# Задачи по ООП

Информатика
10-11 классы

6 марта 2012 г.

# Задание

- Создать следующие классы: человек, ученик, ученик–раздолбай, учитель, директор.
- Каждый человек имеет: фамилию, имя, отчество, год рождения. Наследование определено в соответствии со здравым смыслом (ученик–раздолбай — наследник ученика). Все сущности имеют методы:
  1. Посчитать возраст (getAges).
  2. обратиться по имени (getName) по правилу: учитель и директор — имя + отчество, ученик — имя, ученик-раздолбай — “Бяка” + имя.
  3. булевский метод главный (head?): для директора возвращается истина, для остальных — ложь.
- ФИО и год рождения должно задаваться в конструкторе.
- После реализации создать экземпляры каждого класса и вызвать для них методы getName, getAges, head?.

# Шаг 1

- Прежде всего, создадим класс Person.
- В классе есть четыре свойства: first_name, last_name, middle_name, birthday

Listing 1: Основа Person

```
class Person
  attr_accessor :first_name, :last_name,
                :middle_name, :birthday
end
```

# Шаг 2

- В условии задачи требуется, чтобы основные свойства класса задавались сразу.
- То есть, мы хотим, чтобы работал следующий код:

Listing 2: Пример объекта

```
p = Person.new("Иванов", "Иван", "Иванович", 1975)
```

# Шаг 3

- Для того, чтобы при создании можно было сразу задать какие-либо параметры, нам нужно определить конструктор.
- Конструктор на вход (в соответствии с кодом предыдущего слайда) будет принимать 4 аргумента.

Listing 3: Конструктор

```
class Person
  attr_accessor :first_name, :last_name,
                :middle_name, :birthday
  def initialize(fname, lname, mname, birthday)
    @first_name  = fname
    @last_name   = lname
    @middle_name = mname
    @birthday    = birthday
  end
end
```

# Шаг 4

- Возраст у экземпляров класса Person и у его наследников будет считаться всегда одинаково.
- Поэтому определим соответствующий метод в самом классе Person.
- Через механизм наследования метод автоматически будет доступен всем наследникам.

Listing 4: Метод age

```
class Person
  ...
  def age
    2012 - @birthday
  end
end
```

# Шаг 5

- В большинстве случаев метод head? будет возвращать ложь (за исключением объекта класса Директор).
- Поэтому создадим базовый метод в классе Person, а в классе Headmaster используем полиморфизм (переопределение метода) для изменения результата.

Listing 5: Метод head

```
class Person
  ...
  def head?
    false
  end
end
```

# Шаг 6

- Стандартное обращение к человеку — по имени–отчеству.
- Раз стандартное — значит, определяем в классе–родителе. При необходимости используем полиморфизм.

Listing 6: Метод name

```
class Person
  ...
  def name
    @first_name + "␣" + @middle_name
  end
end
```

# Шаг 7

- Проверим класс Person, вызвав последовательно все методы.

**Listing 7: Person**

```
p = Person.new("Иванов", "Иван", "Иванович", 1975)
puts p.name
puts p.age
puts p.head?
```

# Шаг 8

- Класс Teacher имеет абсолютно стандартную реализацию.
- Все методы в нём совпадают с методами Person.
- Поэтому достаточно просто его определить.

Listing 8: Teacher

```
class Teacher < Person
end
```

# Шаг 9

- У ученика другое обращение.
- Используем полиморфизм для переопределения метода **name**.

Listing 9: Student

```
class Student < Person
  def name
    @first_name
  end
end
```

# Шаг 10

- Класс BadStudent также имеет отличное ото всех обращение с приставкой «Бяка».

Listing 10: BadStudent

```
class BadStudent < Student
  def name
    "Byaka " + @first_name
  end
end
```

# Шаг 11

- Класс Headmaster имеет стандартное обращение по имени–отчеству.
- Значит, метод name переопределять не надо.
- А вот метод head? должен возвращать истину.

