INF600A - Rapport pour le TP3

Présenté par :
Olivier Rochefort - ROCO15107807
Mathieu Lavoie - LAVM31019201

Pour :
Guy Tremblay

Proposition

Notre application, nommée "gt" (pour "Gestion Taux"), est un gem qui gère une base de données de devises monétaires. Celle-ci peut modifier et consulter la base de données. Pour modifier, les commandes disponibles sont "ajouter" et "supprimer", alors que pour questionner nous avons "lister" et "taux_devise". Cette collection de commandes permet d'ajouter les différentes devises, les lister, et surtout, demander le taux de conversion d'une devise vers une autre.

Format de la base de données locale

<devise>;<devise conversion 1>;<devise conversion 2>;etc…

Par exemple :

CAN;USD:0.75364;EUR:0.66268;GBP:0.58735
USD;CAN:1.32664;EUR:0.87928;GBP:0.77922
etc...

Pour mieux se comprendre voici la terminologie utilisée dans notre document et dans le code :

Exemple d'entrée dans la bd : CAN;USD:0.75364;EUR:0.66268	
CAN	Devise. Le nom est une chaîne d'exactement trois lettres.
USD:0.75364	Devise de conversion. Devise rattachée à la devise "principale". Autrement dit, une devise de conversion indique combien vaut la devise principale une fois convertie en "devise conversion". Ici, 1$ CAN vaut 0.75364 dollars USD. Le nom de devise est une chaîne d'exactement trois lettres et le taux un nombre décimal avec au moins un chiffre après le point.

Commandes

Tel que demandé, voici d'abord la sortie de la commande help. Suivent des explications plus détaillées pour chaque commande.

```
NAME
  gt - Un programme pour la gestion de taux de change

SYNOPSIS
  gt [global options] command [command options] [arguments...]

VERSION
  0.0.1

GLOBAL OPTIONS
  --depot=arg           - Depot de donnees a utiliser pour les taux (default:
                        .taux.txt)
  --help                - Show this message
  --[no-]stdin, --[no-] - Utilisation de stdin plutot que le depot
  --version             - Display the program version

COMMANDS
  Ajouter     - Ajoute une devise ainsi que ses differentes devises de
                conversion au depot
  Help        - Shows a list of commands or help for one command
  Init        - Cree une nouvelle base de donnees (vide) pour gerer des taux
                (dans './.taux.txt' si --depot n'est pas specifie)
  Lister      - Liste toutes les devises du depot
  Supprimer   - Supprime une devise (et ses devises de conversion) du depot
  Taux_devise - Retourne la valeur d'une devise par rapport a une autre
```

Explications plus détaillées sur les commandes

1) ajouter(devise, *devises_conversion)
Ajoute une devise ainsi que ses différentes devises de conversion au dépôt. Prend en argument la devise et ses différentes devises de conversion. Permet aussi d'ajouter une devise de conversion à une devise existante avec la switch --append.

```
cat .taux.txt
USD;CAD:1.32664;EUR:0.87928;GBP:0.77922

/.gt.rb ajouter "CAD" "USD:1.2345" "EUR:2.34"

cat .taux.txt
```

```
USD;CAN:1.32664;EUR:0.87928;GBP:0.77922
CAD;USD:1.2345";EUR:2.34
```

2) taux_devise(devise_base, devise_conversion)
Retourne la valeur d'une devise par rapport à une autre.

```
cat .taux.txt
CAD;USD:0.75364;EUR:0.66268;GBP:0.58735
USD;CAD:1.32664;EUR:0.87928;GBP:0.77922

./gt .taux_devise "CAD" "EUR"
0.66268
```

3) lister()
Liste toutes les devises du dépôt (ne prend aucun argument). On peut aussi utiliser la switch --raw pour obtenir un "dump" du dépôt.

```
cat .taux.txt
USD;CAD:1.32664;EUR:0.87928;GBP:0.77922
CAD;USD:1.2345;EUR:2.34

./gt.rb lister
USD
CAD
```

4) init
Crée une nouvelle base de données (vide). Utiliser la switch -d pour écraser une base de données existante.

5) supprimer_devise(devise)
Supprime une devise (et ses devises de conversion) du depot.

```
cat .taux.txt
USD;CAD:1.32664;EUR:0.87928;GBP:0.77922
CAD;USD:1.2345;EUR:2.34

./gt supprimer "CAD"

cat .taux.txt
USD;CAD:1.32664;EUR:0.87928;GBP:0.77922
```

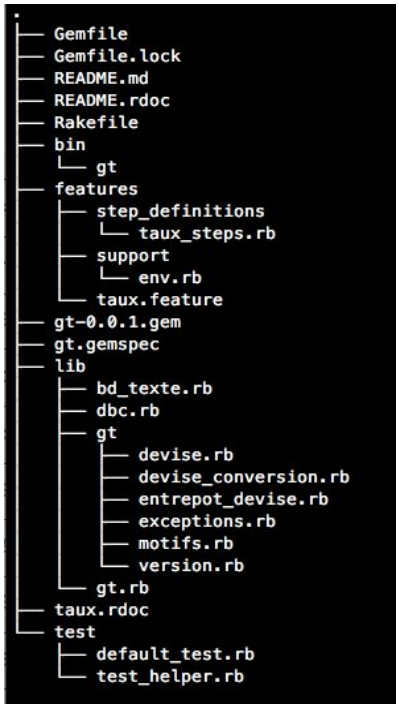
Environnement de développement

Pour ce travail, je (Mathieu) n'ai pas utilisé d'IDE, la totalité de mon code à été écrit dans vim et via le terminal. Pour la consultation et assurer parfois plus d'efficacité, j'ai utilisé, par moments, Sublime Text

