

MGL7460 : Laboratoire #7

Définition d'une API coulante (DSL interne) Ruby pour des recettes

1^{er} décembre 2016

Le but de ce laboratoire est de vous familiariser la définition de **DSL internes sous forme d'API coulantes** (*fluent interface*), qui utilisent soit du chainage de méthodes (comme de nombreux DSL Java), soit des blocs (comme de nombreux DSL Ruby, e.g., `gli`, `Rake`, `gemspec`, etc.).

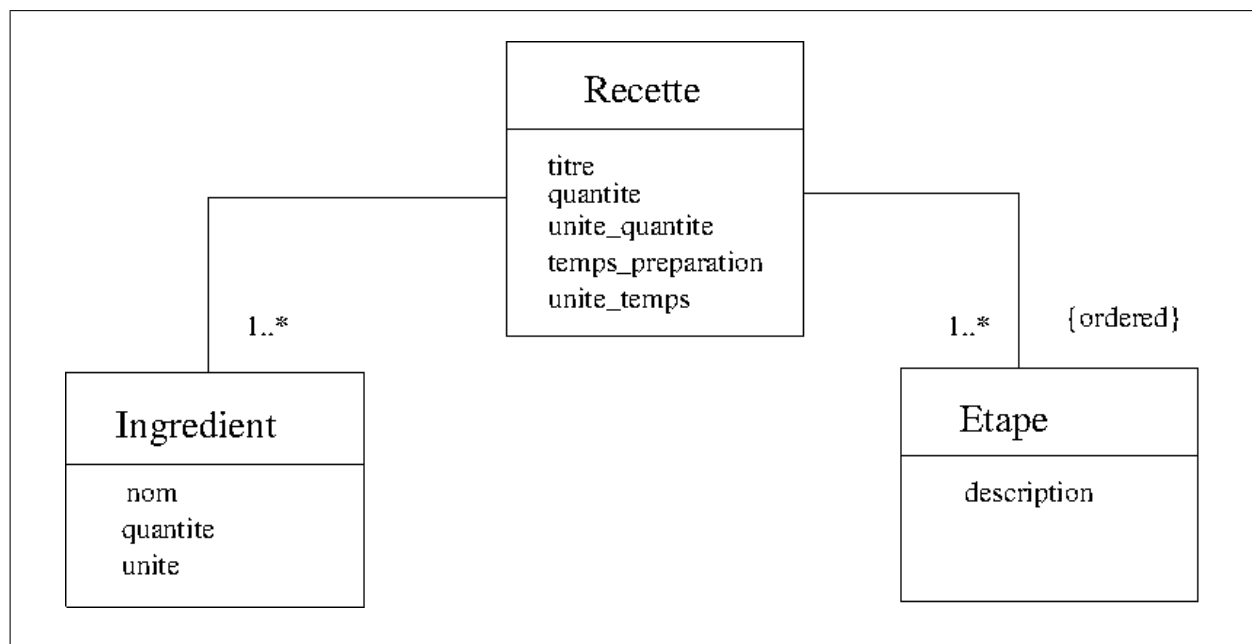


Figure 1: Modèle UML simplifié de classes permettant de définir des recettes.

La Figure 1 présente un diagramme de classes UML modélisant des recettes de cuisine : titre de la recette, quantité résultante (par ex., 4 portions, 4 personnes, 4 tasses, etc.), temps de préparation (par ex., 60 minutes, 1 heure), ensemble d'ingrédients (nom, quantité, unité de mesure) et liste (ordonnée!) des étapes de préparation (de simples chaînes).

