Projet S4 2017/2018 *Université Paul Sabatier*

CONCEPTION JAVA

- La BlockChain en orienté objet -

GODINO Pierre & ROLLAND Florian *09/05/2018*

Personnel investi :

GODINO Pierre

Etudiant en Informatique

Université Paul Sabatier

Responsable Projet & JAVA

LHOSPICE Cloé

Etudiante en sciences de l'éducation

Université Jean Jaurès

Monkey Testeur

Sommaire :

1. Les Structures de données

Les Structures de Données, ou Classes en JAVA, sont ici séparées en plusieurs catégories :

-Les Structures d'objets telles que les Blocks, ou bien la Blockchain, ou Transaction.

-Les Structures Utilitaires telles que Date.

-Les Structures que j'appelle 'Globales' qui ne s'instancient pas étant donné qu'elles ne contiennent que des attributs et méthodes statiques, ici la Classe Paramètres.

Je vais maintenant vous détailler toutes mes structures :

La Classe Block :

Cette Classe est la classe d'un bloc de la blockchain, qui contient les transactions, les hashs etc. … On y retrouve :

```
(Entier) Index : Numéro du bloc dans la continuité de la blockchain
(Chaine de caractères) Timestamp : Date de création du bloc
(Chaine de caractères) Hash : Hash du bloc
(Chaine de caractères) HashMerk : hash de toutes les transactions
(Chaine de caractères) HashPrev : hash du bloc précédent dans le chainage
(Entier) NbTrans : nombre de transactions que contient le bloc
(Entier) Nonce : Chiffre adaptable permettant d'appliquer la difficulté au hash
(Chaines de caractères) Transactions : Tableau contenant toutes les transactions
(Entier) NbTransMax : nombre maximum de transactions que peut avoir le bloc
```

La Classe BlockChain :

Ici, nous sommes au cœur de la machine, on y gère la difficulté, les blocks, etc… On y retrouve :

```
(Entier) Difficulty : Nombre qui représente la difficulté de la BlockChain
(Entier) NbBlocks : Nombre de Blocks qui composeront la BlockChain
(Entier) NbTransMax : Nombre de transactions maximum par bloc
(ArrayList<Block>) Blockchain : Liste de tous les blocs
```

La Classe Date :

Classe utilitaire créée pour Donner l'heure exacte, notamment sous forme de chaine de caractères, et sans avoir besoin d'instancier quoi que ce soit.
On y retrouve :

```
(Calendar) DateCourante : Permet d'accéder aux fonctionnalités de Calendar
(Entier) Day : Contient le numéro dans le mois du jour actuel
(Entier) Month : Contient le numéro du mois actuel
(Entier) Year : Contient l'année actuelle
(Entier) Hour : Contient l'heure actuelle
(Entier) Minute : Contient le chiffre de la minute actuelle
(Entier) Second : Contient le chiffre des secondes actuelles
```

La Classe Paramètres :

La Classe utilisée pour stocker les paramètres de la BlockChain, elle distribue les paramètres à n'importe quelles Classe de manière globale sans avoir besoin d'être instanciée.
On y retrouve :

```
(Entier) Difficulty : Difficulté de la Blockchain
(Entier) NbBlock : Nombre de blocs qu'aura la Blockchain
(Entier) NbTransMax : Nombre maximum qui pourra être généré pour déterminer
le nombre de transactions par bloc
```

2. Les Méthodes

La Classe Block :

Dans la classe Block, on retrouve plusieurs méthodes essentielles, notamment dans le mécanisme de calcul du hash, et du hash de Merkle.

Tout d'abord, nous avons la fonction generateTrans() qui génère une transaction aléatoire parmi deux lots de 8 prénoms donnés, elle fonctionne grâce à la bibliothèque java RANDOM qui permet d'avoir une sélection à peu près aléatoire.

Ensuite nous avons la fonction fillBlock(), qui, comme son nom l'indique remplis le tableau de transactions avec des transactions générées grâce à la méthode generateTrans().

Maintenant que nous avons nos transactions au complet, nous allons pouvoir calculer le hash de ces dernières à l'aide de la méthode merkle() ;
Qui copie la liste de transactions, et manipule la copie de manière à hasher deux par deux les transactions (hash avec la fonction du package google fournis) jusqu'à obtenir qu'un seul hash, donc issu initialement du hash de chaque transaction.
Cette fonction, initialement a été implémentée avec deux listes communiquant ensemble, mais par soucis de performances lorsqu'il y a beaucoup de transactions, elle a été optimisée pour ne fonctionner qu'avec un seul tableau de hashs qui ne prends pas plus de mémoire qu'il n'y a de transactions dans le bloc.

