UFR ST - Besançon- L2 Info - Année 2015/16
Programmation par Objets
TP 4 - La Machine de Turing

Ce TP est à réaliser sur **deux séances**. L'objectif est d'implanter un objet simulant une machine de Turing, puis d'utiliser cet objet en lui fournissant un programme pour qu'il réalise un calcul.

Préambule.

Alan Turing (1912-1954) était un mathématicien et logicien anglais. On le considère aujourd'hui comme le père de l'informatique en tant que science. Son premier coup de génie a été la publication en 1936 d'un article dans lequel, pour résoudre un problème mathématique (celui de la calculabilité) posé par Hilbert, il imagine une « machine universelle » telle que tout problème dont la solution est calculable en temps fini l'est par cette machine. Sinon, le problème ne possède pas de solution calculable. On dirait aujourd'hui qu'il est *indécidable*.

Sa machine, purement théorique en 1936, constitue néanmoins le modèle de tout ordinateur du réel. Aucun ordinateur ne peut calculer plus que ce pourrait calculer la machine de Turing.

1re Partie - Codage de la machine en Java

Exercice 1. Coder une classe `TuringMachine`

La machine imaginée par Turing fonctionne avec un ruban de papier (*tape*) découpé en cases, qui lui sert de mémoire. Dans les cases, grâce à une tête de lecture/écriture, on peut lire, écrire ou effacer des symboles. La tête est capable de se déplacer case par case sur le ruban, soit à droite, soit à gauche. Un exemple de machine est donné dans la figure 1.

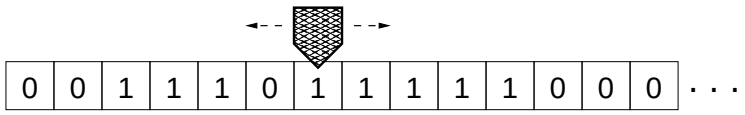

Fig. 1 – Un exemple de machine de Turing

La machine est également dans un certain état (*state*), ce qui va permettre de la programmer comme expliqué plus loin (voir « programmation de la machine »). On retient uniquement pour l'instant que cet état est un numéro (*état 1*, *état 2*, *état 3*, etc.) qu'un programme va faire varier en s'exécutant.

Codage par une classe

Attributs

Plus la machine connaît de symboles, plus elle est facile à programmer. Toutefois pour ce TP, on considère la version la plus fruste de la machine, celle qui ne connaît que 2 symboles : le 0 et le 1. Le 0 sera codé par la valeur `false` et le 1 par la valeur `true`, ce qui permettra de coder le ruban par un simple tableau nommé `tape` de booléens[1].

Remarque. Dans une telle machine, il faut coder les nombres en *unaire*. Le nombre de symboles '1' consécutifs indique tout simplement le nombre codé. Ainsi, le nombre 3 est codé par | 1 | 1 | 1 |, le nombre 5 par | 1 | 1 | 1 | 1 | 1 |, etc.

[1] Dans l'article de Turing, le ruban est théoriquement infini. En pratique, on sera bien entendu obligé de limiter la taille du tableau.

La position de la tête de lecture sur le ruban est elle aussi très simple à coder par un entier nommé index : c'est l'indice de la case dans le tableau où se trouve la tête. On l'appelle la *case courante*. Par exemple, index = 0 signifie que la tête est au début du ruban, index = 3 signifie que la tête est sur la 4e case du ruban, etc.

L'état de la machine est codé par un entier nommé state.

Constructeurs

On instancie la machine en indiquant la taille de son ruban. Si on n'indique pas de taille, une machine à 1000 cases est instanciée.

Autres méthodes

Les opérations de la machine seront assurées par les méthodes suivantes (à vous de réfléchir aux paramètres et aux types de retour).

- write() remplit la case courante (par la valeur true) ;
- erase() efface la case courante (elle y inscrit la valeur false) ;
- read() lit et retourne la valeur inscrite dans la case courante ;
- getState() retourne l'état state de la machine ;
- goRight() déplace la tête d'une case à droite ;
- goLeft() déplace la tête d'une case à gauche.

Si vous le voulez, vous pouvez rendre la machine plus facile à manipuler en rajoutant des méthodes permettant de retourner directement au début du ruban, ou de se déplacer de plusieurs cases en une seule fois.

toString()

Le toString() de la machine représente dans la chaîne retournée la succession des cases du ruban. Un *underscore* (symbole '_') représente une case vide (false) et un '1' représente une case pleine (true). La case courante est entourée de crochets.

Par exemple, la machine de la figure 1 est représentée par la chaîne "__111_[1]1111___".

Raccourci syntaxique. On peut utiliser *l'opérateur conditionnel* autorisé en Java pour décrire en une seule ligne des expressions conditionnelles. Cet opérateur prend la forme

```
condition ? resultat_si_vrai : resultat_si_faux
```

Par exemple, pour mettre dans une chaîne ch le mot "pair" si un nombre n est pair, et le mot "impair" si n est impair, on peut écrire :

```
String ch = (n%2==0) ? "pair" : "impair" ;
```

Exercice 2. Instancier la machine

Créez un programme principal qui crée une machine de Turing à 14 cases, la place dans la même configuration que la machine de la figure 1, puis l'affiche via son toString().

2e Partie - Rédaction de programmes pour la machine

Exercice 3. Programmer la machine

La machine se programme au moyen d'une table d'instructions. Une ligne de cette table est une *instruction* qui indique, en fonction de l'état de la machine (1, 2, 3, ...) et du symbole lu ('0' ou '1'), quelles suite d'actions de la machine il faut réaliser (*effacer*, *aller à droite*, ...), et dans quel état (0, 1, 2 , 3, ...) la machine se retrouvera ensuite.

