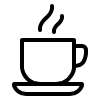
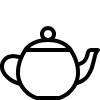
Projet Semestre 5

Rapport niveau 2 : Padawan

Café ou Thé

Eric Ung

Thomas Greaux

Florian Feraud

Fabien Durando

16 Novembre 2016

Table des matières

I. Contexte 2

II. Architecture des classes 4

III. Démonstration des fonctionnalités 7

IV. Diagramme Uml

Annexes

I. Contexte

Ce rapport a pour but de décrire les solutions apportés par notre équipe afin de doter à l'interpréteur Brainf\*ck les nouvelles fonctionnalités telles que :

• l'interprétation de la syntaxe abrégée et des images Brainf\*ck

• la réécriture en abrégée à partir du programme donné en entré

*(à clarifier si on accepte les images)*

• la représentation image des instructions

• la gestion des entrées/sorties

• la gestion des boucles

Nous sommes partis sur notre base de travail présentée lors de l'évaluation 1, puis apporté des changements pour assurer une bonne évolutivité à notre interpréteur.

/

Les détails de ces changements seront décrits dans la partie suivante.

Les démonstrations dans la partie finale vont permettre de valider les fonctionnalités attendues par le client.

II. Architecture des classes

Gestion des nouvelles syntaxes

Afin d'exploiter au mieux les classes de commandes, la syntaxe abrégée a été ajoutée dans les propriété des classes correspondantes aux instructions, i.e pour la commande INCR nous avons défini la variable nameShort qui a pour valeur "+" (cf. document 1 en annexe).

Cela permet à ce que l'instruction et sa syntaxe abrégée soit traitée de la même manière grâce à la classe CommandFactory

De la même manière, nous avons géré la lecture des images en ajoutant à chaque classe de commande le code couleur qui lui correspond par la variable colorRGB/Hexa

La classe ReadImage va donc parcourir l'image en analysant la couleur de chaque pixel supérieur gauche étant donné que chaque instructions possède une zone de 3x3 pixels

Pixel parcouru

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

Une case couleur (3x3) représentant une instruction

Et pour chaque case d'instructions, son exécution se fait également à l'aide de la classe CommandFactory.

/

Gestion des différentes réécritures

La fonctionnalité --rewrite est implémentée dans la classe Reader qui va donner le la syntaxe abrégée d'une instruction (qu'elle soit déjà abrégée ou non) en faisant appel à la méthode getNameShort. Celle ci "remonte" dans la classe CommandFactory qui va sélectionner la syntaxe abrégée adéquate rangée dans nameShort

A reformuler

La fonctionnalité --translate est gérée à l'aide des classes CreateImage et ReadImage

Gestion des entrées/sorties

Gestion des boucles

III. Démonstration

Annexe

Document 1

|  |
| --- |
|  |
| |  |  | | --- | --- | | 1 | package bfck; | | 2 |  | | 3 | import *java.awt.Color*; | | 4 | import *java.util.ArrayList*; | | 5 |  | | 6 | public class Increment implements *Command* { | | 7 |  | | 8 | protected *Color* colorRGB = new *Color*(255, 255, 255); | | 9 | protected *String* colorHexa = new *String*("ffffff"); | | 10 | protected *String* name = new *String*("INCR"); | | 11 | protected *String* nameShort = new *String*("+"); | | 12 |  | | 13 | public *void* execute(*Memory* *m*) throws *Exception* { | | 14 | m.incr(); | | 15 | } | | 16 |  | | 17 | public *ArrayList<String>* getProperties(){ | | 18 | *ArrayList<String>* l = new *ArrayList<String>*(); | | 19 | l.add(colorHexa); | | 20 | l.add(name); | | 21 | l.add(nameShort); | | 22 | return l; | | 23 | } | | 24 |  | | 25 | public *String* getNameShort(){ | | 26 | return this.nameShort; | | 27 | } | | 28 |  | | 29 | public *Color* getRGBColor(){ | | 30 | return this.colorRGB; | | 31 | } | | 32 | } | |

Document 2

|  |
| --- |
|  |
| |  |  | | --- | --- | | 1 | package CoT.bfck; | | 2 |  | | 3 | import *CoT.bfck.Command.*\*; | | 4 |  | | 5 | import *java.io.IOException*; | | 6 | import *java.util.ArrayList*; | | 7 |  | | 8 | public class CommandFactory { | | 9 | *ArrayList<String>* properties = new *ArrayList<String>*(); | | 10 | *Increment* i = new *Increment*(); | | 11 | *Decrement* d = new *Decrement*(); | | 12 | *Left* l = new *Left*(); | | 13 | *Right* r = new *Right*(); | | 14 | *Jump* j = new *Jump*(); | | 15 | *Back* b = new *Back*(); | | 16 |  | | 17 | /\*\* | | 18 | \* Used to make connection between the string given in parameter | | 19 | \* and the corresponding instruction. | | 20 | \* | | 21 | \* @param shapeType | | 22 | \* @return | | 23 | \* @throws Exception | | 24 | \*/ | | 25 | public *Command* getCommand(*String* *command*) throws *Exception* { | | 26 | if (command == null) { | | 27 | return null; | | 28 | } | | 29 | properties = i.getProperties(); | | 30 | for(*String* p : properties) | | 31 | if(p.equals(command)){return i;} | | 32 |  | | 33 | properties = d.getProperties(); | | 34 | for(*String* p : properties) | | 35 | if(p.equals(command)){return d;} | | 36 |  | | 37 | properties = l.getProperties(); | | 38 | for(*String* p : properties) | | 39 | if(p.equals(command)){return l;} | | 40 |  | | 41 | properties = r.getProperties(); | | 42 | for(*String* p : properties) | | 43 | if(p.equals(command)){return r;} | | 44 |  | | 45 | properties = b.getProperties(); | | 46 | for(*String* p : properties) | | 47 | if(p.equals(command)){return b;} | | 48 |  | | 49 | properties = j.getProperties(); | | 50 | for(*String* p : properties) | | 51 | if(p.equals(command)){return j;} | | 52 |  | | 53 | throw new *IOException*(); | | 54 | } | | 55 | } | |