ЛЕКЦИЯ 11

КЛАССЫ И ОБЪЕКТЫ В RUBY

ПЛАН ЗАНЯТИЯ

- Как работать со временем в Ruby
- Запись в файлы, пишем дневник

- Мы научимся писать данные в файлы, узнаем как работать со временем в Ruby, напишем программу-дневник.

КЛАССЫ

- В программах мы постоянно оперируем объектами, мы уже говорили об этом в 4-м уроке: строки, числа, массивы.
- Наши объекты хранятся с помощью переменных: неких ярлыков, которые позволяют обращаться к объектам по имени.

КЛАССЫ

- Вы уже знаете, что в Ruby есть много разных видов объектов: строки (String), целые числа (Fixnum), массивы (Array). Пришло время осознать, что этих типов гораздо больше: есть ещё файлы (File), ассоциативные массивы (Hash), метки (Symbol) даже моменты времени (Time) и даты (Date), а также много-много всего другого.

КЛАССЫ

- Типы объектов в программировании называются **классами**. Ruby не исключение.
- Ruby вообще очень высокоразвитый язык, там любая закорючка — это объект какого-то класса. Но это так, лирическое отступление.
- Напомним, что посмотреть класс любого объекта можно вызвав у этого объекта по цепочке методы .class и .name.

КЛАССЫ

- Что такое методы объектов станет понятно к концу занятия, а пока просто напомним, что:

```
puts "Я строка".class.name
```

- выведет на экран String, а

```
puts ["А","я","массив","строк"].class.name
```

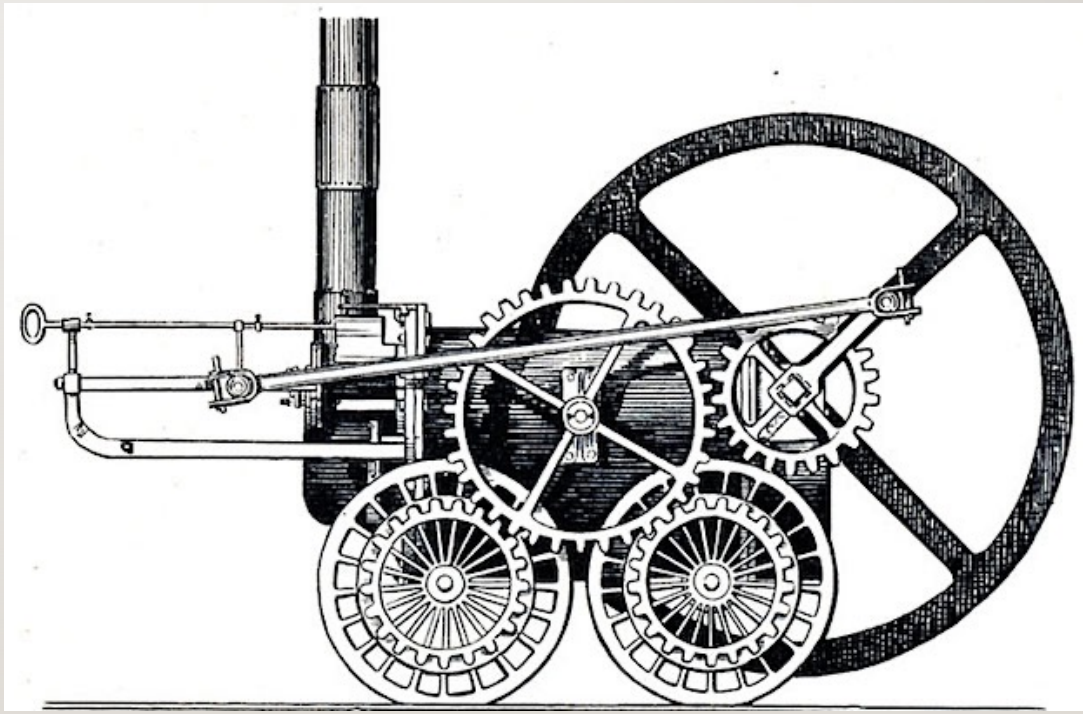
- выведет на экран Array.

