Разделение функциональности: наследование, модули, примеси

Андрей Васильев

2017

Принцип DRY

Do not Repeat Yourself - при разработке программного обеспечения необходимо устранять ненужное дублирование

Если логика изменится, то нужно исправить только в 1 месте

Классы

Классы описывают общую логику действий для множества различных объектов: методы класса доступны всем объектам

Примеры задач с общей логикой

- Разработка приложения, работающего с поставкой товаров. У каждого типа доставки есть общая логика, например, вычисление веса и габаритов посылки
- Разработка приложения, вычисляющего налоговые отчисления для заказов, товаров, услуг

В каждом из этих случаев следует избегать дублирования

Наследование и сообщения

Механизм наследования позволяет создавать новые версии класса, являющиеся его уточнением или специализацией

- Новые классы называются наследниками (подкласс)
- Исходный класс называется родителями (суперкласс)
- При наследовании подкласс наследует все возможности суперкласса - все методы доступны

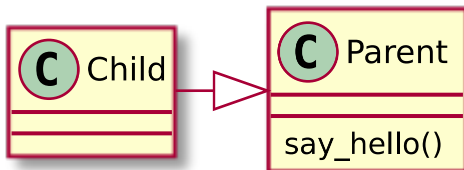

Рис.: Пример наследования

Синтаксис наследования

```
class Parent
  ...
end
class Child < Parent
end
```

- Для описания наследования используется символ <
- Символ говорит о том, что класс слева является специализированной версией класса справа
- Класс `Child` является подклассом `Parent`
- Все методы суперкласса доступны в подклассе
- `self` содержит ссылку на объект, на котором вызван метод

Первый пример в листинге.

Дерево наследования

- Для исследования дерева наследования можно воспользоваться методом `superclass`
- Для подкласса данный метод возвратит имя суперкласса
- Если нет явного суперкласса, тогда им является `Object`
- Суперклассом для `Object` является `BasicObject`, у которого нет суперкласса

Второй пример в листинге.

Последовательность вызова метода

- Если метод есть у класса, вызывать его
- Если метод есть у суперкласса, вызвать его
- Если метод есть у суперкласса суперкласса…

Перекрытие методов в подклассах

- Вы можете в подклассе создать метод с именем метода из суперкласса
- Этот метод будет вызван раньше метода из суперкласса и перекроет оригинальную реализацию
- Если семантика метода будет сохранена, тогда вызывающий класс не отличит оригинальный метод от подмены
- Для вызова метода из суперкласса используйте `super`

Метод to_s класса `Object`

- Метод преобразует объект к строке, что является стандартным подходом, используемый строками
- Можно реализовать свой метод `to_s`, реализующий логичное преобразование, перекрывающий стандартный

Третий пример в листинге.

Примера в листинге нет

Проблема роста приложений

При написании приложения разработчики всегда сталкиваются с проблемой роста приложения.

- Необходимо придумывать уникальные имена классов
- Необходимо логически объединять функции
- Необходимо логически объединять классы

В `Java` *проблема решается обязательным выделением пакетов*

Подходы к решению проблемы

- Правильное распределение элементов по файлам
- Использование классов для описания сущностей
- Объединение классов и функций в модули

Модули в языке Ruby

Модуль - замкнутое именованное пространство

```
module Trig
  PI = 3.141592654
  def Trig.sin(x) # Определяем метод модуля
    # ..
  end
  def Trig.cos(x)
    # ..
  end
end

puts Trig::PI # Доступ к константе
puts Trig.sin(Trig::PI/4) # Вызов метода
```

Пятый пример в листинге.

- Для доступа к константам, определённым внутри метода используются два двоеточия `::`
- Для вызова метода модуля используйте точку

Определение методов

Определение методов модулей похоже на определение методов классов (не методов экземпляров классов). Можно даже использовать синтаксис с ключевым словом `self`

```
module ModuleWithBigName
  def self.info
    'I am working'
  end
end

puts ModuleWithBigName.info
```

Структура джема

В каталоге `lib` обычно располагается один файл с названием, соответствующим названию джема. В самом файле определена система из вложенных модулей. Все классы, помещённые в данные модули, представляют собой публичную часть интерфейса модуля.

Mixin (Примеси, Миксины, Агрегация)

Модули позволяют решить вопросы множественного наследования, реализуя возможность агрегирования: модули можно примешивать к определениям классов

```
module Debug
  def who_am_i?
    "#{self.class.name} (id: #{self.object_id})"
  end
end
class Test
  include Debug
end
Test.new().who_am_i?
```

- Методы модуля становятся методами экземпляра класса
- Модуль включается с помощью метода `include`

Особенности примеси модулей

- `include` не подключает файл, а только ссылается на описанный ранее модуль
- `include` не копирует методы из модуля в класс, а только лишь ссылку
- Если несколько классов примешивают в себя классы, тогда они все ссылаются на один и тот же модуль
- При изменении модуля, изменяются все классы, которые уже включили его в себя

Шестой пример в листинге.

Пример с примесью Comparable

Другие схемы наследования

Множественное наследование (C++)

- У класса может быть множество родительских классов
- Каждый из родительских классов может полноценно реализовывать свою логику

Один класс-родитель (Java < 8, C#)

- У класса может быть только один родительский класс
- Класс может реализовывать множество интерфейсов

В Java 8 интерфейсы могут содержать код, который может выполняться. Это открывает возможности по использованию их в качестве примесей, однако такой практики пока ещё не сложилось.

Проблема наследования от классов, имеющих общего предка

Предположим, что у нас есть некоторый базовый класс.

Мы наследуем от него два других класса.

От этих двух классов наследуем третий класс. Получили “ромб”

Вопросы:

- Сколько должно быть экземпляров переменных, определённых в базовом классе?
- В какой момент времени должен быть вызван конструктор базового класса?
- Какие методы из методов родителей должны быть вызваны при обращении к родительскому классу?

Использование примеси Enumerable

Модуль `Enumerable` предоставляет большое число методов для работы с данными, которые можно перечислить

Классу, использующему модуль Enumerable необходимо реализовать следующие методы:

- итератор `each` для обхода всех элементов коллекции
- объекты должны реализовывать метод сравнения <=>

После примеси модуля становятся доступными все его методы:

- `map`
- `find`
- `collect`
- ...

Проектирование модулей

Методы, примешанные к классу, могут взаимодействовать не только с методами, но также и с переменными экземпляра

```
module First
  def foo
    @abc = 10
  end
end
module Second
  def bar
    @abc = 'abc'
  end
end
```

Не следует разрабатывать модулей, которые оперируют состоянием переменных экземпляра напрямую

Какой метод будет вызван?

Когда у вас появляется возможность использовать примеси, то должен встать вопрос: как происходит поиск методов среди всех связных компонент? Порядок достаточно простой:

- Методы объекта
- Методы включённых модулей
- Методы суперкласса
- Методы включённых в суперкласс модулей
- ...

Использование примесей и наследования

Механизм наследования позволяет определить чёткую иерархию свойств объектов. Для любого класса-ребёнка должно выполняться правило: класс-ребёнок является классом-родителем. Т.е. в программе можно заменить класс-родитель на класс-ребёнок и приложение должно продолжить правильно функционировать.

В большинстве случаев реального мира мы имеем дело с предметами, обладающими множеством различных свойств и множеством контрагентов. То есть объект живёт в некоторой среде и комбинирует в себе качества множества различных компонент. Такое поведение обычно сложно представить в виде подмножества какого-то конкретного класса.

Простое правило при проектировании классов: «композиция лучше наследования»