Pour pouvoir calculer le hash du bloc suivant la difficulté, la méthode isHashOK() a été implémenté en tant que fonction auxiliaire à la fonction de calcul du hash.
Cette fonction retourne un booléen, vrai si le hash qu'on lui passe en paramètre correspond bien à la difficulté de la blockchain (n zéros au début du hashs, où n est la difficulté) sinon faux.

Nous avons maintenant tout pour calculer le hash du bloc, on introduit donc la méthode calculateHash() qui hash le timestamp, le HashPrev, le HashMerk, le nonce, le NbTrans et l'index du bloc ensemble, et qui par la suite envoie le hash obtenu en vérification à isHashOK(), si la fonction retourne Vrai, alors le hash est correct et respecte la difficulté, sinon le nonce est incrémenté de 1 et le processus recommence, et ce jusqu'à ce que le hash soit correct.

A part de tout cela nous avons aussi la fonction printBlock() qui retourne une chaine de caractère représentant le bloc avec ses principaux attributs. Cette fonction n'est utile que pour l'interface et l'affichage d'un bloc dans le terminal.

Bien sûr, il y a aussi quelques getter et setters.

La Classe BlockChain :

Dans la classe BlockChain, on retrouve quelques méthodes intéressantes pour la manipulation de la blockchain.

Par exemple, la méthode JSONWritting(String) va créer un fichier du nom de la chaine de caractères passée en paramètres, et ensuite y écrire la BlockChain au format JSON à l'intérieur afin de la conserver pour plus tard, ou pour l'altérer manuellement afin de tester les fonctions de vérification.

On dispose aussi d'un constructeur spécial afin de charger une blockchain enregistrée dans un fichier.

La méthode CreateBlock() est certes basique, mais essentielle car elle permet de créer un block en suivant la continuité de la blockchain et des blocs déjà présents dans cette dernière.

On dispose ensuite de la méthode generateBlockchain() qui créé la blockchain de toute pièces à partir du block numéro 2, en appelant la fonction createBlock() autant de fois que nécessaire pour respecter le paramètre NbBlocks, et s'occupe du chainage en mettant à jour les HashPrev des blocs.

Les fonctions Verif1() et Verif2() sont aussi disponibles, afin de vérifier l'intégrité de la BlockChain, notamment la présence du Genesis (block 1), la cohérence des Hashs et HashMerk, et le chainage.

Pour finir, la méthode parcourir() sert uniquement à afficher la Blockchain, elle retourne sous forme de chaine de caractères la concaténation de n printBlock(), où n est le nombre de blocs, et affiche donc dans son entièreté la BlockChain, et dans l'ordre.

La Classe Date :

Dans cette Classe, on ne dispose que d'une seule méthode, qui est la raison principale de l'existence de cette Classe : la fonction chaine().

Cette fonction retourne sous forme de chaine de caractères la date et l'heure courante, utile lors de la création des différents timestamps, il suffit de l'appeler pour qu'elle donne en temps réel l'heure et la date.

La Classe Paramètres :

Dans cette Classe, les fonctions sont majoritairement des getters et setters qui servirons dans l'interface pour choisir les options de la BlockChain, néanmoins on y retrouve la fonction très utile GenerateNbTransMax(), qui permet de calculer de manière aléatoire le nombre de transactions qu'aura un bloc dans un fourchette entre 1 et un chiffre choisi, NbTransMax.
La fonction retourne l'entier et sera ensuite utilisée par la Classe Block pour sa construction.

3. L’interface graphique

Une fois le programme implémenté de manière presque total, il était temps de s’attaquer au visuel.

Une idée plus ou moins concrète se dessinait de base dans ma tête quant à l’apparence qu’aurait cette interface, sobre, un menu de tab, avec des boutons pour chaque action et un terminal noir pour afficher les résultats et autres indications.
C’est lors de la concertation de mon collègue que nous avons finalement décidés l’IHM de notre interface.

L’interface commença donc à être implémentée via le logiciel NetBeans IDE, grâce au JAVA Swing.

