Projet JAVA

La machine de Turing

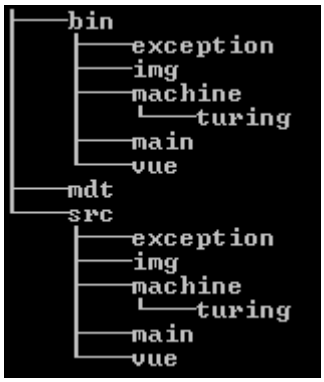
Développement d'une application permettant de construire et simuler une machine de Turing à un ou plusieurs rubans.

Par Omar EDDASSER – L3 ISC parcours MIAGE
Sous l'enseignement de : M. Romain PECHOUX
2011 / 2012

Sommaire

I. Structure du projet

1. Contenu de l'archive



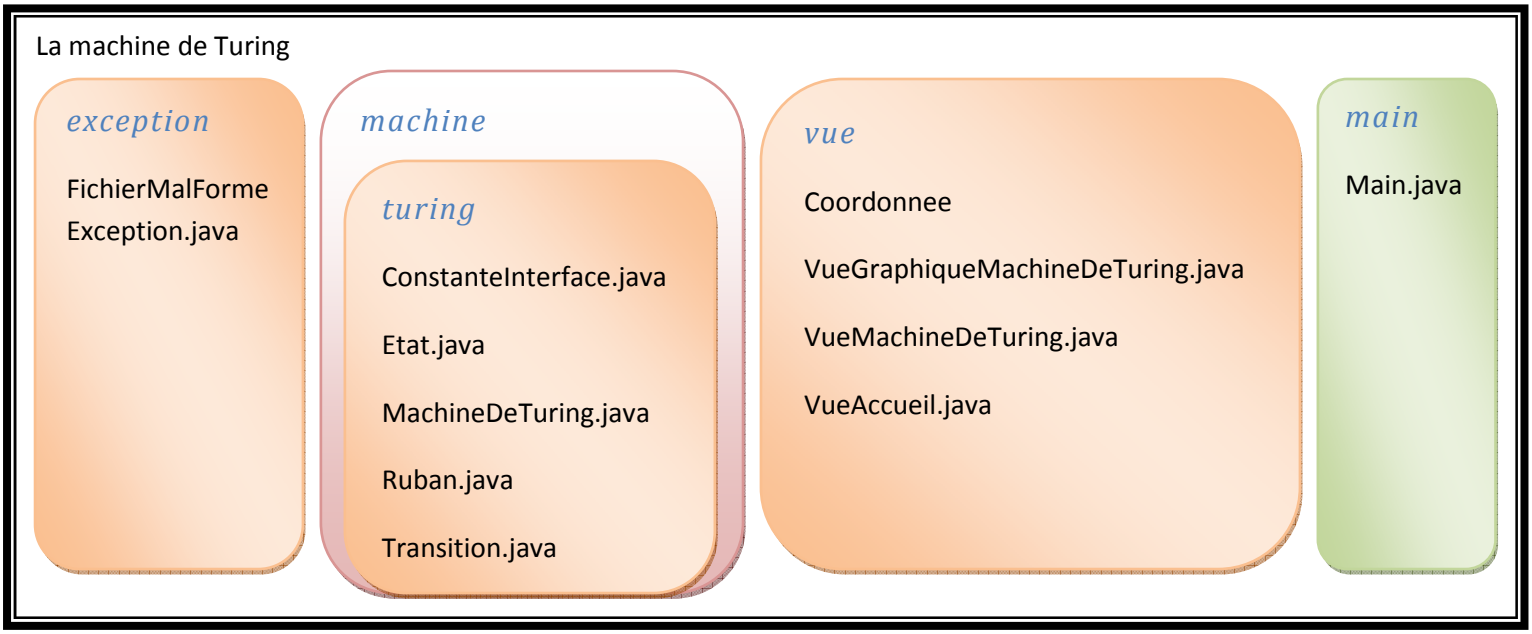
/bin : contient le programme compilé, fichiers « .class »

/src : contient les fichiers sources, fichiers « .java »

/mdt : Répertoire (par défaut) dans lequel se trouvent les machines de Turing que l'application simule.

