Técnicas Avanzadas de Programación – UTN – FRBA
1 Cuatrimestre 2015
Ejercicio Metaprogramación: Metabuilder

Dominio

Se desea poder crear un builder para crear builders (o sea, un metabuilder). La idea general para todos los puntos es poder indicarle a un objeto builder como se debería comportar otro builder genérico y luego obtener una instancia del mismo para aplicarlo en el dominio particular que corresponda. El metabuilder, entonces, será nuestro framework de builders que es agnóstico de los dominios de negocio, mientras que los builders generados por este estarán particularizados para un dominio.

1. El metabuilder

Se desea poder crear un metabuilder al cual se le defina cuales son las propiedades del objeto a construir y la clase del mismo. Es importante destacar que el builder de dominio sólo debe permitir configurar las propiedades declaradas. Cualquier otra propiedad que se le quiera configurar se debe considerar como un error y debe tirar una excepción en algún momento.

```
class Perro
  attr_accessor :raza, :edad, :duenio

  def initialize
    @duenio = "Alguien"
  end
end
```

```
it 'puedo crear un builder de perros' do
  metabuilder = Metabuilder.new
  metabuilder.set_target_class(Perro)
  metabuilder.add_property(:raza)
  metabuilder.add_property(:edad)

  builder_de_perros = metabuilder.build
  builder_de_perros.raza = "Fox Terrier"
  builder_de_perros.edad = 4
  perro = builder_de_perros.build

  expect(perro.raza).to eq("Fox Terrier")
  expect(perro.edad).to eq(4)
  expect(perro.duenio).to eq("Cesar Millan")
end
```

2. Sintaxis del metabuilder

Considerando el punto 1, se desea extender el Metabuilder para que soporte la siguiente sintaxis (la sintaxis anterior debe seguir siendo soportada).

```
it 'soporta configuracion por bloques' do
  metabuilder = Metabuilder.config {
    target_class Perro
    property :raza
    property :edad
  }

  builder_de_perros = metabuilder.build
  builder_de_perros.raza = "Fox Terrier"
  builder_de_perros.edad = 4
```

```
  perro = builder_de_perros.build

  expect(perro.raza).to eq("Fox Terrier")
  expect(perro.edad).to eq(4)
end
```

3. Validaciones

Se desean incorporar validaciones, las cuales se ejecutan en el momento de crear la instancia. Si no se cumplen todas las validaciones, entonces no se debe de crear la instancia y el builder debe tirar una excepción. Tener en cuenta que dentro de las validaciones se debe poder hacer referencia a cualquiera de las propiedades definidas, simplemente escribiendo el nombre de las mismas.

```
it 'puedo definir validaciones que rompen' do
  metabuilder = Metabuilder.config {
    target_class Perro
    property :raza
    property :edad
    validate {
      ["Fox Terrier", "San Bernardo"].include?(raza)
    }
    validate {
      edad > 0 && edad < 20
    }
  }

  builder_de_perros = metabuilder.build
  builder_de_perros.raza = "Fox Terrier"
  builder_de_perros.edad = -5
  expect {
    builder_de_perros.build
  }.to raise_error ValidationError
end
```

4. Comportamiento

Para este punto se quiere poder agregar ciertos comportamientos según una condición dada. La idea es poder agregar dichos comportamientos solamente a la instancia construida, si cumple la condición.

```
it 'agrega metodos cuando se cumple la condicion' do
  metabuilder = Metabuilder.config {
    target_class Perro
    property :raza
    property :edad
    conditional_method(:caza_un_zorro,
        proc {
          raza == "Fox Terrier" && edad > 2
        },
        proc {
          "Ahora voy #{duenio}"
        }
    )
  }

  builder1 = metabuilder.build
  builder1.raza = "Fox Terrier"
  builder1.edad = 3
  fox_terrier = builder1.build

  expect(fox_terrier.caza_un_zorro).to eq("Ahora voy Cesar Millan")

  builder2 = metabuilder.build
  builder2.raza = "San Bernardo"
  builder2.edad = 3
  san_bernardo = builder2.build

  expect {
    san_bernardo.caza_un_zorro
  }.to raise_error(NoMethodError)
end
```