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!

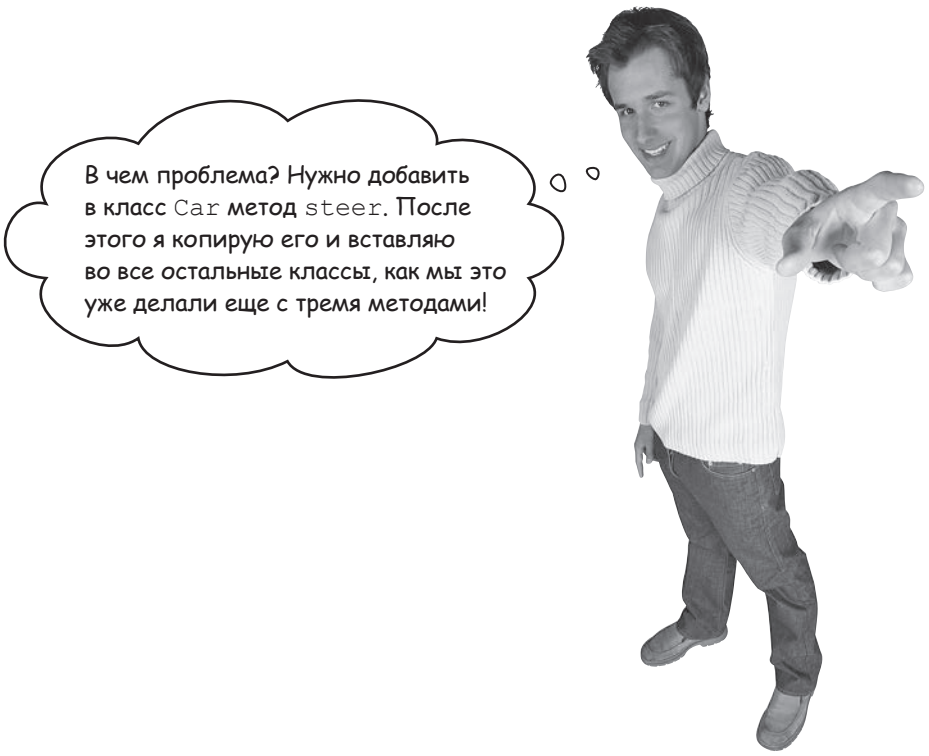
Копировать, вставлять... Столько проблем!

Разработчики из Got-A-Motor, Inc. решили внедрить принципы объектно-ориентированного программирования в своей работе. Старое приложение виртуального тест-драйва было переработано так, чтобы каждый тип машины был представлен отдельным классом. У них есть классы, представляющие легковые машины (Car), грузовики (Truck) и мотоциклы (Motorcycle).

На данный момент структура классов выглядит так:

	Car	Truck	Motorcycle
переменные экземпляра	odometer gas_used	odometer gas_used	odometer gas_used
методы экземпляра	mileage accelerate sound_horn	mileage accelerate sound_horn	mileage accelerate sound_horn

Прислушавшись к пожеланиям клиентов, руководство захотело включить во все типы машин метод для управления рулем. Майк, неопытный разработчик из Got-A-Motor, считает, что с этим требованием справиться несложно.



Код Майка для приложения виртуального тест-драйва

```
class Car

  attr_accessor :odometer
  attr_accessor :gas_used

  def mileage
    @odometer / @gas_used
  end

  def accelerate
    puts "Floor it!"
  end

  def sound_horn
    puts "Beep! Beep!"
  end

  def steer                          Копируем!
    puts "Turn front 2 wheels."
  end

end
```

```
class Truck

  attr_accessor :odometer
  attr_accessor :gas_used

  def mileage
    @odometer / @gas_used
  end

  def accelerate
    puts "Floor it!"
  end

  def sound_horn
    puts "Beep! Beep!"
  end

  def steer                          Вставляем!
    puts "Turn front 2 wheels."
  end

end
```

```
class Motorcycle

  attr_accessor :odometer
  attr_accessor :gas_used

  def mileage
    @odometer / @gas_used
  end

  def accelerate
    puts "Floor it!"
  end

  def sound_horn
    puts "Beep! Beep!"
  end

  def steer                          Вставляем!
    puts "Turn front 2 wheels."
  end

end
```

Но у Марси, опытного объектно-ориентированного разработчика из этой группы, такая идея вызывает сомнения.

Марси права; сопровождение кода вскоре превратится в сущий кошмар. Для начала разберемся, как решить проблему с дублированием, а затем исправим метод `steer` для объектов `Motorcycle`.

На помощь приходит наследование!

К счастью, в Ruby, как и в большинстве объектно-ориентированных языков, поддерживается концепция **наследования**, позволяющая классам наследовать методы друг от друга. Если один класс поддерживает некоторую функциональность, другие классы, наследующие от него, наделяются этой функциональностью *автоматически*.

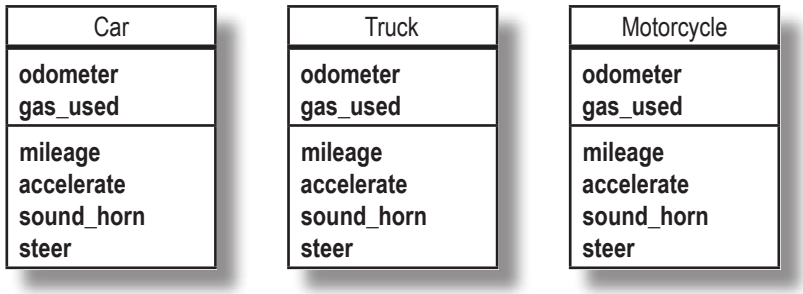
Вместо того чтобы повторять определения метода во многих похожих классах, вы выделяете общие методы в один класс, а затем указываете, что другие классы наследуют от него. Класс, содержащий общие методы, называется **суперклассом**, а классы, наследующие методы, называются **субклассами**.

Если суперкласс содержит методы экземпляра, то эти методы автоматически наследуются его субклассами. Вы можете получить доступ ко всем нужным методам суперкласса без дублирования их кода в каждом субклассе.

А теперь посмотрим, как наследование поможет исключить дублирование кода в приложении виртуального тест-драйва...

Наследование позволяет нескольким субклассам наследовать методы от одного суперкласса.

1. Мы видим, что классы `Car`, `Truck` и `Motorcycle` содержат несколько общих методов экземпляра и атрибутов.



2. Каждый из этих классов является разновидностью машины. Мы можем создать новый класс (допустим, он будет называться `Vehicle`) и переместить в него общие методы и атрибуты.