Técnicas Avanzadas de Programación – UTN – FRBA
1 cuatrimestre 2014
Ejercicio Metaprogramación: Metabuilder

Dominio

Se desea poder crear un builder para crear builders (o sea, un meta builder). 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 meta builder, 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.

Este ejercicio fue un parcial y se utilizará como ejercicio de preparación para el parcial de lengujes dinámicos.

1 El meta builder

Se desea poder crear un meta builder 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.

El test para el primer punto es:

```
builder_de_perros = Metabuilder.new
    .set_target_class(Perro)
    .add_property(:raza)
    .add_property(:edad)
    .add_property(:peso)
    .build

 ###-----------------------------------
 builder_de_perros.raza = 'Fox terrier'
 builder_de_perros.edad = 4
 builder_de_perros.peso = 14
 perro = builder_de_perros.build

 ###-----------------------------------
 perro.raza.should == 'Fox terrier'
 perro.edad.should == 4
 perro.peso.should == 14
```

2 Sintaxis del meta builder

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

```
  builder_de_perros = Metabuilder.build {
    property(:raza)
    property(:edad)
    property(:peso)
    target_class(Perro)
  }

  ###-----------------------------------

  builder_de_perros.raza = 'Fox terrier'
  builder_de_perros.edad = 4
  builder_de_perros.peso = 14
  perro = builder_de_perros.build
```

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 una 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 la misma.

```
builder_de_perros = Metabuilder.build {
  property(:raza)
  property(:edad)
  property(:peso)
  target_class(Perro)
  validate {
    ['Fox terrier', 'salchicha', 'chihuahua'].include? raza
  }
  validate {
    edad > 0 && edad < 20
  }
}

###---------------------------------------

builder_de_perros.raza = 'Fox terrier'
builder_de_perros.edad = 4
builder_de_perros.peso = 14
perro = builder_de_perros.build
```

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 y si cumple la condición. Ejemplo de uso:

```
builder_de_perros = Metabuilder.build {
  property(:raza)
  property(:edad)
  property(:peso)
  target_class(Perro)
  behave_when 'expectativa_de_vida', proc {raza == 'Fox terrier'}, proc {
    20 - edad
  }
  behave_when 'expectativa_de_vida', proc {raza == 'salchicha'}, proc {
   50 - (edad + peso * 2)
  }
  behave_when 'expectativa_de_vida', proc {raza == 'chihuahua'}, proc {
    15
  }
}

builder_de_perros.raza = 'Fox terrier'
builder_de_perros.edad = 4
builder_de_perros.peso = 14
perro = builder_de_perros.build

perro.raza.should == 'Fox terrier'
perro.edad.should == 4
perro.peso.should == 14
perro.expectativa_de_vida.should == 16
```