        this.serialNumber = serialNumber;
        this.price = price;
        this.builder = builder;
        this.model = model;
        this.type = type;
        this.backWood = backWood;
        this.topWood = topWood;
    }

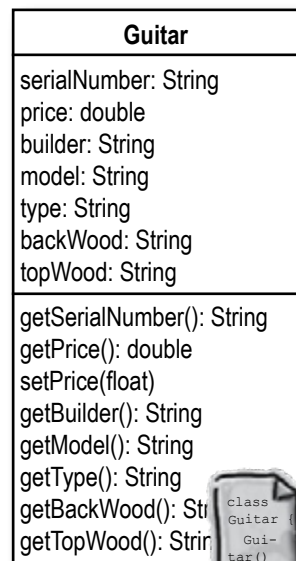
    public String getSerialNumber() {
        return serialNumber;
    }

    public double getPrice() {
        return price;
    }
    public void setPrice(float newPrice) {
        this.price = newPrice;
    }
    public String getBuilder() {
        return builder;
    }
    public String getModel() {
        return model;
    }
    public String getType() {
        return type;
    }
    public String getBackWood() {
        return backWood;
    }
    public String getTopWood() {
        return topWood;
    }
}
```

Свойства, которые мы видели на диаграмме класса *Guitar*.

На диаграммах классов UML конструкторы не отображаются. Впрочем, конструктор *Guitar* делает ровно то, что можно от него ожидать: он задает все свойства нового объекта *Guitar*.

Методы на диаграмме классов соответствуют методам в коде класса *Guitar*.



```
class
Guitar {
    Guitar()
}
```

Guitar.java

## U Inventory.java...

```

public class Inventory {
    private List guitars;

    public Inventory() {
        guitars = new LinkedList();
    }

    public void addGuitar(String serialNumber, double price,
                          String builder, String model,
                          String type, String backWood, String topWood) {
        Guitar guitar = new Guitar(serialNumber, price, builder,
                                   model, type, backWood, topWood);
        guitars.add(guitar);
    }

    public Guitar getGuitar(String serialNumber) {
        for (Iterator i = guitars.iterator(); i.hasNext(); ) {
            Guitar guitar = (Guitar)i.next();
            if (guitar.getSerialNumber().equals(serialNumber)) {
                return guitar;
            }
        }
        return null;
    }

    public Guitar search(Guitar searchGuitar) {
        for (Iterator i = guitars.iterator(); i.hasNext(); ) {
            Guitar guitar = (Guitar)i.next();
            // Серийный номер игнорируется, так как он уникален
            // Цена игнорируется, так как она уникальна
            String builder = searchGuitar.getBuilder();
            if ((builder != null) && (!builder.equals("")) &&
                (!builder.equals(guitar.getBuilder())))
                continue;
            String model = searchGuitar.getModel();
            if ((model != null) && (!model.equals("")) &&
                (!model.equals(guitar.getModel())))
                continue;
            String type = searchGuitar.getType();
            if ((type != null) && (!searchGuitar.equals("")) &&
                (!type.equals(guitar.getType())))
                continue;
            String backWood = searchGuitar.getBackWood();
            if ((backWood != null) && (!backWood.equals("")) &&
                (!backWood.equals(guitar.getBackWood())))
                continue;
            String topWood = searchGuitar.getTopWood();
            if ((topWood != null) && (!topWood.equals("")) &&
                (!topWood.equals(guitar.getTopWood())))
                continue;
        }
        return null;
    }
}

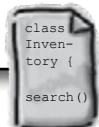
```

← Не забудьте, что директивы импорта были опущены для экономии места.

← addGuitar() получает все свойства, необходимые для создания нового экземпляра Guitar, создает его и включает в каталог.

← Метод работает довольно примитивно... Каждое свойство полученного объекта Guitar сравнивается с аналогичным свойством каждого объекта Guitar в каталоге Рика.

Inventory
guitars: List
addGuitar(String, double, String, String, String, String, String)
getGuitar(String): Guitar
search(Guitar): Guitar



Inventory.java



## А потом от Рика стали уходить клиенты...

Похоже, независимо от запросов клиента новая поисковая программа Рика всегда выдает пустой результат. Но Рик совершенно точно знает, что у него на складе есть подходящая гитара... Что происходит?

*FindGuitarTester.java имитирует типичный рабочий день Рика. Приходит клиент, говорит, что ему нужно, — Рик проводит поиск по каталогу.*