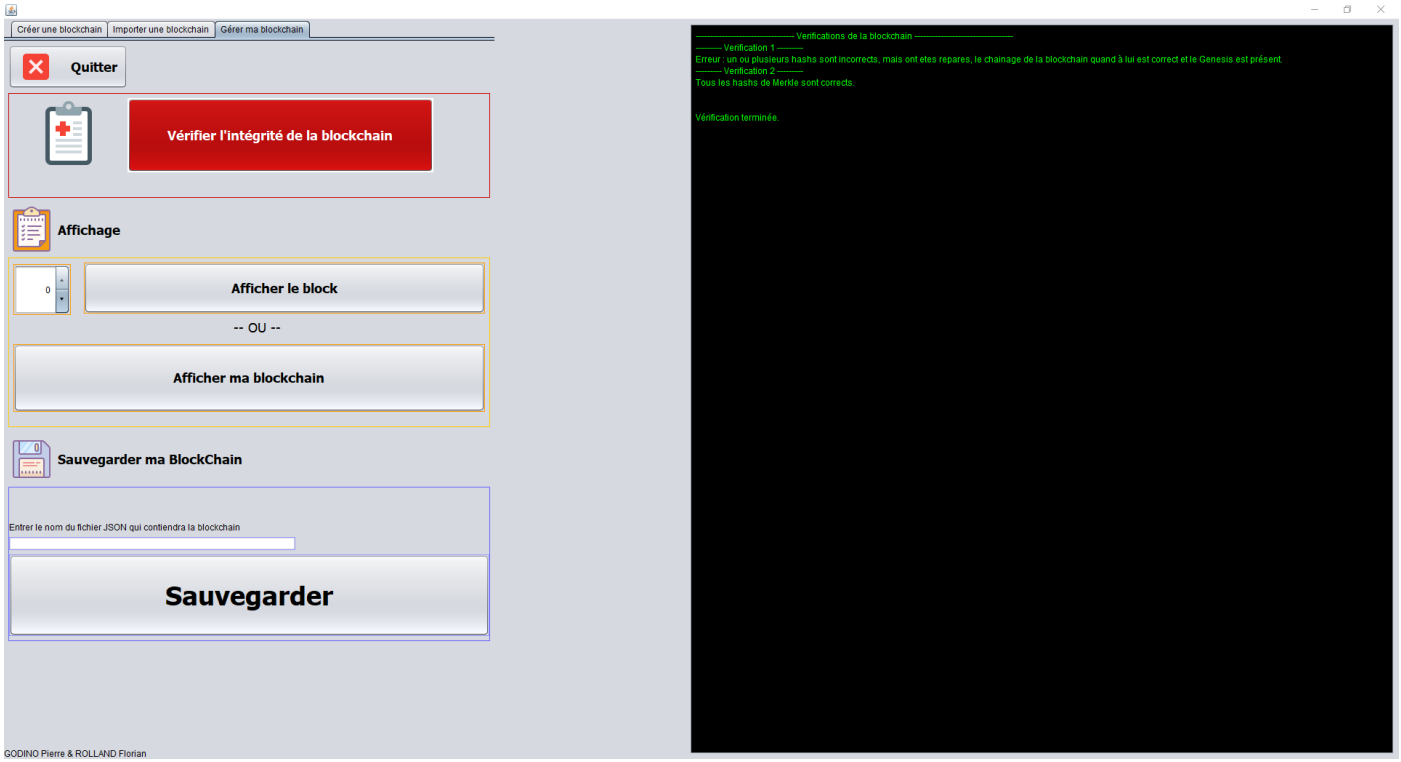
Ce fut bien plus laborieux qu’attendu, et le temps initialement alloué à l’interface dans le planning initial a été doublé voir triplé.
La réalisation de l’interface a nécessité le codage de nombreuses fonctions permettant l’affichage sur un terminal et le traitement en ordre des fonctions des Classes BlockChain et Block, sans parler de la modification d’une grande partie du code initial afin d’adapter leur manière de gérer les données de sortie pour pouvoir y avoir accès depuis le terminal.

Une première version BETA de l’interface fût créée, avec toutes les fonctionnalités attendues telles que :

