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 = "Cesar Millan"
  end
end
```

```
it 'puedo crear un builder de perros' do
    #El  cdigo  del metabuilder en si va ac

    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

De acuerdo con lo ya realizado en el punto 1, queremos expandir nuestro ejemplo del metabuilder pudiendo setar los atributos de builder ya definidos en nuestro metabuilder de manera que se le mande un mensaje directamente a nuestra instancia del builder, de la siguiente manera:

```
it 'soporta seteo de los atributos mandando un mensaje al metabuilder' do

  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
```

Hay que dejar el motor de manera que pueda seguir soportándose la sintaxis anterior.

3. Validaciones

Ahora se desea incorporar las validaciones a nuestro modelo del metabuilder, las cuales se ejecutan en el momento de crear la instancia. Si no se cumplen todas las validaciones declaradas, no se debería crear la instanci y el builder debería lanzar una excepción. Dentro de dichas validaciones se puede hacer referencia a cualquiera de las propiedades definidas en la clase.

```
it 'puedo definir validaciones que rompen' do
    #codigo del metabuilder previo
    metabuilder.validate {
      ["Fox Terrier", "San Bernardo"].include?(raza)
    }
    metabuilder.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. Definir Metabuilders de clases no declarads

Se desea que ahora nuestro metabuilder pueda entender ahora el caso en el que querramos crear un builder de una clase que aún no se creo y que nuestro metabuilder cree la clase y genere un builder de esta. Para ello se pasará a un método el nombre de la clase que se desea crear y la superclase que posee y opcionalmente un bloque que se evaluará en el contexto de la clase.

```
it 'Puedo definir un Metabuilder de clases que aun no existen' do
  metabuilder = Metabuilder.new
  metabuilder.set_target_class_hierarchy :Gato, nil, false do
    attr_accessor :raza, :pelaje
  end
  # Ms  declaraciones del builder ac

  builder_gato = metabuilder.build
  builder_gato.raza = 'Siames'
  builder_gato.pelaje = 'corto'
end
```

5. 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.new
    #  Ms  declaraciones del metabuilder
    metabuilder.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
```

6. Mejorar la manera de definir una clase no definida

Este requerimiento permite que exista otra manera de crear clases que no han sido definidas aún, una que sea algo más natural y que no dependa en sí de la sintaxis de Ruby, para ello, vamos a tener una sintaxis del siguiente tipo:

```
    metabuilder = Metabuilder.new
    metabuilder.set_target_class_hierarchy :Gato, nil do
      add_attributes :raza, :pelaje
      add_method :saludar do
        'Miau'
      end
    end
    # Ms  definiciones del metabuilder

    builder_gato = metabuilder.build
    builder_gato.raza = 'Siames'
    builder_gato.pelaje = 'corto'
    gato = builder_gato.build

    expect(gato.raza).to eq('Siames')
    expect(gato.saludar).to eq('Miau')
```

Se debe modificar el motor de manera tal que se pueda seguir soportando la sintaxis previamente desarrollada