Инкапсуляция и полиформизм

Информатика
10-11 классы

28 февраля 2012 г.

Разбор задач.

- **Задача 1**. Написать класс Прямоугольник — наследник Polygon. Определить в нём метод подсчёта площади. Проверить корректность его работы.
- Самым простым способом подсчёта площади является перемножение длинной стороны прямоугольника на короткую. Данные о сторонах мы имеем в свойстве sides, поэтому задача становится весьма несложной.

Задача 1

Listing 1: Задача 1

```
class Polygon
  ...
end

class Rectangle < Polygon
  def square
    @square = @sides[0]*@sides[1]
  end
end

r = Rectangle.new
r.sides = [10,2,10,2]
puts r.square
```

Задача 2

- **Задача 2.** Написать в классе Прямоугольник метод, определяющий, является ли прямоугольник квадратом. Метод должен возвращать булевский ответ. Проверить корректность работы метода.
- Вспомним, что булевский ответ — это истина или ложь. В качестве правил хорошего тона булевские методы следует оканчивать на знак вопроса.
- Назовём наш метод square?.
- **Алгоритм:** прямоугольник является квадратом, когда все его углы и стороны равны между собой. Достаточно проверить три угла, так как чётвёртый получается вычитанием из 360.

Решение задачи 2

Listing 2: Задача 2

```
class Rectangle < Polygon
  ...
  def square?
    if ( (@sides[0] == @sides[1]) &&
         (@sides[1] == @sides[2]) &&
         (@sides[2] == @sides[3]) &&
         (@corners[0] == 90) &&
         (@corners[1] == 90) &&
         (@corners[2] == 90)
         )
      true
    else
      false
    end
  end
end
```

Задача 3

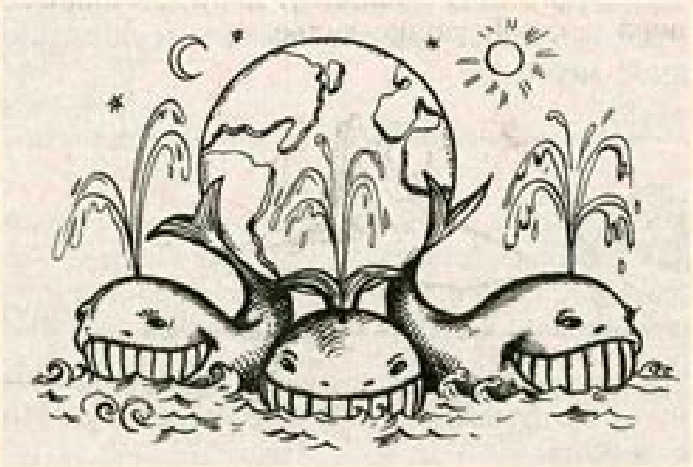
- **Задача 3**. Создать в классе Треугольник метод, проверяющий, является ли данный треугольник прямоугольным. Проверить корректность работы метода.
- **Алгоритм:** треугольник является прямоугольным, если выполнено условие теоремы Пифагора: сумма квадратов катетов равна квадрату гипотенузы.
- Для быстрого определения, какая сторона самая большая, используем метод sort для массива сторон.

Решение задачи 3

Listing 3: Задача 3

```
class Triangle < Polygon
  ...
  def rectangular?
    sides = @sides.sort
    if (sides[2]**2 == (sides[0]**2 + sides[1]**2))
      true
    else
      false
    end
  end
  ...
end
```

Три кита ООП



- Инкапсуляция
- Наследование
- Полиморфизм

- Инкапсуляция
- ~~Наследование~~
- Полиморфизм

Инкапсуляция

Инкапсуляция

- Объектно–ориентированное программирование позволяет использовать парадигму чёрного ящика для сокрытия логики приложения.
- Написав однажды какой-либо метод, нет смысла впоследствии вникать в его содержимое.
- Более того, другие программисты могут вообще не знать реализацию конкретного метода, но вполне уметь его использовать.
- Такой подход в объектно-ориентированном программировании называется *инкапсуляция*.

Пример с уравнением ax + b = c

Listing 4: Инкапсуляция

```
class LinearEquation
  attr_accessor :a, :b, :c
  def initialize(a,b,c)
    @a = a
    @b = b
    @c = c
  end
  def solve
    if (@a == 0)
      return "any" if (@b == @c)
      return "no solutions"
    else
      x = (@c - @b) / @a
    end
  end
end
```

Разбор кода

- В этом коде были использованы несколько новых конструкций. Вы можете его не понимать. Но самое важное — он работает, а, значит, в соответствии с принципом инкапсуляции (в данном случае — сокрытия) вы можете его использовать.
- Например, решим уравнение: 2x - 4 = 6.

Listing 5: Используем код

```
eq = LinearEquation.new(2, -4, 6)
puts eq.solve
```

- Итого: инкапсуляция позволяет использовать любой код без необходимости понимать, как оно устроено внутри.

Конструкторы

- В классе LinearEquation мы использовали неизвестный нам ранее метод **initialize**.
- Это — специальный метод. Он называется **конструктор**.
- Конструктор — это метод, который вызывается **при создании** нового объекта.
- Конструкторы используются для автоматизации задач, которые нужно выполнить при создании объекта.
- В нашем примере мы сразу в конструктор передаём исходные данные задачи, чтобы не “забивать” их вручную.
- Для передачи данных в конструктор мы в метод **new** передаём нужные параметры.