```
public class FindGuitarTester {

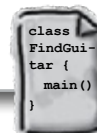
    public static void main(String[] args) {
        // Инициализация каталога гитар
        Inventory inventory = new Inventory();
        initializeInventory(inventory);

        Guitar whatErinLikes = new Guitar("", 0, "fender", "Stratocaster",
                                           "electric", "Alder", "Alder");

        Guitar guitar = inventory.search(whatErinLikes);
        if (guitar != null) {
            System.out.println("Erin, you might like this " +
                               guitar.getBuilder() + " " + guitar.getModel() + " " +
                               guitar.getType() + " guitar:\n    " +
                               guitar.getBackWood() + " back and sides,\n    " +
                               guitar.getTopWood() + " top.\nYou can have it for only $" +
                               guitar.getPrice() + "!");
        } else {
            System.out.println("Sorry, Erin, we have nothing for you.");
        }
    }

    private static void initializeInventory(Inventory inventory) {
        // Включение описаний гитар в каталог.
    }
}
```

*Эрин ищет гитару Фендер «Страто-кастер», сделанную из ольхи (Alder).*



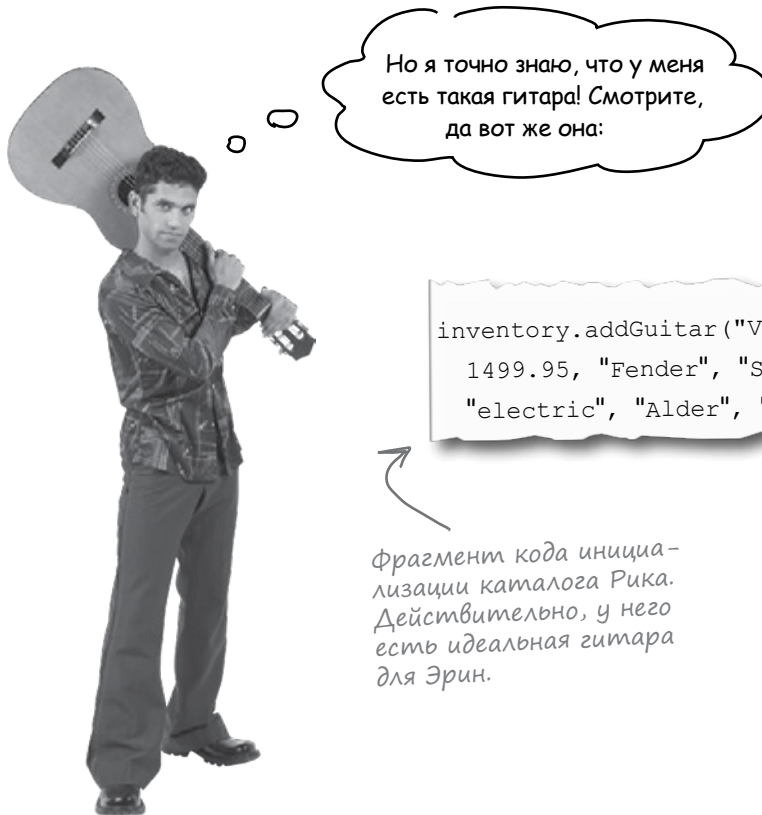
FindGuitarTester.java