ДЛЯ ЧЕГО СОЗДАЮТСЯ КЛАССЫ?

- Класс — это некое описание типа объектов, которые можно создавать. Прежде чем человек создал первый паровоз, он как-то описал (на бумаге, в своем воображении, в чертежах) новое для того времени понятие «паровоз». Он наверняка придумал какими свойствами должен обладать паровоз, как он должен функционировать и так далее.

ДЛЯ ЧЕГО СОЗДАЮТСЯ КЛАССЫ?

- Другими словами он придумал новое понятие, новый тип объектов «паровоз». Программисты бы сказали — создал класс Паровоз.
- И уже потом человечество начало производить различные конкретные паровозы, создавать объекты этого класса.



ДЛЯ ЧЕГО СОЗДАЮТСЯ КЛАССЫ?

- То есть — прежде чем создавать какие-то объекты в вашей программе, Ruby должен знать их класс. А для этого нужно сперва объявить класс. Объявить класс это значит описать в программе, как должен класс называться и главное — какими свойствами и поведением он должен обладать.
- До сих пор мы использовали встроенные в Ruby классы (строки, числа, массивы) – мы создавали объекты этих классов и с ними играли. Нам не нужно было описывать эти классы, ведь они уже описаны в самом языке Ruby.

ДЛЯ ЧЕГО СОЗДАЮТСЯ КЛАССЫ?

- Что делать, если вам понадобился в программе новый тип объектов, которых нету в языке? А бывает так очень часто. Тогда вам нужно написать свой класс.
- Давайте определимся с философским вопросом: **«Когда нужно создавать новые классы»**. Сразу скажем, что понимание это приходит с опытом, поэтому не бойтесь экспериментировать и делать классы когда считаете это нужным.
- Ругать вас за это никто не будет. Дадим лишь несколько советов, как понять, что ситуация требует именно нового класса, а не просто метода.

ДЛЯ ЧЕГО СОЗДАЮТСЯ КЛАССЫ?

- 1. Если вы понимаете, что некую часть вашей программы можно выделить в независимый объект. Объект со своим собственным поведением и свойствами.
- Главное, чтобы вы чётко понимали, как вы объясните другому человеку, что это за класс. Если вы можете сформулировать на русском языке название класса в виде простого слова — это хороший признак, что он заслуживает существования.
- Например, для нашей виселицы из предыдущего урока можно было бы создать класс: «Печатальщик результата», который бы занимался всем, что связано с выводом информации для пользователя в консоль.

ДЛЯ ЧЕГО СОЗДАЮТСЯ КЛАССЫ?

- 2. Если в языке программирования не предусмотрен какой-то уже имеющийся класс для вашей цели.
- Чаще всего просто погуглив, вы либо найдёте нужный класс в Ruby, либо поймёте, как в этой ситуации поступают другие люди. Если же информацию быстро найти не удалось, смело делайте свой класс.

ДЛЯ ЧЕГО СОЗДАЮТСЯ КЛАССЫ?

- 3. Если в вашей программе есть абстрактная модель чего-то.
- Например, если вы пишете программу для управления библиотекой, то понятно, что вам нужен класс книги или даже класс стеллажа, может быть, понадобится класс автора и класс жанра.
- Это всё определяется в момент проектирования программы, подробнее об этом процессе мы говорили на 10-м занятии.

КАК СОЗДАТЬ КЛАСС В RUBY?

- Во-первых, классы солидны. То есть, класс представляет собой такой конкретный объёмный кусок программы. Часто случается так, что программисты в своей работе используют один и тот же класс в нескольких проектах.
- Иногда классы даже включаются в структуру языка, как это стало с классами строки String и момента времени Time, настолько они удобные.
- Классы принято описывать в отдельных файлах. Каждому классу — свой файл. Это существенно упрощает понимание программы.