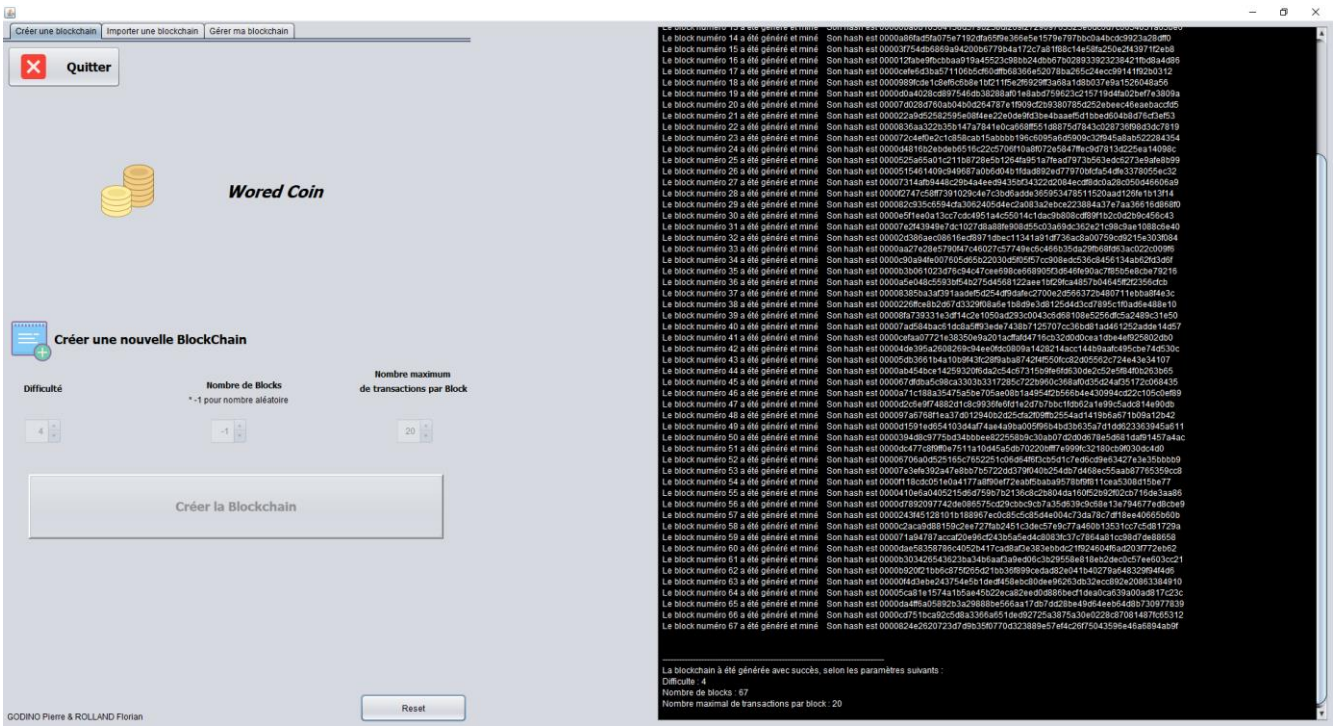
- La vérification de la Blockchain
- L’affichage de la Blockchain
- L’affichage d’un Block
- La création d’une Blockchain selon des paramètres variables
- L’importation d’une Blockchain enregistrée sous format JSON
- La sauvegarde de la Blockchain sous format JSON

Il y a eu plusieurs versions de l'interface, selon les retours, les avis extérieurs et surtout selon le rapport du Monkey Testeur, qui nous a été très utile, notamment pour trouver des défauts mineurs au niveau de l'orthographe et de l'agencement de l'interface.

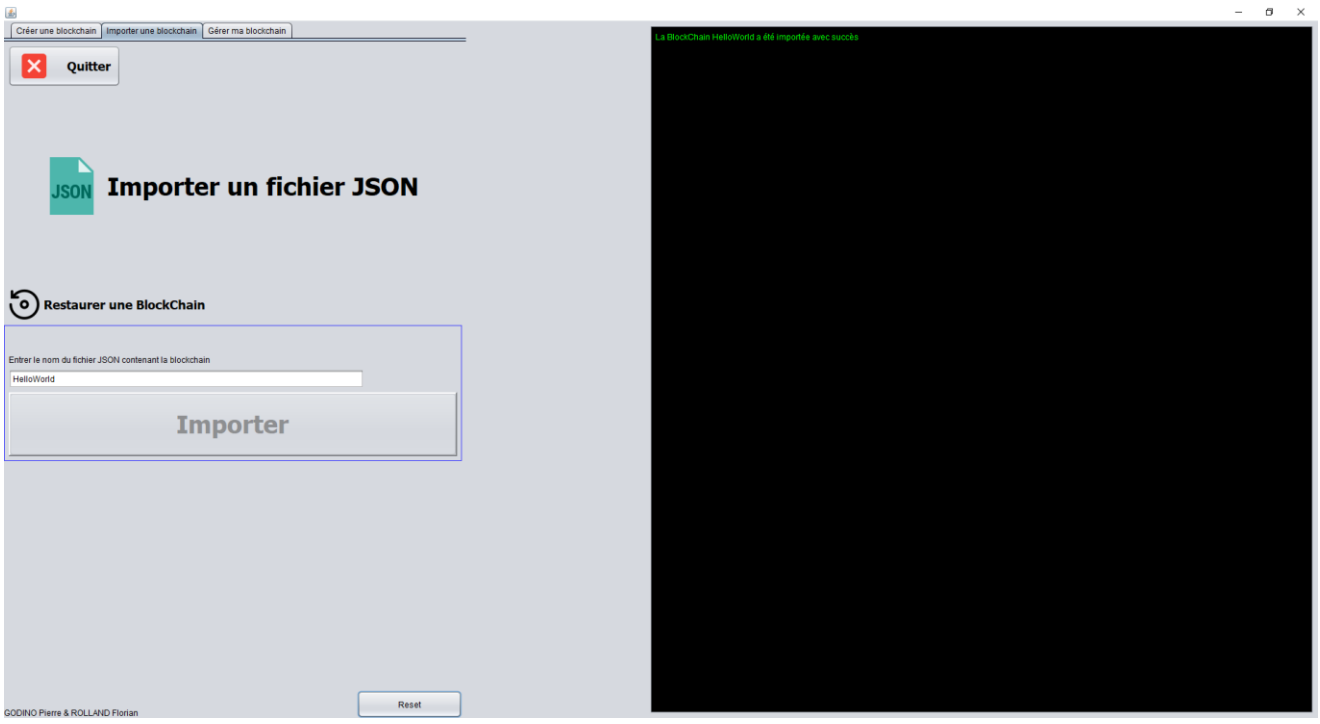
Voici quelques images de la version finale de l'interface :