/Machine_de_Turing_EDDASSER.jar : fichier exécutable permettent de lancer le programme

2. Les packages



Note : Toutes les classes ne sont pas affichées dans le schéma ci-dessus

exception : Package contenant mes propres classes d'exception héritant de la classe Exception.

machine : Package contenant la classe abstraite Machine décrivant les machines de façon générique.

turing : Sous-Package du package « ***machine*** » contenant toutes les classes nécessaire à l'implémentation de la machine de Turing.

vue : Package contenant toutes les classes permettant de visualiser une machine de Turing.

main : Package principale contenant la classe « main » permettant de lancer l'application.

3. Description des choix techniques d'implémentation

La principale liberté que je me suis autorisé à été l'utilisation de fichier pour stocker les machines de Turing. En effet, mon application lit et construit les machines à partir de fichier de la forme :

TTTTT UU V W X YY Z

Où :

TTTT : désigne l'identifiant du ruban (chaine de caractères quelconque)

UU : désigne le nom de l'état de départ (chaine de caractères quelconque)

V : le caractère lu sur le ruban

W : le caractère écrit sur le ruban

X : le mouvement du pointeur sur le ruban (dans l'ensemble :{G, D, I} pour gauche, droite, immobile ; par défaut, c'est immobile)

YY : désigne le nom de l'état d'arrivé (chaine de caractères quelconque)

Z : (facultatif) permettant de spécifier si l'état de départ est acceptant ou non (dans l'ensemble :{1, *} pour 1 acceptant sinon non)

Dans le cas d'un état acceptant n'ayant aucune transition (vers un autre état), il est possible d'abrégé la syntaxe en : TTTT UU 1 (TTTT étant l'identifiant du ruban, UU le nom de l'état)

J'ai choisi d'utiliser des fichiers car je souhaitais une grande flexibilité au niveau de la construction des machines de Turing. Il n'y a aucune limitation en termes de nom d'état, de caractère autorisé par la machine… En effet, puisque chaque machine est définie dans un fichier où sont spécifiés les caractères que peut lire la machine sur le ruban à partir d'un état donné. Il y a cependant quelques restrictions liée à l'application (comme par exemple, le caractère 'e', « l'espace » comme séparateur dans les fichiers, ou encore l'extension des fichiers…) principalement définies sous forme de constante dans l'interface *ConstanteInterface*.

Aussi, pour la structure globale du projet, j'ai choisi une architecture similaire au MVC (bien que n'ayant pas de contrôleur dans mon application) car celle-ci est pratique pour la réutilisation ou modification (plus tard) du code.

Le reste des choix étant imposé par l'énoncé du projet, comme par exemple l'utilisation d'une structure dynamique pour représenté un ruban « infini » (dans mon cas, j'ai utilisé une ArrayList).

4. Exécution du programme

L'application dispose d'une interface graphique, vous devez :

- spécifier le répertoire où se trouvent les machines de Turing (menu *Fichier > Changer répertoire machines*),
- choisir la machine à exécuter (menu *Machine de Turing*),
- remplir le(s) ruban(s) en fonction de la machine sélectionnée.

Pour tester les machines que j'ai implémentées :

- **Division par deux d'un nombre binaire** : Sur le ruban n°1 mettre le nombre à diviser (ex : **0100**), le résultat se trouve sur le ruban en dernière étape (ici : **010**)
- **Somme de deux nombres binaire** : Mettre sur le ruban n°1 les deux nombres à additionner en les séparant d'un 'e' (ex : **0100e0100**)
- **Somme de plusieurs nombres binaire** : Version améliorée de la précédente, on peut mettre plusieurs nombres d'affilé à additionner (ex : **0100e0100e0100**)
- **Somme de deux nombres binaire sur 3 rubans** : Mettre le premier nombre sur le ruban n°1, le second nombre sur le ruban n°2, et laisser vide le ruban n°3 (ex : **0100** et **0100**), le résultat se situe la dernière étape du ruban n°3 (ici : **1000**)