КАК СОЗДАТЬ КЛАСС В RUBY?

- Во-вторых, классы важны. **Мы не можем создавать классы, называя их абы как.** Придумайте своему классу осмысленное название, чтобы вы могли посреди ночи проснуться и по названию класса сказать, что он делает, хотя бы приблизительно.

КАК СОЗДАТЬ КЛАСС В RUBY?

- Для примера создадим класс Мост (Bridge), который мы опишем в файле bridge.rb:

```
class Bridge
  # Описание класса
end
```

- Чтобы Ruby понял, что это не просто код, а описание класса, мы обернули его в конструкцию class-end. Обратите внимание, что все имена классов в Ruby (в других языках тоже) обычно **начинаются с большой буквы**.

КАК СОЗДАТЬ КЛАСС В RUBY?

- Если бы класс состоял из двух слов, то второе слово тоже было бы с большой буквы:

```
class RoadBridge
  ...
end
```

- Внутри конструкции class-end мы пишем методы нашего класса. Как мы уже знаем, методы описываются с помощью конструкции def-end.
- Особое внимание следует обратить на метод initialize — это так называемый конструктор класса, но об этом чуть позже.

СОЗДАНИЕ ЭКЗЕМПЛЯРА КЛАССА

- Пока мы просто описали класс, ничего интересного не произойдёт. Нужно создать хотя бы один объект этого класса. Для этого нам в нашей основной программе doroga.rb необходимо подключить файл bridge.rb с описанием класса Bridge.
- Мы умеем делать это с помощью команды require:

```
require "bridge.rb"
```

СОЗДАНИЕ ЭКЗЕМПЛЯРА КЛАССА

- После этого можно создать новый объект нашего нового класса Bridge. Для этого мы пишем:

```
bridge = Bridge.new
```

- Это очень важный момент! Давайте разберёмся, что значит каждое слово в этой записи.

СОЗДАНИЕ ЭКЗЕМПЛЯРА КЛАССА

```
bridge = Bridge.new
```

- Во-первых, что такое bridge, оно написано маленькими буквами, значит это не класс, а объект, вернее переменная.
- Во-вторых знак равно (=), он означает, что мы в переменную bridge хотим что-то записать, хотим, чтобы переменная bridge указывал на то, что будет справа от знака равно.
- В-третьих, мы видим название нашего нового класса Bridge. Мы только что описали этот класс в отдельном файле bridge.rb и подключили его (файл) с помощью команды require.

СОЗДАНИЕ ЭКЗЕМПЛЯРА КЛАССА

```
bridge = Bridge.new
```

- Наконец, .new — мы вызвали у нашего класса специальный метод, который как бы говорит классу: «О великий и могучий! Создай для нас свое земное воплощение в виде конкретного объекта!»
- А класс отвечает: «Так и быть, я сжалюсь над тобой, смертный и дам тебе свой экземпляр, но при одном условии — я сразу же вызову у этой копии метод initialize».
- Поэтому метод .new возвращает новый объект данного класса.

СОЗДАНИЕ ЭКЗЕМПЛЯРА КЛАССА

- Причем при создании объекта у него вызывается специальный метод с именем initialize. Такой метод в программировании называется **конструктор**.

```
class Bridge
  def initialize
    puts "Мост создан"
  end
end
```

СОЗДАНИЕ ЭКЗЕМПЛЯРА КЛАССА

- Вы можете объявить в классе такой метод и написать в нем какой-то функционал – тогда этот функционал будет выполнен один раз при создании каждого объекта этого класса. Но можно и не писать, тогда конструктор будет пустой, объект создастся без каких-то дополнительных действий.
- Конкретный объект какого-то класса в программировании называется **экземпляр класса.** По-английски instance. Запомните эти слова, вот увидите, они несут свет озарения в чистом виде! ;)

СОЗДАНИЕ ЭКЗЕМПЛЯРА КЛАССА

- Конечно, всю эту драму придумали разработчики, чтобы было удобнее создавать новые классы. В методе initialize, который вызывается каждый раз, когда создаётся новый объект указанного класса, описывается, что должно произойти с экземпляром класса перед тем, как он будет создан. Если это класс книги, например, то нужно заполнить её название и год издания. Может быть ещё имя и фамилию автора и жанр. Всё на усмотрение разработчика класса.