Dans les fichiers qui vous sont fournis (voir plus bas), un fichier `soupe-oignon-new.rb` contient la description d'une recette de «soupe à l'oignon», et ce en utilisant directement les constructeurs de base. Pour cette recette, la définition est donc la suivante — notez qu'elle ne serait pas correcte si on interchangeait certaines lignes, puisque l'ordre des éléments dépend de l'ordre des arguments de la méthode `Recette.initialize` :

```
soupe_oignon = Recette.new( "Soupe a l'oignon",
  4, Portion,
  45, Minute,
  [Ingredient.new( "oignons", 4 ),
   Ingredient.new( "bouillon de poulet", 3, Tasse ),
   Ingredient.new( "beurre", 4, Cuillere_a_soupe )],
  ],
  [Etape.new( "Peler les oignons"),
   Etape.new( "Faire sauter les oignons dans le beurre"),
   Etape.new( "Ajouter le bouillon de poulet"),
   Etape.new( "Faire mijoter 30 minutes")],
  ]
)
```

Une telle description est complexe et monolithique, lourde à lire et à comprendre. On désire donc, à l'aide d'une **API coulante**, définir un DSL interne qui va permettre de spécifier de façon plus simple, claire et élégante de telles recettes.

De plus, on veut aussi s'assurer qu'une (instance de) `Recette` soit correcte et complète. Par exemple, pour qu'une recette puisse être affichée avec `to_s`, il faut qu'elle soit `complete?` (pré-condition de la méthode `to_s`).

Pour ce faire, un objet de la classe `Recette` possède donc une variable d'instance `@complete` et une méthode `complete?`. Ces deux éléments permettent conjointement de déterminer si une recette est `complete?` ou pas — c'est-à-dire si tous les champs ont été définis et sont d'un type correct. La valeur de cet attribut peut être examinée — mais aussi implicitement définie — avec la méthode `complete?` — voir `lib/recettes/recette.rb`.

1 Ce qui vous est fourni

Pour obtenir une copie des fichiers fournis :

```
$ git clone http://www.labunix.uqam.ca/~tremblay/git/Recettes.git
```

L'organisation des répertoires et fichiers obtenus est présentée à la Figure 2.

```
$ tree Recettes
Recettes
|-- lib
|   |-- dbc.rb
|   |-- recettes
|       |-- etape.rb
|       |-- ingredient.rb
|       |-- recette-dsl-bloc.rb
|       |-- recette-dsl-chainage.rb
|       |-- recette.rb
|       '-- unite.rb
|-- '-- recettes.rb
|-- makefile
|-- soupe-oignon-bloc.rb
|-- soupe-oignon-chainage.rb
|-- soupe-oignon-new.rb
'-- spec
    |-- dsl_bloc_spec.rb
    |-- dsl_chainage_spec.rb
    |-- ingredient_spec.rb
    |-- recette_spec.rb
    |-- spec-helper.rb
    '-- unite_spec.rb
```

Figure 2: La structure du répertoire qui vous est fourni.

Les principaux répertoires et fichiers sont les suivants :

- `lib` : Répertoire contenant le code source, donc les définitions des classes fournies. Ces classes sont définies à l'intérieur du module `Recettes`, d'où la structure de répertoires utilisée (classes définies dans le répertoire `recettes` du répertoire `lib`) :
 - `recettes/etape.rb` : Classe `Etape`.
 - `recettes/ingredient.rb` : Classe `Ingredient`.
 - `recettes/recette-dsl-{bloc,chainage}.rb` : Deux extensions de la classe `Recette` avec **des méthodes que vous devez compléter pour les deux DSL**.

Ces extensions sont **mutuellement exclusives**. Donc, il faut définir correctement la version en cours de traitement en modifiant la constante `DSL` au début du fichier `makefile`, car les deux versions ne peuvent coexister — pour simplifier, certains noms de méthodes sont les mêmes.
 - `recettes/recette.rb` : Classe `Recette` avec le constructeur de base (`initialize`) et les méthodes `complete?` et `to_s`.
 - `recettes/unite.rb` : Classe `Unite` pour les diverses unités de mesure.
 - `recettes.rb` : Contient les `requires` pour toutes les classes du module.
- Des fichiers `soupe-oignon-{bloc,chainage}.rb` qui contiennent la recette de soupe à l'oignon de la page 2, mais exprimée à l'aide de l'une ou l'autre API coulante — voir les exemples plus bas.
- Un répertoire `spec` de tests unitaires, notamment `dsl_{bloc,chainage}_spec.rb`, qui définissent des tests unitaires pour chaque DSL — notamment, un test charge `soupe-oignon-new.rb` et `soupe-oignon-bloc.rb` ou `soupe-oignon-bloc.rb`, les évalue, puis compare les formes textuelles résultantes (produites par un appel à `to_s`).
- Un `makefile` pour faciliter le lancement des tests — cibles `tests_dsl_{chainage,bloc}` — et des exemples d'exécution — `exemple_dsl_{chainage,bloc}`.

