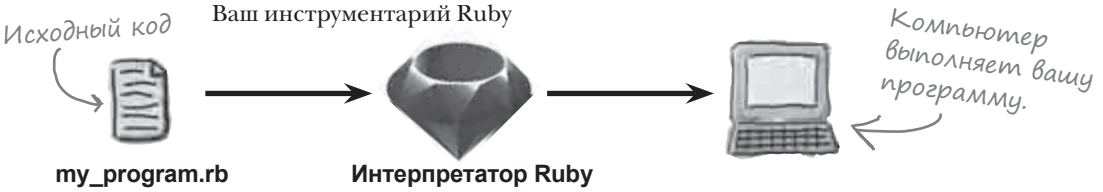
1 Как сделать больше меньшими усилиями

Программируйте так, как вам удобно

Вас интересует, почему вокруг столько говорят о языке Ruby и подойдет ли он вам? Ответьте на простой вопрос: ***вам нравится работать эффективно?*** Кажется ли вам, что с многочисленными компиляторами, библиотеками, файлами классов и комбинациями клавиш других языков вы действительно становитесь ближе к **готовому продукту**, **восхищению коллег** и **толпе счастливых заказчиков**? Вы хотите, чтобы язык программирования **занимался техническими подробностями** за вас? Если вам хочется перестать писать рутинный код и *работать над задачей*, то язык Ruby — для вас. Ruby позволит вам **сделать больше с меньшим объемом кода.**



2 Методы и классы

Наводим порядок

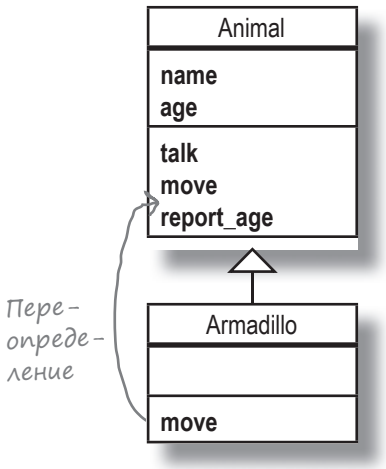
В вашей работе кое-чего не хватало. Да, вы вызывали методы и создавали объекты, как настоящий профессионал. Но при этом вы могли вызывать только те методы и создавать только те виды объектов, которые были определены за вас в Ruby. Теперь ваша очередь. В этой главе вы научитесь создавать *свои* методы, а также свои **классы** — своего рода «шаблоны» для создания новых объектов. *Вы сами решаете*, как будут выглядеть объекты, созданные на базе вашего класса. **Переменные экземпляра** определяют, какая информация *хранится* в ваших объектах, а **методы экземпляра** определяют, что эти объекты *делают*. А самое главное — вы узнаете, как определение собственных классов *упрощает чтение и сопровождение вашего кода*.

3 Наследование

С родительской помощью

Столько повторений! Новые классы, представляющие разные типы машин или животных, удобны — это правда. Но ведь вам придется *копировать методы экземпляра из класса в класс*. И эти копии будут вести самостоятельную жизнь — одни будут работать нормально, в других могут появиться ошибки. Разве классы создавались не для того, чтобы *упростить* сопровождение кода?

В этой главе вы научитесь применять **наследование**, чтобы ваши классы могли использовать *одни и те же* методы. Меньше дубликатов — меньше проблем с сопровождением!



4

Инициализация экземпляров

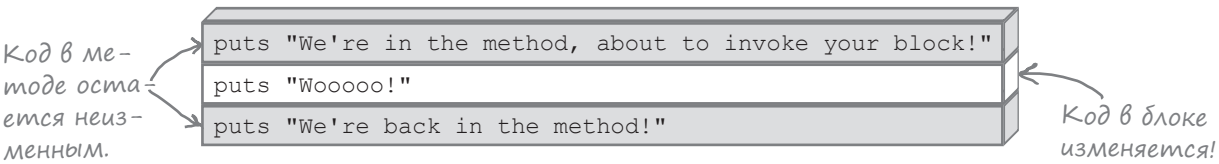
Хороший старт — половина дела

Пока что ваш класс напоминает бомбу с часовым механизмом. Каждый экземпляр, созданный вами, начинается «с пустого места». Если вы вызовете методы экземпляра до того, как в нем появятся данные, то может произойти ошибка, приводящая к аварийному завершению программы. В этой главе мы представим пару способов создания объектов, которые можно сразу безопасно использовать. Начнем с метода `initialize`, который позволяет передать набор аргументов для заполнения данных объекта *на момент его создания*. Затем мы покажем, как написать **методы класса**, которые позволяют **еще** проще создавать и инициализировать объекты.