СОЗДАНИЕ ЭКЗЕМПЛЯРА КЛАССА

- Ещё раз, **объект** (**экземпляр класса**) и **класс** — это разные вещи, как есть, например «Михаил» и «Иван» — объекты, а есть «Человек» — класс, некий собирательный образ, абстракция для всех людей на Земле (и на её орбите, а возможно и в других галактиках).

СОЗДАНИЕ ЭКЗЕМПЛЯРА КЛАССА

- Итак, мы создали новый экземпляр класса Bridge и сделали так, что переменная bridge указывает на этот объект.
- Если мы напишем:

```
puts bridge.class.name
```

- То увидим название нашего класса Bridge.
- *А теперь смертельный номер. Если всё в Ruby это экземпляр какого-то класса, то что же тогда такое этот наш Bridge? Какого будет вам узнать, что это тоже объект! «Какой же у него класс?» — спросите вы. Посмотрите сами ;-)*

ИСПОЛЬЗОВАНИЕ МЕТОДОВ КЛАССА

```
def open
  puts "Мост открыт, можно ехать"
end
```

- Внутри нашего класса Bridge мы написали метод open. Этот метод на самом деле есть не у самого класса, а именно у его экземпляра.

ИСПОЛЬЗОВАНИЕ МЕТОДОВ КЛАССА

- Для того, чтобы «открыть» мост (объект класса Bridge), на который указывает переменная bridge, нам необходимо вызвать у этого объекта метод open. Это делается очень просто и изящно:

```
bridge.open
```

- и мы увидим в консоли наш текст открытия моста:

```
Мост открыт, можно ехать!
```

ИСПОЛЬЗОВАНИЕ МЕТОДОВ КЛАССА

- Именно вызов метода экземпляра класса мы делали, когда вызывали у массива, например, метод to_s:

```
array = [1,2,3]
puts array.to_s
```

- выводит в консоль "[1, 2, 3]" — мы вызываем у объекта array (экземпляра класса Array) метод to_s, который возвращает этот массив, но уже как строку (экземпляр класса String).

ИСПОЛЬЗОВАНИЕ МЕТОДОВ КЛАССА

- Именно вызов метода экземпляра класса мы делали, когда вызывали у массива, например, метод to_s:

```
array = [1,2,3]
puts array.to_s
```

- выводит в консоль "[1, 2, 3]" — мы вызываем у объекта array (экземпляра класса Array) метод to_s, который возвращает этот массив, но уже как строку (экземпляр класса String).

ПОЛЯ КЛАССА

- В методы класса, как и в обычные методы можно передавать параметры, как и обычные методы, они возвращают (или нет) какие-то значения.
- Единственное отличие этих методов, в том, что они привязаны к экземпляру класса и в этих методах в связи с этим доступны «**поля класса**» или «**переменные экземпляра класса**» или «**переменные объекта**». Такие переменные используются для хранения состояния экземпляра класса, его свойств.
- Например, наш мост bridge (экземпляр класса Bridge) может быть каменным или деревянным, длинным или коротким, узким или широким, пешеходным или автомобильным (или даже железнодорожным) и так далее.

