MGL7460 : Laboratoire #4
Spécification de «test unitaires» à l'aide de `cucumber` (!?)

27 octobre 2016

Le but du présent laboratoire est donc de vous familiariser avec l'utilisation d'un outil de BDD tel que `cucumber`, outil habituellement utilisé pour spécifier des tests d'acceptation.

Dans le laboratoire #2, nous avons vu comment on pouvait utiliser `MiniTest`, **un outil de tests unitaires**, pour spécifier **des tests d'acceptation**.[1] Dans le présent laboratoire, à l'inverse (et de façon exceptionnelle!), nous allons utiliser `cucumber`, un outil de **tests d'acceptation**, **pour spécifier des tests unitaires**.

Nous allons procéder ainsi pour simplifier le problème, plus précisément, pour éviter d'introduire **à la fois** une interface personne–machine **et** une application qu'il faudrait vérifier.

L'objectif plus spécifique du laboratoire est donc de comprendre le rôles «*steps*», définis dans les fichiers `features/*_steps.rb`, qui font le lien entre la description des *features* — exprimés sous forme de scénarios utilisateurs — avec le code de «l'application» — ici, un simple groupe de classes et module en Ruby. Un autre objectif est aussi de vous familiariser avec l'écriture de clauses de scénarios — `Given`, `When` et `Then`.

En utilisant `cucumber`, vous allez donc devoir développer des tests «unitaires» — exprimés sous forme de scénarios — pour les classes `Emprunt` et `Emprunts` qui, combinées avec le module `EmpruntsTxt`, définissent un *Repository*, tel que décrit dans le document suivant (présenté en classe jeudi 20 octobre) :

`http://www.labunix.uqam.ca/~tremblay/MGL7460/Materiel/repository.pdf`

---

1 Pour obtenir le code à compléter

Pour obtenir les fichiers pour ce laboratoire :

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

[1] Pour ce faire, le programme de tests lançait l'exécution d'un programme externe avec «%x{...}»— ce qui était possible car un langage de script tel que Ruby peut jouer le rôle de *glue language* — puis analysait le résultat émis sur `STDOUT` par l'exécution du programme (résultat émis, donc analysé, sous forme d'une chaine de caractères).

2 Les principaux fichiers

Le répertoire `features` contient les fichiers présentés dans le Tableau 1.

Note : Dans ce qui suit, toutes les références à des noms de fichier — sauf pour le fichier `Rakefile` — se feront **relativement** au répertoire `features`

`emprunt-to_s.feature`
`emprunt-perte.feature`
`emprunt-spaceship.feature`
	Scénarios pour la classe `Emprunt`

`emprunts_txt.feature`	Scénarios pour la classe `EmpruntsTxt`

`emprunts-ouvrir.feature`
`emprunts-ajouter-supprimer.feature`
`emprunts-selectionner.feature`
	Scénarios pour la classe `Emprunts`

`assertions.rb`	Assertions à utiliser dans les postconditions

`emprunt_steps.rb`
`emprunts_txt_steps.rb`
`emprunts_steps.rb`
`support/hooks.rb`
	Steps pour `Emprunt`
	Steps pour `EmpruntsTxt`
	Steps pour `Emprunts`
	Idem

Table 1: Les divers fichiers du répertoire `features`.

Les fichiers que vous devez modifier/compléter sont les suivants :

a. `emprunt_steps.rb`

b. `emprunts-selectionner.feature`
   `emprunts_steps.rb`

c. `emprunts-ajouter-supprimer.feature`
   `emprunts_steps.rb`

Les assertions que vous pouvez utiliser dans les post-conditions sont les suivantes, définies dans le fichier `assertions.rb` :

```
def assert_equal( expected, obtained, msg = nil )
def assert( condition, msg = nil )
def refute( condition, msg = nil )
```

3 Ce qu'il faut faire

a. Complétez le fichier `emprunt_steps.rb` pour que les scénarios du fichier `emprunt-perte.feature` s'exécutent avec succès.

   Remarques/indices sur la façon de procéder :

   - Lancez l'exécution des tests courants (`WIP` = *Work In Progress*), i.e., ceux pour l'exercice `a` — initialement, «`rake`» == «`rake exercice_a`» puisque `:WIP => :exercice_a`.
   - `cucumber` vous indiquera les éléments de scénario qui n'ont pas de mises en oeuvre associées (un `When` et un `Then`). Ajoutez les définitions suggérées par `cucumber` dans le fichier `emprunt_steps.rb`.
   - Reexécutez `cucumber` pour assurer que les nouvelles étapes sont bien `pending` — donc qu'elles *matchent* correctement le texte des scénarios.
   - Complétez la mise en oeuvre de ces deux nouvelles étapes en tenant compte que les emprunts créés par les clauses `Given` sont conservés dans `@emprunts_given`, un `Hash`, la clé utilisée étant le nom (`String`) utilisé pour identifier l'emprunt dans le scénario.

     Par exemple, `"e1"` dans le scénario suivant (`emprunt-perte.feature`) :

     ```
     Given un emprunt "e1" fait par "Guy" ["@"] pour "Code Complete" ...
     When j'indique que l'emprunt "e1" est perdu
     Then l'emprunt "e1" est considere comme etant perdu
     ```

b. Complétez les fichiers `emprunts-selectionner.feature` et `emprunts_steps.rb` pour que les scénarios s'exécutent avec succès.

   Plus précisément, vous devrez :

   (a) Modifiez le `Rakefile` pour que la cible par défaut soit `exercice_b`.

   (b) Complétez la spécification d'une (1) clause `Then` de l'un des scénarios (dans `emprunts-selectionner.feature`), puisqu'initialement la clause indique uniquement «`Then A completer!`».

   (c) Donnez la mise en oeuvre des trois (3) clauses `Then`.

   Remarques/indices :

   - Le *repository* créé par la clause `Before` (implicite : cf. `support/hooks.rb`) est conservé dans la variable `@emprunts_repository`, une instance de la classe `Emprunts`.
   - Lorsqu'une action de sélection est exécutée, le ou les emprunts sélectionnés sont conservés dans `@selection`.
   - La méthode `Emprunts#selectionner` retourne, par défaut, une collection (`Array`) d'objets `Emprunt`. Par contre, on peut spécifier une option qui permet de ne sélectionner **qu'un unique** `Emprunt` — mais uniquement si on est certain qu'il ne peut y en avoir qu'un seul.

     Plus précisément, si `emps.kind_of?(Emprunts)`, alors :

```
# Si foo(x) peut etre vrai pour 0, 1 ou plusieurs emprunts.
# Note: Si foo(e) est toujours faux, le resultat est [].
emps.selectionner { |e| foo(e) }.kind_of?(Array)

# Si foo(x) est vrai pour *exactement* un (1) emprunt.
emps.selectionner( unique: true ) { |e| foo(e) }.kind_of?(Emprunt)
```

c. Complétez les fichiers `emprunts-ajouter-supprimer.feature` et `emprunts_steps.rb` pour que les scénarios s'exécutent avec succès.

   (a) Modifiez le `Rakefile`.

   (b) Complétez la spécification d'un `When` et d'un `Then` dans `emprunts-ajouter-supprimer.feature`

   (c) Donnez la mise en oeuvre des deux `When` et du `Then`.