5 Массивы и блоки

Лучше, чем цикл

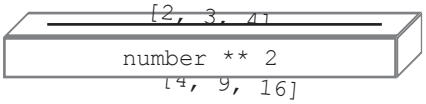
Очень многие задачи из области программирования связаны с обработкой списков. Списки адресов. Списки телефонных номеров. Списки продуктов. Мац, создатель языка Ruby, знал об этом. Поэтому он *основательно* потрудился над тем, чтобы работать со списками в Ruby было *действительно просто*. Сначала он поработал над тем, чтобы **массивы**, используемые для работы со списками в Ruby, обладали множеством *мощных методов* для выполнения практически любых операций. Затем он осознал, что написание кода для *перебора всех элементов списка* для выполнения некоторой операции с каждым элементом — утомительная рутинная работа, которую программистам приходится выполнять *очень часто*. Поэтому он добавил в язык **блоки**, благодаря которым код перебора стал лишним. Что же это такое — блок? Сейчас узнаете...

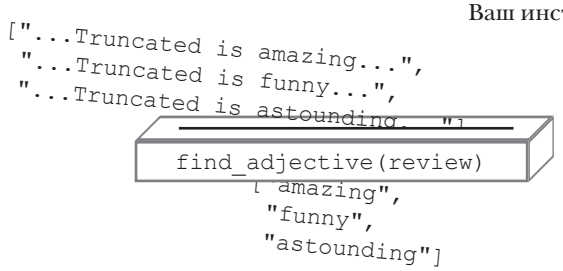


6 Возвращаемые значения блоков

Как будем обрабатывать?

Вы видели лишь малую часть возможностей блоков. До настоящего момента *методы* передавали данные *блоку* и ожидали, что блок сделает все необходимое. Однако *блок* также может возвращать данные *методу*. Эта возможность позволяет методу получать *команды* от блока, позволяя ему выполнять более содержательную работу. Методы, рассмотренные в этой главе, получают *большие и сложные* коллекции и используют **возвращаемые значения блоков** для сокращения их размера.





7 Хеши

Пометка данных

Хранить данные в одной большой куче удобно... До тех пор, пока вам не потребуется в них что-нибудь найти. Вы уже видели, как создать коллекцию объектов с использованием *массива*. Вы уже видели, как обработать *каждый элемент* массива и как *найти нужные* элементы. В обоих случаях мы начинаем от начала массива и *проверяем каждый отдельный объект.* Вы уже видели методы, получающие большие коллекции в параметрах. Вам уже известно, какие проблемы это создаст: вызовы методов требуют передачи большой, *запутанной коллекции аргументов,* для которой вам нужно помнить точный порядок. А не существует ли коллекции, у которой *все данные* уже снабжены *метками*? Тогда вы могли бы *быстро найти* нужные элементы! В этой главе рассматриваются **хеши** Ruby, предназначенные именно для этого.

Массив

Хеш

8 Ссылки

Путаница с сообщениями

Вы когда-нибудь отправляли сообщения не тому контакту?

Наверное, вам пришлось изрядно потрудиться, чтобы разобраться с возникшей путаницей. *Объекты Ruby* очень похожи на *контакты* в адресной книге, и *вызов методов* для них напоминает *отправку сообщений*. Если содержимое вашей адресной книги будет *перепутано*, вы рискуете отправить сообщение *не тому объекту*. В этой главе вы научитесь *распознавать такие ситуации* и узнаете, как снова *наладить работу ваших программ*.

Западная улица в реальности

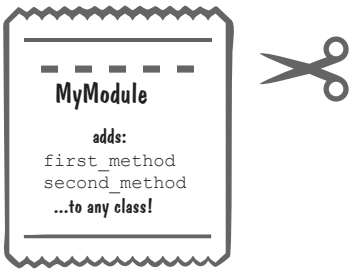
Западная улица по адресной книге Артура

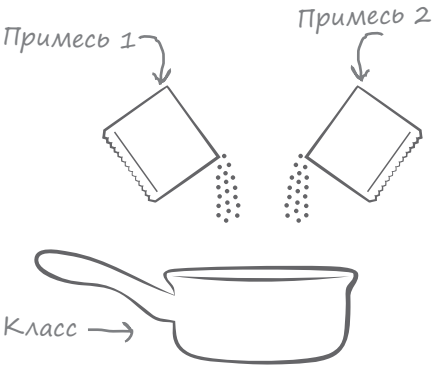
9 Примеси

Аккуратный выбор

У наследования есть свои ограничения. Наследовать методы можно только от одного класса. Но что, если вам понадобилось использовать *некоторое поведение* в совершенно разных классах? Представьте методы для запуска цикла зарядки и получения информации о текущем уровне заряда — такие методы могут использоваться телефонами, электродрелями и электромобилями. И вы готовы создать *один* суперкласс для всех *этих* устройств? (Не пытайтесь, ничем хорошим это не кончится.) Или методы запуска и остановки двигателя. Конечно, эти методы могут пригодиться дрели и автомобилю, но не телефону!