ПОЛЯ КЛАССА

- Давайте сделаем наш мост открывающимся и для этого создадим поле класса opened (открыт). В руби поля класса начинаются с символа «собаки» — @ (чтобы не путались с методами), поэтому в конструкторе мы опишем поведение моста по умолчанию в таком виде:

```
def initialize
  puts "Мост создан"
  @opened = false
end
```

- а в метод open добавим изменение этого внутреннего поля на true:

```
def open
  @opened = true
  puts "Мост открыт, можно ехать"
end
```

ПОЛЯ КЛАССА

- Все важные поля вашего объекта должны быть объявлены в конструкторе! Вам нужно сообщить Ruby заранее какими свойствами будут обладать объекты вашего класса.
- Текущее значение всех полей какого-то объекта определяют так называемое *состояние* объекта. Фактически один объект отличается от другого объекта того же класса своим состоянием (один мост открыт, другой закрыт, например).

ПОЛЯ КЛАССА

- Мы также напишем новый метод 'is_opened?', который будет возвращать true, если мост открыт и false, если закрыт:

```
def is_opened?
  return @opened
end
```

- Программисты Ruby договорились между собой, что все методы, которые возвращают true или false, будут заканчиваться знаком вопроса. В других языках как правило знак вопроса не используют в названиях методов.

ПОЛЯ КЛАССА

- Мы также напишем новый метод 'is_opened?', который будет возвращать true, если мост открыт и false, если закрыт:

```
def is_opened?
  return @opened
end
```

- Программисты Ruby договорились между собой, что *все методы, которые возвращают true или false, будут заканчиваться знаком вопроса*. В других языках как правило знак вопроса не используют в названиях методов.

ПОЛЯ КЛАССА

- Обратите внимание, что мы никак не можем достучаться до поля класса из нашей программы, именно поэтому для каждого обращения к ней нам нужен отдельный метод (если это действительно необходимо делать из нашей программы).
- В самой программе doroga.rb мы теперь перепишем открытие моста только для случая, когда мост закрыт:

```
if !bridge.is_open?
  bridge.open
end
```

- После этого наш мост откроется и напишет Мост открыт, можно ехать!.

ПОЛЯ КЛАССА

- Ещё раз обратим ваше внимание, что если мы создадим новый мост:

```
another_bridge = Bridge.new
```

- то этот новый мост будет закрыт. another_bridge.is_open? вернёт false.
- Надо просто немного привыкнуть к этой концепции класс-объект. После небольшой практики вы будете в этом как рыба в воде.

ПОЛЯ КЛАССА

- Кстати, рыба и селедка — селедка это объект (если конкретная селедка, вот эта).

- А просто "рыба" это уже класс ;)

ХАМЕЛЕОН

- Создайте класс «Хамелеон», у которого есть один метод - поменять цвет. Метод принимает на вход один параметр — цвет в виде строки (например "красный") и выводит на экран строку:

```
Теперь я красный!
```

- Создайте экземпляр хамелеона и поменяйте его цвет несколько раз.

ХАМЕЛЕОН. ПОДСКАЗКА

- Создайте файл cameleo.rb (хамелеон) и определите в нём класс Cameleo, как в уроке.
- Потом напишите в этом классе метод change_color, который принимает на вход параметр color_name. В теле метода напишите нужный текст, добавив параметр с цветом.
- А в самой программке подключите файл с помощью require (как в уроке), создайте нового Хамелеона, вызвав Cameleo.new и сохранив его в переменную cameleo.
- А потом трижды вызовите метод change_color у объекта cameleo с разными параметрами.

ЧЕЛОВЕК С ИМЕНЕМ И ФАМИЛИЕЙ

- Создайте класс «Человек» с двумя свойствами: имя и отчество.
- В этом классе напишите два метода: конструктор и метод, который будет возвращать полное имя человека. Конструктор принимает имя и отчество и записывает их в нужные поля. Второй метод возвращает полное имя человека.
- Напишите программу, которая использует этот класс: создайте трёх разных людей и выведите на экран их полные имена:

```
У нас есть три человека:
Гаврила Петрович
Фёдор Петрович
Василий Алибабаевич
```

ЧЕЛОВЕК С ИМЕНЕМ И ФАМИЛИЕЙ. ПОДСКАЗКА

- Сделайте класс Person по образу и подобию класса Bridge, который мы разбирали в уроке (только вместо свойства @open у экземпляра этого класса будут два других: @first_name и @middle_name).
- Напишите также для этого класса метод full_name, который будет выводить полное имя: значение этих двух переменных с пробелом посередине.
- Подключайте новый класс в вашу программу. В программе заведите три переменные и присвойте им значения новых объектов вашего класса. И потом выведите на экран полное имя этих объектов, используя написанный метод.

