UFR ST - Besançon- L2 Info - POA

TD 1 - La Machine de Turing

Ce premier TD, non trivial, est une remise à niveau sur les concepts de classe, de constructeur et d'encapsulation. L'objectif est d'implanter par des objets la simulation d'une machine de Turing, ainsi que des programmes qu'elle exécuterait pour réaliser des calculs.

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 réel actuel. Aucun ordinateur à ce jour ne peut calculer plus que ce pourrait calculer la machine de Turing. Ce TD vous propose de découvrir ce concept de machine de Turing, fondamental en informatique, et d'implanter en Java son fonctionnement.

1re Partie - Codage de la machine en Java

N.B. Vous allez *programmer* une machine *programmable*. Le mot « programmer » est donc ambigu dans ce contexte car il désigne à la fois votre activité de codage en java du fonctionnement de la machine de Turing, ainsi que le fait de donner des instructions à exécuter à cette machine. Dans ce sujet on réserve le mot « programmer » à cette dernière activité : programmation d'instructions destinées à être exécutées par la machine de Turing. Pour votre développement en java, on emploiera le terme « implanter ».

Exercice 1. Implanter une classe `TuringMachine`

Une *machine de Turing* est un modèle mathématique du calcul. Elle manipule des symboles disposés sur un ruban en obéissant à une table d'instructions.

Telle qu'imaginée par Turing, la machine 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 ruban de cette machine est donné dans la figure 1.

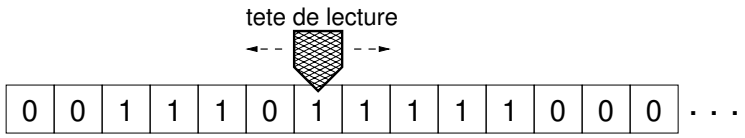

Figure 1 – Un exemple de ruban pour la machine de Turing

L'intuition géniale de Turing est que tout calcul peut se réaliser comme une suite de transformations des symboles du ruban, au moyen de quatre opérations élémentaires : déplacement à droite, déplacement à gauche, effacement d'un symbole, écriture d'un symbole. On ajoute une action de lecture du symbole inscrit sur le ruban qui permettra de décider des opérations suivantes à exécuter.

Écrire un programme pour cette machine consiste à décrire une séquence d'opérations à appliquer sur le ruban. Cette séquence est enregistrée dans une *table d'instructions* qui fait passer la machine d'un *état* à un autre. Vous pouvez voir un exemple de table d'instructions en page 3 de ce sujet (voir la table 1). La machine, en fonction à la fois de son état actuel et du symbole qu'elle lit sur le ruban, va appliquer la suite d'opérations indiquées par la table et passer dans l'état futur indiqué, et ainsi de suite en fonction de ce nouvel état et du nouveau symbole lu.

Implantation par classes

On procédera en deux temps en implantant d'abord le ruban encapsulé dans sa propre classe (classe Tape), puis la machine de Turing elle-même (classe TuringMachine) qui utilise ce ruban comme un objet.

La classe Tape

Attributs

Plus la machine connaît de symboles, plus elle est facile à *programmer*, mais plus elle serait difficile à *implanter* (pour vous donc). Ainsi pour ce TD/TP, on considère la version la plus rudimentaire de la machine, celle qui ne connaît *qu'un seul symbole* : le 1. On utilise aussi le 0, mais pour représenter une case vide. 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.

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.

N.B. On demande d'appliquer strictement le concept d'encapsulation : les attributs sont privés, de telle sorte que le ruban ne pourra être manipulé que par l'utilisation des méthodes, qui elles sont publiques.

Constructeurs

On instancie le ruban en indiquant sa taille. Si on n'indique pas de taille, un ruban à 1000 cases est instancié.

Autres méthodes

La machine se manipule au moyen des cinq opérations élémentaires décrites ci-dessus (à vous de réfléchir aux paramètres et aux types de retour) :

- goRight() déplace la tête d'une case à droite ;
- goLeft() déplace la tête d'une case à gauche ;
- 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 (vrai ou faux) inscrite dans la case courante.

toString()

