Классы, объекты и переменные

Андрей Васильев

2015

Задача

Мы управляем магазином поддержанных книг. Каждую неделю проводится инвентаризация. Работники сканируют бар-коды на книгах и сохраняют их в CSV-списки.

Пример файла

```
"Date","ISBN","Price"
"2013-04-12","978-1-9343561-0-4",39.45
"2013-04-13","978-1-9343561-6-6",45.67
"2013-04-14","978-1-9343560-7-4",36.95
```

Задачи системы

- Выяснить количество книг каждого типа
- Общую стоимость всех книг

Идентификация

При проектировании решения в объектно-ориентированном подходе сначала необходимо идентифицировать элементы, с которыми необходимо работать.

- Сущность, описывающая одну запись в таблице
- Коллекция объектов - все данные в таблице

Назовём эту сущность `BookInStock`

Инициализация объектов

- После создания каждого объекта Ruby инициализирует объект, вызывая метод `initiailize`
- Метод используется для установки значения переменным экземпляра класса
- Переменные экземпляра начинаются с символа @
- Данные переменные описывают состояние объекта
- Установленные значения должны позволять вызывать любые методы данного класса
- Метод может проверять переданные в него данные

«Печать» объектов

- Метод `p` показывает внутреннее состояние объекта
- Метод `puts` пытается преобразовать объект к строке
- Стандартный способ преобразования - имя класса + уникальный идентификатор

Преобразование к строке

- При преобразовании объекта к строке вызывается метод `to_s`
- Каждый объект содержит метод `to_s` с описанным выше поведением
- Метод можно переопределить
- Метод `to_s` не принимает аргументов
- Метод `to_s` должен вернуть строку

Атрибуты объекта

- Все переменные экземпляра являются приватными
- Обычно создаются методы для доступа и манипулирования внутренним состоянием объекта
- Видимые внешним объектам переменные, описывающие состояние, называются атрибутами
- Создадим методы для доступа к переменным экземпляра

`attr_reader` создаёт методы для чтения значения переменных экземпляра класса

```
attr_reader :isbn, :price
```

- Символами описываем имена переменных
- `attr_reader` создаёт методы, а не меняет видимость переменных

Изменение атрибутов

Обычным для объектно-ориентированных языков способом изменения атрибута является создание специального метода

```
public void setPrice(double newPrice) {
    price = newPrice
}
```

- В Ruby принято оформлять взаимодействие с атрибутами, как с обычными переменными
- Для этого метод, устанавливающий значение имеет на конце символ =

`attr_accessor` предоставляет доступ как на чтение, так и на запись `attr_writer` предоставляет доступ на запись

Виртуальные атрибуты

В Ruby вы всегда взаимодействуете с методами, а не с переменными экземпляра, это позволяет отделить интерфейс класса

- Видимое состояние
- Методы для управления видимым состоянием

от реализации этого интерфейса

виртуальный атрибут - стоимость в копейках

- Чтение значения: `book.price_in_copeks`
- Присваивание значения: `book.price_in_copeks = 15`

Взаимодействие между классами

Во время решения задач классами становятся не только элементы, над которыми мы работаем, но также и внутренние элементы

Класс CsvReader

- Чтение информации из нескольких CVS-файлов
  - `read_in_csv_data`
- Вычисление нужных характеристик
  - `total_value_in_stock`
  - `number_of_each_isbn`

Хранение информации о записях

- Класс `CvsReader` должен сохранять информацию о книгах
- Используем массив для хранения данных

Чтение данных из CVS-файла

- Стандартная библиотека включает в себя класс `CSV`
- Для чтения можно воспользоваться методом `foreach`

Добавление данных в массив

- Метод `<<` добавляет объект в конец массива
- Метод `push` добавляет несколько объектов в конец

Структурирование файлов приложения

- Обычно один исходный файл содержит один класс или один модуль
  - Позволяет точно следить за зависимостями
  - Облегчает рефакторинг исходного кода
  - Облегчает модульное тестирование
  - Предлагает возможности для повторного использования кода
- Для подключения других файлов используются методы
  - `require` для подключения библиотек
  - `require_relative` для подключения собственных файлов
- Приложение также содержит модуль, инициализирующий работу приложения

Контроль доступа

Ruby поддерживает уже известные вам 3 уровня контроля доступа

- Публичные (`public`) методы могут быть вызваны любым объектом
  - По умолчанию все методы кроме `initialize` являются публичными
- Защищённые (`protected`) методы могут быть вызваны внутри дерева наследования
- Приватные (`private`) методы могут быть вызваны только лишь внутри данного класса

Отличия от знакомых вам языков

- Приватные методы нельзя вызывать даже из других объектов этого же класса
- Контроль за вызовом методов осуществляется во время выполнения

Указание контроля доступа

Для указания контроля доступа используются методы `public`, `protected`, `private`

Указание уровня доступа для секции

```ruby
class MyClass
  private
  def method_one
  end
  def method_two
  end
end
```

Указание уровня доступа для методов

```ruby
class MyClass
  pritave :method_one, :method_two
end
```

Переменные

- Основная задача переменных - хранение ссылки на объект
- Переменные не являются объектами

```
person1 = "Tim"
person2 = person1
person1[0] = 'J'
```

- Оператор присваивания записывает ссылку на объект
- Все изменения объекта видны во всех переменных, ссылающихся на данный объект

Предотвращение непродуманных изменений

Использование явного копирования

```
person1 = "Tim"
person2 = person1.dup
person1[0] = 'J'
```

Запрет всех последующих изменений

```
person1 = "Tim"
person2 = person1
person1.freeze
person1[0] = 'J'
```

Проектирование объекта неизменяемым