ЧЕЛОВЕК В ВОЗРАСТЕ

- Доработайте программу из предыдущего задания так, чтобы в конструкторе теперь передавался (и сохранялся в переменной экземпляра класса) еще один параметр: возраст.
- Добавьте в класс метод, который говорит, пожилой человек (возраст > 60) или нет. А метод, который выводит полное имя, поправьте так, чтобы молодежь он называл только по имени, а пожилых уважительно, по имени и отчеству.

ЧЕЛОВЕК В ВОЗРАСТЕ

- Создайте в программе пару человек с разными именами и возрастами и выведите на экран информацию о них.
- **Например:**

```
У нас есть два человека:
Сергей
И ему 41 – молодой
Константин Львович
И ему 61 – пожилой
```

ЧЕЛОВЕК В ВОЗРАСТЕ. ПОДСКАЗКА

- Просто добавьте в конструктор ещё один параметр age, и сохраняйте его в переменной @age которая будет хранить возраст.
- Метод, который будет возвращать, пожилой человек или молодой назовите old? — с вопросительным знаком на конце. Используйте этот метод, чтобы добавить логику вывода полного имени.

БОДИБИЛДЕРЫ

- Создайте класс Бодибилдер.
- У него должно быть одно свойство на каждую группу мышц (сколько всего мышц, решайте сами, главное, **не меньше 3**). Напишите конструктор, который создаёт бодибилдера-хиляка, у которого все мышцы по нулям.
- Потом создайте для нашего бодибилдера метод прокачать мышцу: в качестве параметра передаётся название мышцы, которое совпадает с названием свойства соответствующей группы мышц.
- Метод увеличивает эту мышцу на 1.
- Создайте также метод, который выводит на экран «прокачку» бодибилдера: на разных строчках выводит текущее состояние каждой группы мышц.

БОДИБИЛДЕРЫ

- Подключите класс, создайте двух-трёх бодибилдеров, покачайте их и покажите их жюри.
- **Например:**

```
Первый бодибилдер:
Бицепсы: 7
Трицепсы: 5
Дельтовидка: 10

Второй бодибилдер:
Бицепсы: 4
Трицепсы: 10
Дельтовидка: 7

Третий бодибилдер:
Бицепсы: 5
Трицепсы: 8
Дельтовидка: 4
```

БОДИБИЛДЕРЫ

- Аналогично второй задаче сделайте файл body_builder.rb и создайте там класс BodyBuilder.
- У него три метода: конструктор initialize, который объявляет переменные экземпляра класса (@triceps, @biceps, @deltovidka), метод для раскачки мышцы (pump), которому передаём в качестве параметра muscle строку.
- Мышцу для раскачки выбираем с помощью case: если передали "triceps" — увеличиваем @triceps на 1 и так далее.
- Потом пишем последний метод: show_muscles, который просто выводит значения всех переменных в консоль с помощью puts.
- После этого подключаем body_builder.rb в основную программу и устраиваем конкурс. Чтобы не писать вызовы методов для раскачки мышц несколько раз, можно воспользоваться циклами.

СПРАВОЧНАЯ ИНФОРМАЦИЯ

- Книга Грэди Буча про ООП, которую каждый должен прочесть!
- Что такое поля класса
- Классы в Ruby

СПАСИБО ЗА ВНИМАНИЕ!

Лекция 10. Классы и объекты в Ruby