La méthode toString() du ruban représente dans la chaîne de caractères retournée la succession des cases ce celui-ci. 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, le ruban de la figure 1 est représenté 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" ;
```

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 classe TuringMachine

La machine de Turing dispose d'un ruban sous la forme d'un attribut tape de type Tape. Afin de pouvoir exécuter des programmes, elle dispose également d'un attribut entier state qui code l'*état* dans lequel se trouve la machine par rapport à la table d'instructions. Cet attribut est initialisé à la valeur 1. Son utilité sera mieux comprise dans la deuxième partie de ce sujet, au moment d'exécuter les instructions de la table.

Les opérations élémentaires de manipulation du ruban sont fournies par encapsulation par l'instance tape. En complément, la machine propose un accesseur (en anglais un *getter*) sur son attribut state, ainsi qu'une méthode toString() qui se contente de retourner la chaîne décrivant son ruban.

Vous pouvez si vous le souhaitez rendre la machine plus facile à manipuler en lui ajoutant des méthodes permettant de retourner directement au début du ruban, de se déplacer de plusieurs cases en une seule fois, ou encore d'enregistrer sur le ruban une séquence de '0' et de '1', donnée en paramètre sous la forme d'une chaîne de caractères par exemple. Puisque le ruban est encapsulé, on est tenus d'implanter ces méthodes sous la forme de boucles d'opérations élémentaires du ruban. Ainsi on ne « triche » pas : la machine ne sait au final faire que des séquences d'opérations élémentaires, qui sont celles fournies par les méthodes de son attribut de type Tape.

Exercice 2. Instancier la machine

Écrivez 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 sa méthode toString().

2e Partie - Rédaction de programmes pour la machine

Exercice 3. Programmer la machine

La machine exécutant un programme passe par une succession d'états (attribut state). Dans chaque état, elle exécute une série d'opérations élémentaires qui varie en fonction du symbole lu par la tête de lecture, ce qui l'amène dans un nouvel état.

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).

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'opérations élémentaires de la machine il faut réaliser (*effacer*, *aller à droite*, ...), et dans quel état (0, 1, 2 , 3, ...) la machine se retrouvera ensuite.

Exemple. Addition de deux nombres. Les deux nombres à additionner doivent être sur le ruban, codés en unaire et séparés par une seule case vide (*false*). Le résultat sera 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 séparatrice, de parcourir le 2e nombre, et pour finir d'effacer le dernier '1' qui est maintenant en trop.

La table d'instructions réalisant ce programme est donnée par la table 1.

état courant	symbole lu	suite d'opérations	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

Table 1 – Table d'instructions réalisant l'addition de deux nombres codés en unaire

Cette table est plus facile à lire sous la forme d'un *automate* : voir la figure 2. Vous étudierez ce formalisme en L3, mais la représentation graphique d'un automate est facile à comprendre. Les cercles représentent les états et une flèche indique le passage d'un état à un autre (éventuellement le même). Ces changements d'état sont étiquetés par le symbole lu et la suite d'opérations (en abrégé : une lettre par opération) à réaliser. Les abréviations utilisées pour les noms d'opération sont celles présentées dans les indications de la question qui suit.

Par exemple, ① $\xrightarrow{'1' : r}$ ② s'interprète (par la machine) comme : « si je suis dans l'état 1 et que je lis le symbole '1', alors je dois réaliser l'opération *aller à droite* (abrégée par r), ce qui m'amène dans l'état 2. »

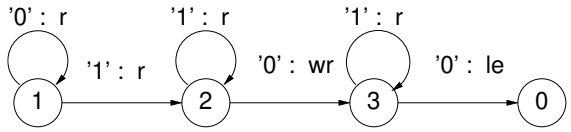

FIGURE 2 – L'automate du programme d'addition

Question 1. Implantez une instruction (c'est à dire une ligne de la table) 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 opérations à réaliser. On peut choisir de représenter chaque opération possible par un caractère : 'r' pour « aller à droite », 'l' pour « aller à gauche », 'e' pour « effacer », et 'w' pour « écrire ». Une séquence d'opérations 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'opération « 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. Implantez un programme (= une table d'instructions) par une classe Java Program.

Indications. Le programme est une table 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 insérées une à une dans la table, au moyen d'une méthode add() prenant en paramètre une instance de Instruction. La méthode add() est surchargée pour aussi accepter directement en paramètres les informations *état courant, état futur, symbole lu* et *suite des opérations*.

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 et retourne 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* sa méthode toString().

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. Implantez dans la classe TuringMachine une méthode execute() qui exécute une opération élémentaire reçue en paramètre, telle que *aller à droite*, ou *écrire*, ... L'opération 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.

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 le ruban avant et après l'exécution de celui-ci.

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 l'ordinateur sur lequel vous implanterez ce sujet en TP.