Listing 11: Headmaster

```
class Headmaster < Person
  def head?
    true
  end
end
```

# Сложное задание

- Реализовать класс **Двумерный Вектор**.
- Класс имеет два свойства: x-компонента, y-компонента.
- Методы класса:
  1. посчитать длину (модуль)
  2. прибавить к текущему вектору другой
  3. отнять от текущего вектора другой
  4. изменить знак вектора (-вектор)
  5. умножить вектор на скаляр (вещественное число)
  6. скалярно умножить на другой вектор

# Шаг 1

- Прежде всего, определимся, что есть у вектора.
- Двумерный вектор — это два числа (x,y) (аналог радиус–вектора в геометрии).
- Других свойств у вектора нет. Но как хранить эти?
- Два варианта:
  1. В свойствах :x, :y.
  2. В едином свойстве :coords
- Второй вариант универсальнее, поэтому будем использовать его.
- **NB**. Кстати, мы сделаем программу для векторов любой размерности.

# Шаг 2

- В конструкторе **по умолчанию** зададим значение координат в виде пустого массива.

Listing 12: Vector

```
class Vector
  attr_accessor :coords

  def initialize(coords = [])
    @coords = coords
  end
end

v = Vector.new([1, 2])
```

# Шаг 3

- Определим метод подсчёта модуля.
- Модуль вычисляется по теореме Пифагора:

$$\sqrt{\sum_i coords[i]^2}$$

Listing 13: Вычисление модуля

```
class Vector
  ..
  def module
    (@coords.inject(0){|res, elem| res+elem**2})**0.5
  end
end
```

# Шаг 4

- Заметим, что модуль будет вычисляться не только для двумерного вектора, но и для вектора любой размерности.
- Теперь определим операцию сложения.
- При сложении компоненты векторов также складываются.
- **Важно**: операция сложения должна возвращать новый вектор — сумму текущего и того, с которым складываем.
- Сложение делаем покоординатно.

# Шаг 4, часть 2

Listing 14: Сложение

```
def +(v)
  sum = Vector.new
  size = @coords.size-1
  for i in 0..size
    sum.coords[i] = @coords[i] + v.coords[i]
  end
  sum
end
```

# Шаг 4, часть 2, альтернатива

- Для упрощения записи используем метод each_index, который проходит по каждому индексу массива.

Listing 15: Упрощение

```
def +(v)
  sum = Vector.new
  @coords.each_index{|i| sum.coords[i] =
    @coords[i]+v.coords[i]}
  sum
end
```

# Шаг 5

- Операция умножения немного сложнее.
- Если мы умножаем на число, то надо все компоненты вектора умножить на данное число.
- Если же мы умножаем вектор на вектор, то надо возвращать скалярное произведение.
- Как отличить, что нам передаётся в качестве аргумента?
- Используем метод **class**, который возвращает строку с названием класса.

# Шаг 5, часть 2

Listing 16: Умножение

```
def *(v)
  if (v.class == Vector)
    product = 0
    @coords.each_index{|i| product+=
      @coords[i]*v.coords[i]}
  else
    product = Vector.new
    @coords.each_index{|i| product.coords[i] =
      @coords[i]*v}
  end
  product
end
```

# Шаг 6

- Аналогично делаем остальные методы.
- Например, метод «отрицание».

Listing 17: Отрицание

```
def -@
  self.coords = self.coords.map{|i| -i}
end
```

# Шаг 7

Listing 18: Упрощение

```
def -(v)
  sum = Vector.new
  @coords.each_index{|i| sum.coords[i] =
    @coords[i]-v.coords[i]}
  sum
end
```

# Мысль

Класс Vector очень и очень похож на класс Array

# Расшифровка мысли

- Класс Vector, как мы определили на шаге 1, имеет только одно свойство — координаты.
- Координаты представляют собой массив.
- Раз всё в ruby — объекты, значит, и массивы тоже.
- А, значит, логично было бы вместо определения класса Vector, отнаследовать его от массива.

Listing 19: Вектор vs Массив

```
class Vector < Array
end
```

# Работа с массивом

- Теперь мы можем обращаться к i-ой координате внутри класса так: **self[i]**.
- Пример метода:

Listing 20: Вектор как массив

```
class Vector < Array
  def +(a)
    sum = Vector.new
    self.each_index{|k| sum[k] = self[k]+a[k]}
    sum
  end
end
```

# Создание векторомассива

Listing 21: Создание

```
v1 = Vector.new[1,2,3]
v2 = Vector.new[3,4,5]
v3 = v1+v3
puts v3.inspect
```

# References

- При подготовке данного материала использовались сайты: http://ru.wikibooks.org/wiki/Ruby, http://rubydev.ru, http://en.wikipedia.org, http://ruby-lang.org.
- Все презентации доступны на http://school.smirik.ru!
- Вопросы, предложения, д/з: smirik@gmail.com