puisque celui-ci est visuellement très clair et permet les sélections de patrons ainsi que les substitutions plus facilement. Quant à lui, mon collègue a développé sous Windows 10 dans le logiciel Visual studio Code. Malgré l'utilisation de deux OS et IDE différents, nous n'avons pas rencontré de problèmes avec le gem, ni au niveau du build, ni pour l'implémentation des commandes. Pour le partage nous avons bien entendu utilisé un dépôt sur GitHub.

Architecture

Couche IPM	**Couche métier**	**Couche fichiers**
Fichier bin/gt basé (héritant) du module GLI::APP. Ce fichier fait appel à toutes les classes de la couche métier, mais principalement à la classe EntrepotDevise.	Classe **Devise**. Utilise la classe **DeviseConversion** pour la représentation des ses différentes devises de conversion. Classe **DeviseConversion**. Classe **EntrepotDevises**. Une collection d'objets Devise. Fait appel à **BDTexte** pour obtenir les données stockées sur disque.	**BDTexte**. Fait usage des classes de la couche métier pour la conversion/chargement des données sur disque en données mémoire.



Dans le répertoire de travail, nous avons un fichier "Readme.md", un Rakefile ainsi que le gemfile classique, soit les trois fichier de base d'une application développé avec un gem. Ensuite, le répertoire

"**bin**" contient le fichier main du programme qui est "gt", celui-ci contient l'appel de base des commandes du programme ainsi que la fonction begin de départ. Le répertoire "**lib**" contient les fondations du programme comme le fichier des require "gt.rb", la classe BDtexte et le module "dbc.rb".
Toujours dans "lib", le dossier "**gt"** contient la version du programme et d'autres classes importantes et module de Gestion_Taux comme DeviseConversion et Devise. Enfin, dans le répertoire du programme on trouvera le dernier répertoire des tests.

Tests

Pour nos tests, nous avons fait un fichier de test d'acceptation par commande soit: "init", "lister", "supprimer", "ajouter" et "taux_devise". Le fichier "test_helper" assure que chaque fichier ait la même base et qu'il soit plus facile à implémenter à l'aide d'un "frame" écrit par Guy Tremblay. Les fichiers sont assez rudimentaires, chacun exerce la commande et compare avec un résultat attendu pour voir s'il y a une différence.

Expérience de programmation avec Ruby

Mathieu Lavoie

Implémenter fut une expérience très enrichissante puisqu'elle représente une des plus importante partie du langage Ruby. Bien que parfois difficile à comprendre au départ, la structure du gem aide l'utilisateur à se situer dans le programme et à assurer une longévité ainsi qu'une cohérence dans l'ajout de code tout en plaçant celui-ci dans un standard qui a fait ses preuves. Le plus difficile je crois, fut de bien comprendre l'entièreté de la structure d'un gem avec les répertoire "bin", "lib" et "test" qui doivent chacun contenir les bon fichiers pour assurer la cohérence dans l'implémentation. Chaque dossier à ses standards et sa spécificité qu'il ne faut pas briser. Travailler avec le format csv fut aussi intéressant encore une fois pour le côté standard de la chose. Faire ce tp nous a appris ses standards en plus de renforcer nos bases en Ruby. Ce langage est encore utilisé sur le marché du travail (Ruby on Rails) et l'idée d'apprendre un langage qui mélange à la fois objet et fonctionnel pourra être très utile pour le marché de l'emploi.

Olivier Rochefort

Dans l'ensemble, j'ai aimé développé avec Ruby. Ce langage a vraiment une "personnalité" unique avec ses blocs, ses méthodes pour modifier "on the fly" le code d'une classe, etc… Même si d'autres langages offrent aussi ces fonctionnalités, la différence avec Ruby, c'est qu'on sent qu'il est conçu pour que ces éléments fassent partie de notre code de tous les jours.

J'ai même utilisé Ruby dans le cours de sécurité INF4471 pour développer un petit programme qui simulait le temps pour casser un secret. Les librairies cryptographiques de Ruby sont très biens et faciles à utiliser.

Pour moi la chose que j'aime le moins avec Ruby ça reste les blocs. Je trouve que ça éparpille le code. Je trouve qu'on se retrouve à coder un bout d'une fonction X à une autre place dans le code. Ça rend la

lecture et la compréhension d'une fonction plus difficile. Sûrement qu'avec de l'expérience et les bons outils ce désagrément est atténué.

Ça m'a pris beaucoup de temps pour démarrer avec le TP2, mais une fois que j'ai commencé à comprendre, tout a été très vite. Pas de difficulté majeure avec ce TP hormis le fait de me conformer au style fonctionnel.

Avec le TP3 ça m'a pris un certain temps pour bien assimiler le fonctionnement de MiniTest et de l'organisation du code. On s'est basé beaucoup sur le travail du prof pour ce TP et j'ai été impressionné par la bonne organisation (architecture) de son code. Tout était très bien séparé.