- **Multiplication nombres binaires** : A la manière de la somme sur un ruban, mettre le premier nombre et le seconde séparer par un ‘e’ (ex : **0100e0100**)
- **Chaine de longueur paire** : Mettre sur le ruban une chaine composée de 0 et de 1 (le calcul réussi si la chaine est paire ex: **0100**, il échoue sinon ex: **010**)
- **Calculatrice Binaire** : Mettre sur le ruban une succession d’opération en utilisant les symboles basique (ex : **010*010+010** effectuera les opérations dans l’ordre d’apparition, *attention cette machine n’a pas complètement été testée*)

II. Difficultés rencontrés

1. Au niveau programmation

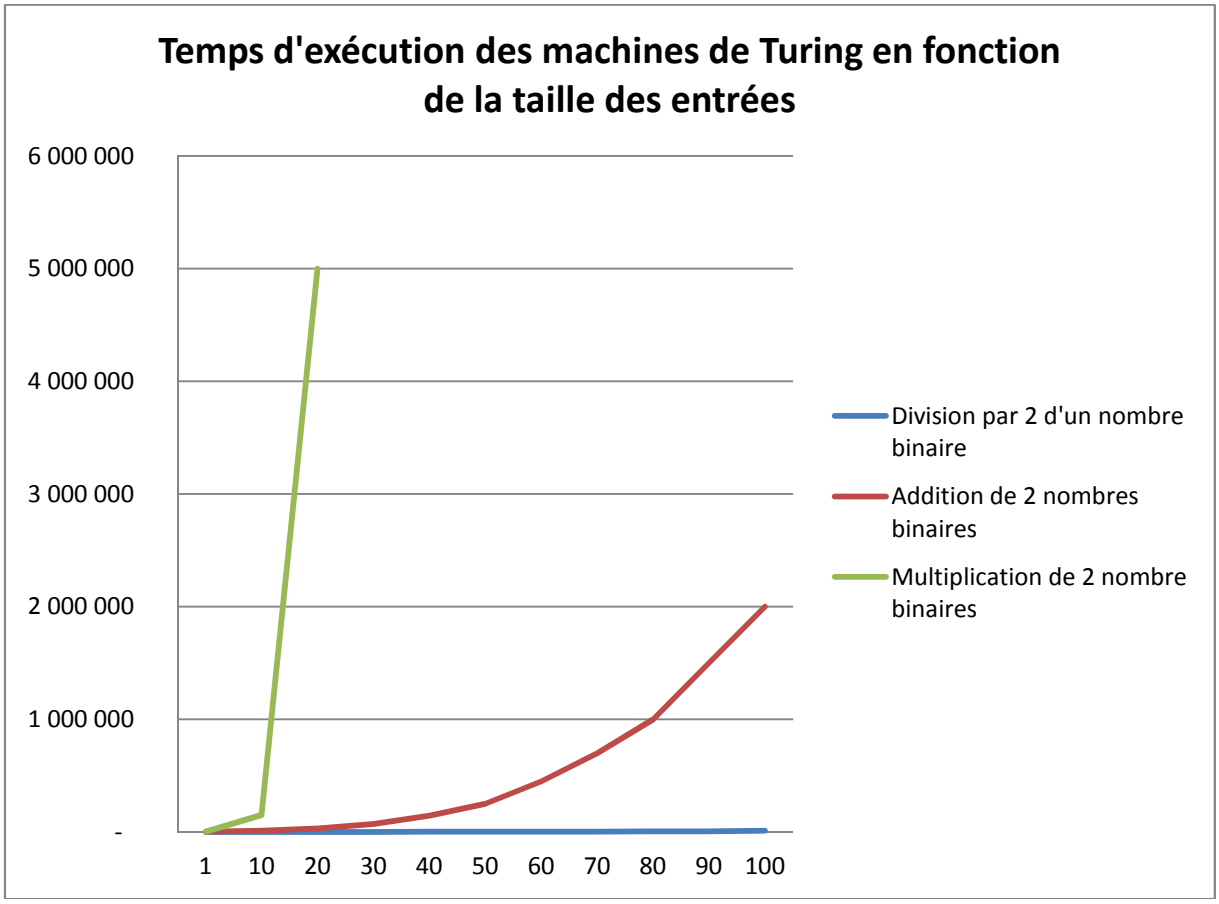
Ayant plus de quatre années de développement en java, je n’ai pas rencontré de réelles difficultés lors du développement de l’application.