В этой главе рассматриваются **модули** и **примеси** — мощный механизм *группировки методов* и их последующего включения *только в те классы, для которых они актуальны*.



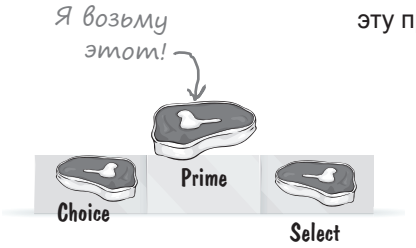


10

Comparable и enumerable

Готовые примеси

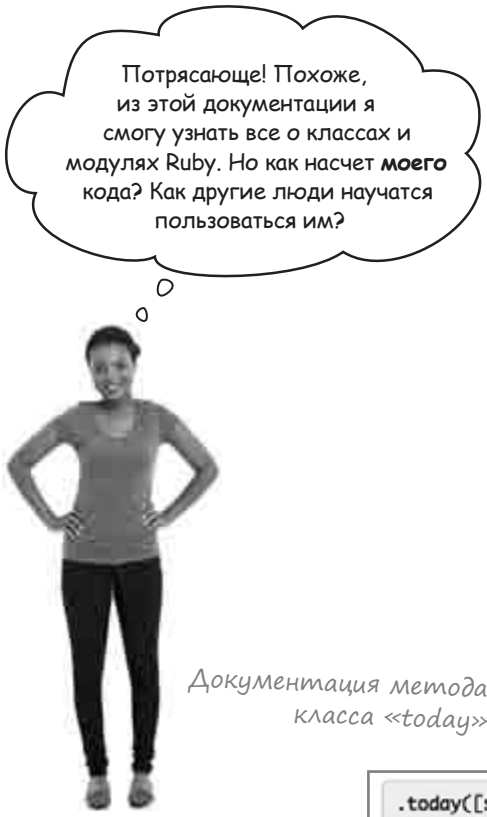
Из предыдущей главы вы уже знаете, что примеси полезны. Но вы еще не видели их истинную мощь. В стандартную библиотеку Ruby входят две совершенно *потрясающие* примеси. Первая, `Comparable`, используется для сравнения объектов. Мы уже использовали такие операторы, как <, > и ==, с числами и строками, а с `Comparable` сможете использовать их с *вашими* классами. Вторая примесь, `Enumerable`, применяется при работе с коллекциями. Помните замечательные методы `find_all`, `reject` и `map`, которые мы использовали с массивами? Так вот, они были предоставлены `Enumerable`. Но это лишь малая часть возможностей `Enumerable`! И разумеется, вы можете включать эту примесь в *свои* классы. Хотите узнать, как это делается? Читайте дальше!



11 Документация

Читайте документацию

В книге не хватит места, чтобы рассказать все о Ruby. Есть старая поговорка: «Дайте человеку рыбу, и он будет сыт один день. Научите его ловить рыбу, и он будет сыт всю жизнь». До сих пор мы *давали вам рыбу*: показали, как использовать некоторые классы и модули Ruby. Однако существуют десятки других классов и модулей, которые могут пригодиться в вашей работе, но нам не хватит места, чтобы описать их в книге. Пришло время *научить вас ловить рыбу*. Существует превосходная бесплатная **документация** по всем классам, модулям и методам Ruby. Вам просто нужно знать, где ее найти и как ее интерпретировать. Именно об этом пойдет речь в этой главе.



Документация метода класса «today»

```
.today([start = Date::ITALY]) ⇒ Object

Date.today #=> #<Date: 2011-06-11 ..>

Creates a date object denoting the present day.
```

```
#year ⇒ Integer

Returns the year.
```

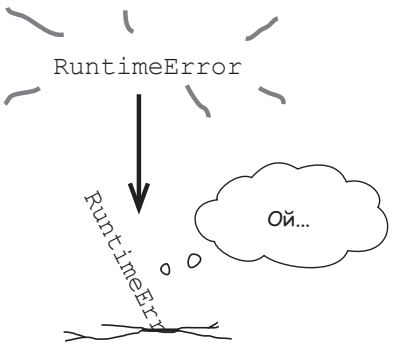
Документация метода экземпляра «year».

12 Исключения

Непредвиденное препятствие

В реальном мире порой происходит нечто неожиданное. Кто-то удаляет файл, который пытается загрузить ваша программа, или сервер, с которым программа пытается связаться, выходит из строя. Вы можете проверять такие исключительные ситуации в своем коде, но такие проверки перемешиваются с кодом, выполняемым в ходе нормальной работы. (И в результате возникает жуткая мешанина, в которой невозможно разобраться.)

Эта глава посвящена средствам обработки исключений в Ruby, которые позволяют писать код для обработки аварийных ситуаций и размещать его отдельно от нормального кода.