Дополнительно об инкапсуляции

- Помимо уже рассмотреного, одной из возможностей инкапсуляции является сокрытие методов.
- Не вдаваясь сейчас в подробности, укажем, что существуют три возможных *видимости* методов:
  1. Публичный метод
  2. Приватный метод
  3. Защищённый метод
- Идея инкапсуляции заключается в сокрытии с помощью видимости тех методов, к которым нежелательно давать доступ программисту. Это позволяет уменьшить количество ошибок в программе.

Полиморфизм

Полиморфизм

- Рассмотрим класс **Человек**. У класса Человек есть свойства *фамилия, имя, отчество* и метод **обратиться по имени**.
- К большинству людей в России принято обращаться по имени–отчеству.
- Однако к школьникам, обычно, обращаются по имени.
- Итого, один и тот же метод для разных классов имеет разные реализации.
- Возможность похожих классов (например, наследников) иметь различную реализацию одного и того же метода называется **полиморфизмом**.

Пример полиморфизма

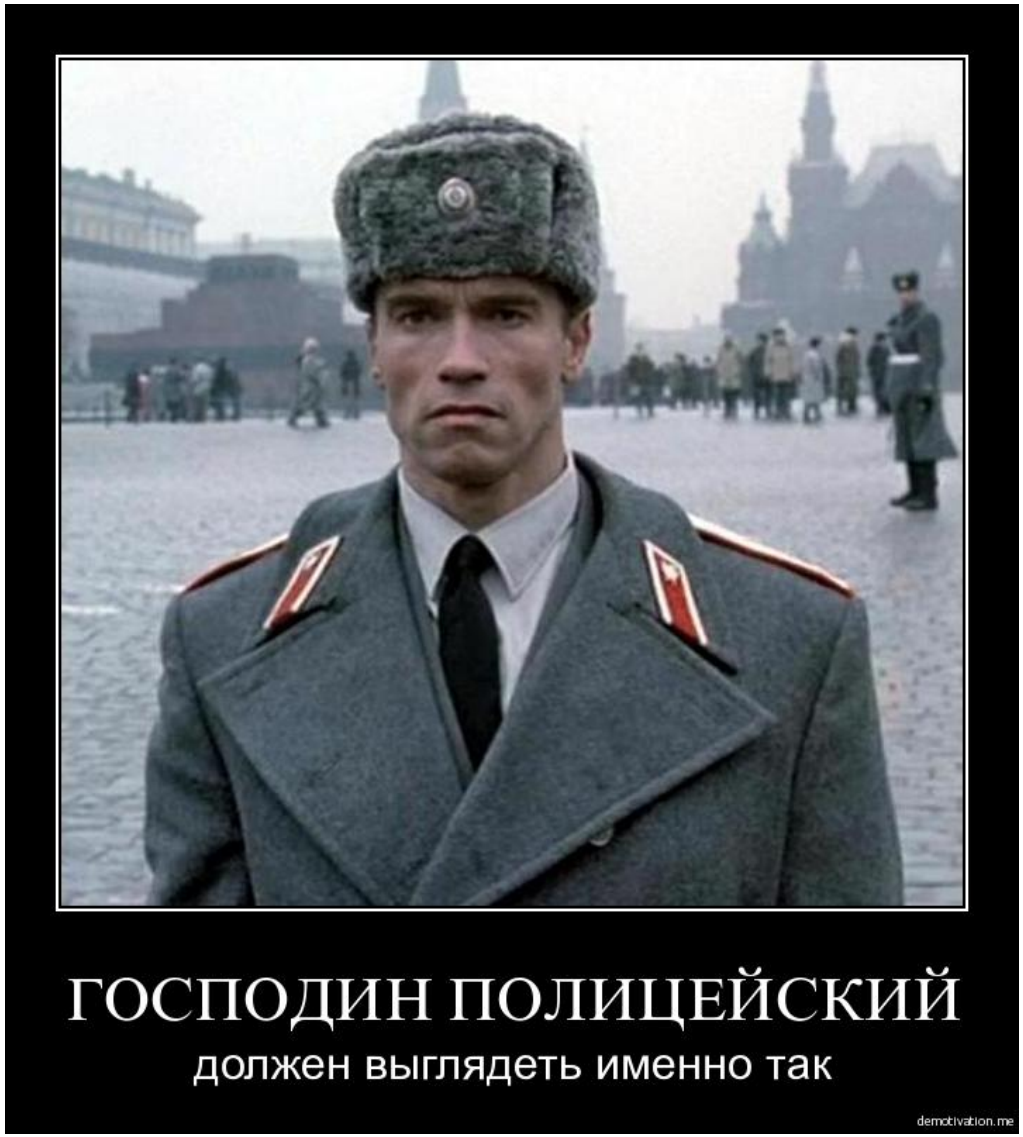
Listing 6: Полиморфизм

```
class Person
  attr_accessor :first_name, :last_name, :middle_name
  def getName
    @first_name + ' ' + @middle_name
  end
end

class Teacher < Person
end

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

Polizei


ГОСПОДИН ПОЛИЦЕЙСКИЙ
должен выглядеть именно так
demotivation.me

Пример полиморфизма

Listing 7: Полиморфизм

```
class Polizei < Person # really Person???
  def getName
    'Herr␣Polizei'
  end
end

p = Polizei.new
puts p.getName
```

Для чего нужен полиморфизм?

- С помощью полиморфизма можно переопределять методы родительского класса.
- Часто имеется следующая ситуация: в 90% случаев методы наследников полностью идентичны. В этом случае общий метод выносят в класс–родитель, чтобы не дублировать код.
- Однако в 10% случаев есть необходимость по-другому реализовать метод.
- Чтобы не вставлять в метод проверки и условия, используют полиморфизм, переопределяя метод только там, где нужно.
- **Самостоятельное изучение**. Перегрузка методов, перегрузка / переопределение операций.

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?.

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

References

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