```
File Edit Window Help C7#5
%java FindGuitarTester
Sorry, Erin, we have nothing for you.
```

*Вот что происходит, когда Эрин приходит в магазин, а Рик пытается подобрать ей гитару.*

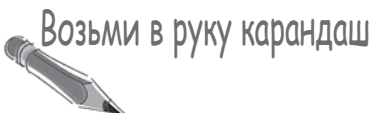
Извини, Рик, я зайду к твоим конкурентам.





Фрагмент кода инициализации каталога Рика. Действительно, у него есть идеальная гитара для Эрин.

И все характеристики совпадают с тем, что хочет Эрин... В чем дело?



Возьми в руку карандаш

Как бы вы переработали приложение Рика?

Просмотрите код приложения на трех последних страницах и результаты поиска. Какие недостатки вы заметили? Что бы вы изменили в этом коде? Запишите внизу ПЕРВОЕ, что бы вы сделали для усовершенствования приложения Рика.

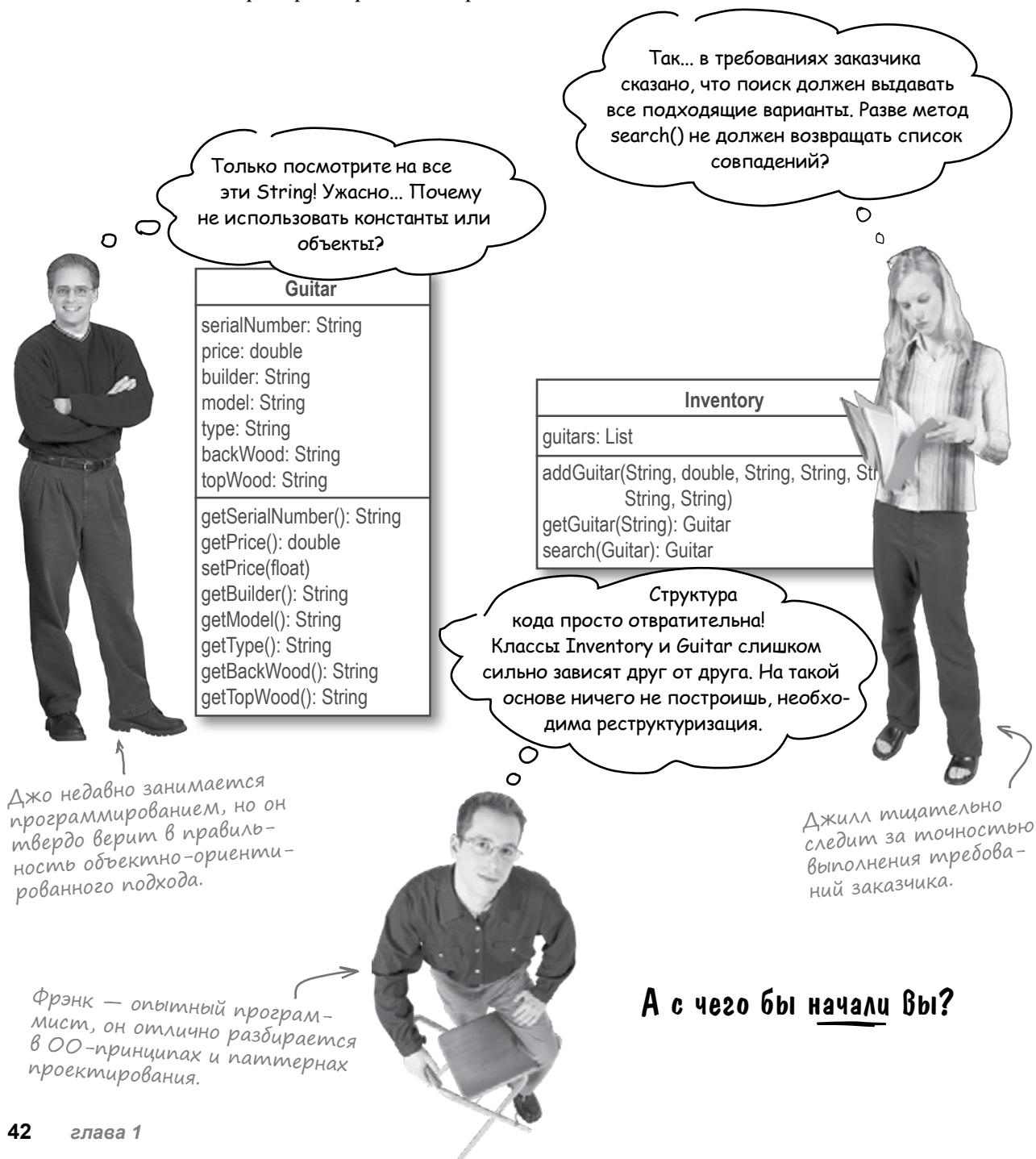
---

---

---

## Что бы вы изменили В ПЕРВУЮ ОЧЕРЕДЬ?

Очевидно, в приложении Рика есть проблемы — но пока не совсем понятно, с чего начинать. А выбор широк, прямо глаза разбегаются:



Откуда я знаю, с чего начинать? В каждом новом проекте, над которым я работаю, у всех разные мнения по поводу того, что делать в первую очередь. Иногда я угадываю верно, иногда приходится переделывать все приложение из-за того, что я начал не с того. А я просто хочу писать хорошие программы! Так с чего нужно начать в приложении Рика?



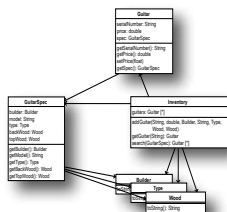
**Как писать  
хорошие  
программы  
каждый раз?**

**Хороший вопрос... и на него есть много разных ответов:**

**«Хорошая программа всегда делает то, что хочет заказчик. Даже если заказчик захочет использовать программу каким-то новым способом, программа не должна сбоить или выдавать неожиданные результаты».**



- При таком подходе главное — чтобы заказчик был доволен тем, как работает приложение.



**«Хорошая программа — это объектно-ориентированный код. Он не содержит дубликатов, а каждый объект более или менее полно управляет своим поведением. Такие программы легко расширять, потому что они имеют надежную и гибкую структуру кода».**

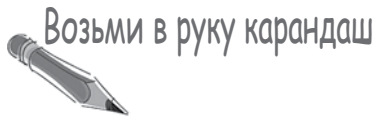
При структурно-ориентированном подходе код оптимизируется для расширения и повторного использования, в нем используются паттерны проектирования и проверенные ОО-методологии.

Хороший ОО-программист всегда старается сделать свой код более гибким.

Пытайтесь понять, что все это значит? Ничего страшного... узнаете в следующих главах.

«Хорошая программа строится с использованием проверенных паттернов и принципов проектирования. Объекты обладают слабой связностью; код открыт для расширения, но закрыт для изменения. Такая структура кода также упрощает его повторное использование, поскольку вам не приходится переписывать все заново, чтобы использовать готовые части своего приложения».





А что, по **вашему** мнению, следует считать «хорошей программой»?