13

Модульное тестирование

Контроль качества кода

А вы уверены, что ваша программа работает правильно? Действительно уверены? Прежде чем передавать новую версию пользователям, вы, вероятно, опробовали новые функции и убедились в том, что они работают. Но проверили ли вы *старые* функции, чтобы убедиться, что они не перестали работать? *Все* старые функции? Если этот вопрос не дает вам покоя, значит, вашей программе необходимо автоматизированное тестирование. Автоматизированные тесты гарантируют, что основные компоненты программы продолжают правильно работать даже после изменения кода. **Модульные тесты** — самая распространенная, самая важная разновидность автоматизированных тестов. В поставку Ruby входит библиотека **MiniTest**, предназначенная для модульного тестирования. В этой главе вы узнаете все, что действительно необходимо знать об этой библиотеке!

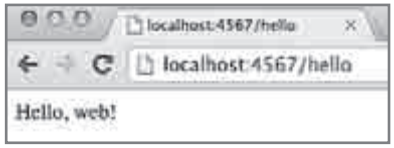
14

Веб-приложения

На раздаче HTML

На дворе XXI век. Пользователи требуют веб-приложения. И Ruby поможет вам в этом! Существуют библиотеки, которые помогут вам развернуть собственные веб-приложения и обеспечить доступ к ним из любого браузера. Итак, в двух последних главах этой книги мы покажем вам, как построить полноценное веб-приложение.

Прежде всего вам понадобится Sinatra — независимая библиотека для создания веб-приложений. Но не беспокойтесь, мы научим вас использовать программу RubyGems (включенную в поставку Ruby) для автоматизации загрузки и установки библиотек! Затем будет рассмотрена разметка HTML — ровно столько, сколько необходимо для создания собственных веб-страниц. И конечно, мы покажем, как предоставить эти страницы браузеру!

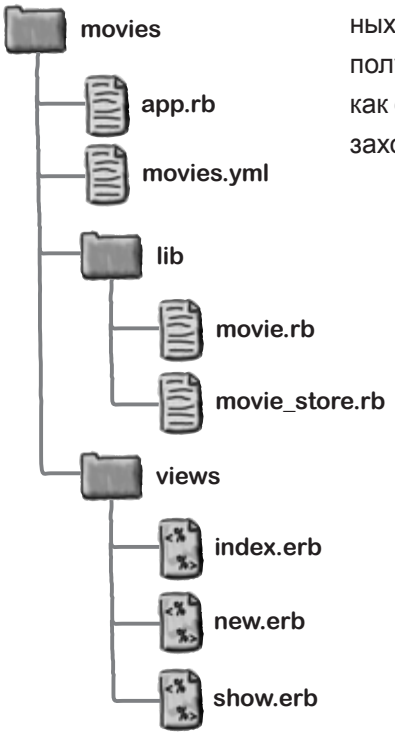


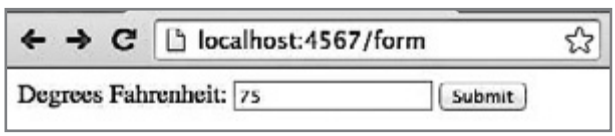
15 Сохранение и загрузка данных

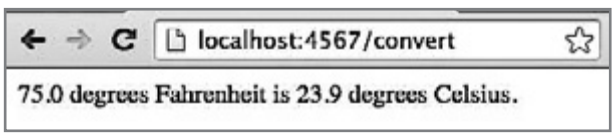
Сбережения пользователя

Сейчас наше веб-приложение просто выбрасывает данные, введенные пользователем. Мы создали форму, на которой пользователь *вводит* данные. Пользователь ожидает. что данные будут *сохранены*, чтобы их можно было *прочитать* и *вывести* позднее. Но сейчас ничего подобного не происходит! Все введенные данные просто *пропадают*.

В последней главе книги приложение будет подготовлено к сохранению данных, введенных пользователем. Мы покажем, как настроить приложение для получения данных формы, как преобразовать эти данные в объекты Ruby, как сохранить их в файле и как загрузить нужный объект, когда пользователь захочет увидеть его. Готовы? Так доделаем наше приложение!







Приложение. Оставшиеся темы

Десять основных тем (не рассмотренных в книге)

Мы прошли долгий путь, и книга почти закончена. Мы будем скучать по вам, но было бы неправильно расставаться и отпускать вас в самостоятельное путешествие без еще *нескольких* напутственных слов. Нам при всем желании не удалось бы уместить все, что вам еще нужно знать о Ruby, на этих страницах… (Вообще-то сначала мы *уместили* все необходимое, уменьшив размер шрифта в 25 000 раз. Все поместилось, но текст было не прочитать без микроскопа, поэтому материал пришлось изрядно сократить.) Но мы оставили все самое лучшее для этого приложения.

И это уже *действительно* конец книги!