N.B. Par convention, la machine est dans l'état 1 au départ d'un programme. Par convention aussi, l'état 0 indique que le programme est terminé (la machine s'arrête).

Exemple. Addition de deux nombres. Les deux nombres à additionner sont codés en unaire, et séparés par une seule case vide (*false*). Le résultat est la somme, enregistrée sur le ruban.

Par exemple, si la machine contient 2 et 3 :

0	0	0	1	1	0	1	1	1	0	0	0	0	0

elle devient :

0	0	0	1	1	1	1	1	0	0	0	0	0	0

après l'opération.

Le principe du programme est d'avancer jusqu'au 1er nombre, de le parcourir, d'écrire '1' dans la case intermédiaire, de parcourir le 2e nombre, et pour finir d'effacer le dernier '1' qui est maintenant en trop.

La table réalisant ce programme est :

état courant	symbole lu	suite d'actions	état futur
1	'0'	aller à droite	1
1	'1'	aller à droite	2
2	'1'	aller à droite	2
2	'0'	écrire ; aller à droite	3
3	'1'	aller à droite	3
3	'0'	aller à gauche ; effacer	0

Question 1. Codez une instruction par une classe Java nommée Instruction.

Indications. Les attributs sont les 4 informations d'une ligne de la table : l'état courant (un entier), l'état futur (un entier), le symbole lu (un booléen), et la suite des actions à réaliser. Pour ces actions, on peut choisir de représenter chaque action possible par un caractère : 'r' pour « aller à droite », 'l' pour « aller à gauche », 'e' pour « effacer », et 'w' pour « écrire ». Une séquence d'actions s'enregistre alors comme une chaîne de caractères. Par exemple, la suite d'instructions « écrire ; aller à droite » (4e ligne de la table) s'écrira : "wr". L'action « aller à gauche » s'écrira "l", etc.

Les méthodes sont les constructeurs, et les accesseurs (*getters*) car les attributs seront privés. De plus, la méthode toString() donne, par exemple pour la 4e ligne, la chaîne "<2, '0', wr, 3>".

Question 2. Codez un programme (= une table d'instructions) par une classe Java Program.

Indications. Le programme est une liste d'instances de Instruction. Un attribut code de type tableau d'instructions (Instruction[] code) permet donc de mémoriser les lignes de la table. On enregistre également comme attributs le nombre de lignes de la table, ainsi qu'un nom donné au programme (exemple : "Addition").

Les lignes sont entrées une à une dans la table, au moyen d'une méthode add() prenant en paramètre une instance de Instruction, ou bien directement les informations *état courant, état futur, symbole lu* et *suite des actions*.

Un *getter* getInstruction() approprié doit retourner, pour un état et un symbole lu donnés en paramètres, l'instance de Instruction correspondante.

Enfin, la méthode toString() construit une représentation lisible du programme, comme dans l'exemple suivant :

```
Program ADDITION
0: <1, '0', r, 1>
1: <1, '1', r, 2>
2: <2, '1', r, 2>
3: <2, '0', wr, 3>
4: <3, '1', r, 3>
5: <3, '0', le, 0>
```

Exercice 4. Instancier un programme

Créez dans votre main() une instance de la classe Program correspondant au programme d'addition donné en exemple, puis affichez le *via* son toString().

3e Partie - Exécution des programmes sur la machine

On dispose maintenant d'une part de la machine, et d'autre part de programmes pour celle-ci. Il ne reste plus qu'à les rassembler en faisant exécuter les programmes par la machine.

Exercice 5. Enrichir la machine de méthodes pour l'exécution d'un programme

Question 1. Codez dans la classe TuringMachine une méthode execute() qui exécute une action élémentaire reçue en paramètre, telle que *aller à droite*, ou *écrire*, ... L'action à exécuter est fournie par un caractère ('r', 'l', 'w' ou 'e') en paramètre.

Question 2. Il ne reste plus qu'à faire exécuter un programme complet. Dotez pour cela votre classe TuringMachine d'une méthode process() acceptant une instance de Program en paramètre. Cette méthode exécute bien sûr la ligne du programme correspondant à l'état actuel de la machine et au symbole lu (cette instruction est récupérée par getInstruction()), en soumettant les actions à exécuter à la méthode execute(), et en mettant à jour l'état (state) de la machine après chaque instruction.

Exercice 6. Faire exécuter un programme par la machine

Dans le programme principal :
- placez la machine dans la configuration de la figure 1 comme à l'exercice 2 ;
- ramenez la tête de lecture au début du ruban ;
- créez une instance du programme « ADDITION » comme à l'exercice 4 ;
- faites exécuter ce programme par la machine.

Vous pouvez vérifier le bon fonctionnement de votre programme en affichant la machine *via* son toString() avant et après l'exécution.

Exercice 7. Pour aller plus loin

Vous pouvez rédiger vous-même d'autres programmes et les essayer. Par exemple, un programme qui efface le prochain « mot » (c'est à dire la prochaine séquence de '1'). Ou encore un programme qui arrondit un nombre, au premier nombre pair supérieur ou égal à lui...

Postambule.

Après cette première machine, Turing poussa le concept un cran plus loin avec sa *machine universelle*. Il venait de découvrir qu'il n'était pas nécessaire que le programme (la table des instructions) soit *extérieur* à la machine. On pouvait lui aussi le coder par des nombres directement sur le ruban de papier. On obtenait alors une machine totalement reprogrammable, dans laquelle les instructions avaient le même statut que les données à traiter, devenant ainsi potentiellement des données elles même. Cela ouvrait la possibilité à un programme d'écrire un autre programme, voire de se modifier lui-même. C'est exactement le modèle de la machine sur laquelle vous êtes en train de réaliser ce TP.