Вы видели мнения нескольких программистов разных типов... кто из них прав? А может, у вас есть свое определение того, что такое «хорошая программа»? Ваша очередь. Запишите те свойства, которыми, по вашему мнению, должна обладать хорошая программа:

Напишите здесь свое имя...

...а здесь — свое определение хорошей программы.

\_\_\_\_\_ :

« \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_ »

## Хорошая программа... сочетание нескольких факторов

Чтобы понять, что именно следует считать «хорошей программой», простого определения недостаточно. Каждый из программистов на с. 44 говорил лишь об *одной* из сторон того, что делает программу действительно хорошей.

**Прежде всего, хорошая программа должна удовлетворять заказчика. Она должна делать то, что от нее хочет заказчик.**



### **Соблюдайте требования заказчика**

*Заказчик будет считать вашу программу хорошей, если она делает то, что ей положено делать.*

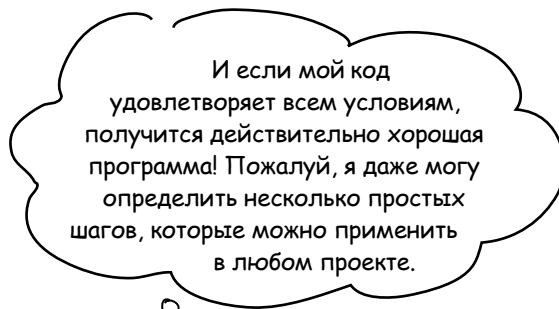
Правильно работающая программа — это, конечно, хорошо... но не все. Вы понимаете это, когда потребуется дополнить код новыми возможностями или использовать его в другом приложении. Недостаточно написать программу, которая выполняет все требования заказчика; ваша программа должна пройти проверку временем.

**Итак, хорошая программа хорошо спроектирована, хорошо запрограммирована, проста в сопровождении, повторном использовании и расширении.**



### **Ваш код должен быть таким же умным, как вы сами.**

*Для вас (и ваших коллег) хорошая программа должна быть простой в сопровождении, расширении и повторном использовании.*



И если мой код удовлетворяет всем условиям, получится действительно хорошая программа! Пожалуй, я даже могу определить несколько простых шагов, которые можно применить в любом проекте.





# Хорошая программа за 3 простых шага

1. Убедитесь в том,  
что поведение програм-  
мы соответствует тре-  
бованиям заказчика.

↑  
Может, сейчас они не кажутся простыми, но, как вы вскоре увидите, применение ООАП и некоторых базовых принципов навсегда изменит качество вашего кода.

← Этот шаг ориентирован на заказчика. ПРЕЖДЕ ВСЕГО убедитесь, что приложение делает то, что положено. В этом вам помогут четко сформулированные требования и анализ.

↓  
2. Применяйте базовые  
ОО-принципы для  
повышения гибкости.

↖  
Когда программа заработает, поищите дубликаты кода, которые могли в ней остаться, и убедитесь в том, что в коде используются проверенные приемы ОО-программирования.

↗  
Получилось хорошее объектно-ориентированное приложение, которое делает то, что нужно? Примените паттерны и принципы, чтобы ваше приложение оставалось жизнеспособным в будущем.

↓  
3. Постарайтесь создать  
структуру кода, упрощаю-  
щую его сопровождение и  
повторное использование.