2. Au niveau des machines de Turing

La principale difficulté du projet à été de trouver les « algorithmes » des machines de Turing (notamment l’addition et la multiplication de nombre binaire). J’ai surmonté ce problème en demandant quelques conseils à M. Guy PERRIER pour l’addition (la multiplication étant en fait une succession d’addition).

III. Statistiques

Temps arrondi en microseconde		Machine de Turing					
		Division par 2 d'un nombre		Addition de 2 nombres		Multiplication de 2	
		Temps	Nb étapes	Temps	Nb étapes	Temps	Nb étapes
Taille des entrées (n)	10	850	12	10 000	348	150 000	4 716
	20	1 000	22	30 000	1 078	5 000 000	46 561
	30	1 300	32	70 000	2 208		
	40	1 500	42	145 000	3 738		
	50	2 000	52	250 000	5 668		
	60	3 000	62	450 000	7 998		
	70	4 000	72	700 000	10 728		
	80	5 000	82	1 000 000	13 858		
	90	7 000	92	1 500 000	17 388		
	100	10 000	102	2 000 000	21 318		
Ratio du temps (ou nb étapes) / taille des entrées	10	85	1	1 000	35	15 000	472
	20	50	1	1 500	54	250 000	2 328
	30	43	1	2 333	74		
	40	38	1	3 625	93		
	50	40	1	5 000	113		
	60	50	1	7 500	133		
	70	57	1	10 000	153		
	80	63	1	12 500	173		
	90	78	1	16 667	193		
	100	100	1	20 000	213		



Note 1: le temps est donné en microseconde
Note 2 : le calcul du temps pour la machine de Turing (réalisant les multiplications) pour des entrées supérieurs à 20 n'a pas été effectué (car trop long).
Note 3 : Tous les tests on été effectué uniquement avec des 1 comme entrée (ce qui donne donc les calculs les plus longs).
Note 4 : Les temps de calcul (notamment pour l'addition et la multiplication) sont globalement importants car il s'agit de mes propres machines de Turing et que je ne l'ai est pas minimisé.

Conclusions sur les statistiques :

On remarque que pour la première machine de Turing (division par 2) : le temps est constant et est proportionnel à la taille de l'entrée. On a également un nombre d'étapes constant (à raison d'une étape par caractères en entrée).

Pour l'addition binaire, le temps et le nombre d'étapes augmente de manière croissante : à chaque fois que l'on ajoute 10 caractères en entrée, le temps est multiplié par 1,5 et le ratio du nombre d'étape par caractère est augmenté de 20.

Bien qu'ayant peu de données pour la multiplication, on remarque que le temps semble croître de manière exponentielle avec la taille des entrées.

IV. Remarques sur le projet

La classe **VueGraphiqueMachineDeTuring.java** n'est pas utilisé dans cette version du projet, de la même manière quelques attributs de certaines classes sont inutilisés (ex : L'attribut **coordonnee** de la classe **Etat**). J'ai en effet prévu de développer plus tard une version 2 de ce projet qui affichera graphiquement une machine de Turing (et l'automate associé), elle sera aussi plus rapide dans l'exécution que celle-ci car elle procédera notamment à la minimisation des machines de Turing fournit dans les fichiers en entrée avant de les simulés (contrairement à cette version qui ne fait que simuler simplement les machines).