Premier exercice : API coulante avec chaînage de méthodes

La première API coulante que vous devez compléter utilise du **chaînage de méthodes**.

Voici la description de la recette de soupe à l'oignon (fichier `soupe-oignon-chainage.rb`), description qui resterait valide **même si on interchangeait certaines des lignes** :

```
soupe_oignon =
  Recette.de( "Soupe a l'oignon" ).
    pour( 4, Portion ).
    requiert( 45, Minute ).
    utilise( "oignons", 4 ).
    et( "bouillon de poulet", 3, Tasse ).
    et( "beurre", 4, Cuillere_a_soupe ).

    pour_debuter( "Peler les oignons" ).
    puis( "Faire sauter les oignons dans le beurre" ).
    ensuite( "Ajouter le bouillon de poulet" ).
    finalement( "Faire mijoter 30 minutes" )
```

Complétez la mise en oeuvre — `lib/recettes/recette-dsl-chainage.rb` — des méthodes permettant de réaliser cette API coulante avec chaînage de méthodes.

L'exemple est dans le fichier `soupe-oignon-chainage.rb` alors que les tests sont dans le fichier `spec/dsl_chainage_spec.rb`.

Les cibles du `makefile` pour cet exercice sont `exemple_dsl_chainage` et `tests_dsl_chainage` (ou `{exemple,tests}_dsl1`).

Remarque : Le seul et unique fichier que vous devez modifier est le suivant :

`lib/recettes/recette-dsl-chainage.rb`.

Vous ne devez donc pas modifier les méthodes déjà existantes du fichier `recette.rb`, sinon le script `soupe-oignon-new.rb` ne fonctionnera plus!

Deuxième exercice : API coulante avec bloc

La deuxième API coulante que vous devez compléter utilise des **blocs**.

Voici la description de la recette de soupe à l'oignon (fichier `soupe-oignon-bloc.rb`), description qui resterait valide **même si on interchangeait plusieurs des lignes** :

```
soupe_oignon = Recette.de( "Soupe a l'oignon" ) do |r|
  r.quantite 4, Portions
  r.temps 45, Minutes # = temps_preparation...

  r.ingredient "oignons", 4
  r.ingredient "bouillon de poulet", 3, Tasse
  r.ingredient "beurre", 4, Cuillere_a_soupe

  r.etape "Peler les oignons"
  r.etape "Faire sauter les oignons dans le beurre"
  r.etape "Ajouter le bouillon de poulet"
  r.etape "Faire mijoter 30 minutes"
end
```

Quelques remarques sur cette API pour décrire des recettes :

- La méthode `temps` sert à définir l'attribut `@temps_preparation` (nom alternatif plus court).
- Pour simplifier l'exemple, seuls des accesseurs **en écriture** (*setters*) sont (partiellement) définis et sont à compléter — fichier `lib/recettes/recette-dsl-bloc.rb`.
- Contrairement à l'exemple avec constructeur et bloc pour les Documents vu dans les diapositives, les méthodes pour définir les attributs d'une `Recette` — donc les accesseurs en écriture de la nouvelle API — **n'utilisent pas** la forme `«r.attr = x»` mais plutôt la forme `«r.attr x»`. L'interface de ces méthodes est donc semblable (en partie) à celle des méthodes pour l'API coulante avec chainage de méthodes pour les Documents (dans les diapositives du cours).

Remarque : Le seul et unique fichier que vous devez modifier est le suivant :

`lib/recettes/recette-dsl-bloc.rb`.

Vous ne devez donc pas modifier les méthodes déjà existantes du fichier `recette.rb`, sinon le script `soupe-oignon-new.rb` ne fonctionnera plus!