Page de gestion de la BlockChain



Page de Création de BlockChain



Page d'importation depuis fichier JSON

4. Premiers tests & retours

Travaillant sur du code JAVA, je n'ai bien entendu eu aucuns soucis de compilation et de gestion de mémoire, néanmoins j'ai dû procéder à des tests pour plusieurs raisons :

Certains algorithmes sont relativement complexes et ont nécessités pas mal de recherche, les algorithmes fournis sont assez compliqués à comprendre, des tests sont alors nécessaires, enfin le développement de classes statiques a pris du temps car peu d'expérience à ce niveau-là, donc beaucoup de tests furent nécessaire pour arriver à quelque chose de complètement fonctionnel.

Quand je parle d'algorithmes je pense notamment à l'algorithme de Merkle qui a nécessité pas mal d'essais avant d'être satisfaisant, sachant qu'il a été implémenté à deux reprises, par la suite, les méthodes de calcul de hash et respect de la difficulté ont nécessitées des recherches car elles ont besoins de méthodes JAVA inconnues pour moi, j'ai donc bien entendu fais des recherches et testé diverses fonctions jusqu'à arriver à un résultat concluant.

Les premiers retours étaient plutôt bons, le programme étant plus rapide que ceux de quelques groupes avec qui j'ai pu comparer, nous étions plutôt satisfaits à ce niveau-là.

En ce qui est de l'interface par contre, les retours n'étaient pas très bons, la V1 de l'interface n'était pas cohérente dans son agencement, des fautes d'orthographe étaient présentes, elle était visuellement fade, triste…

C'est là qu'est intervenu le Monkey Testeur.

5. Monkey Test & Version Finale

En tant que testeuse du programme, je ne connais rien en ce qui est de la programmation, j'ai d'ailleurs appris à l'occasion ce qu'était une Blockchain, mon avis est donc totalement neutre et objectif, je suis le consommateur.

Mon premier avis sur le programme, fut qu'il manquait de couleurs, il était assez terne, triste, on n'avait pas vraiment envie de l'utiliser.
Le responsable du programme a donc revu avec moi ce qui n'allait pas dans le visuel, il a ajouté des couleurs de repère sur les commandes et dans le terminal avec le texte de couleur dans certaines situations.

A partir de là j'ai commencé à regarder d'un autre œil le programme, c'était toujours un peu fade, mais on m'a dit que c'était normal car les connaissances en JAVA du responsable étant limitées, il ne pouvait pas faire plus recherché. Ce n'était déjà pas trop mal de toute façon.

J'ai donc découvert quelques fautes d'orthographes, pas bien grave, mais cela ferait tâche si quelqu'un s'en apercevait, elles ont donc toutes été corrigées.
J'ai donc commencé à utiliser le programme après un mini-cours sur les BlockChain et à première vue, d'après ce que je sais, ça à l'air de marcher. J'ai poussé le programme à ses limites et il n'y a pas de bugs concernant la création de blockchains.

J'ai, par contre, découvert des bugs lors de la vérification d'une Blockchain que j'avais altéré manuellement puis importée, le programme ne détectait pas les erreurs dans le hash de Merkle des blocs.

Et j'ai d'ailleurs aussi trouvé un bug avec l'utilisation des fonctionnalités de sauvegarde et d'import de Blockchain, un problème de librairies m'a dit le responsable.

Lorsque tout fut corrigé, j'ai re-testé le programme une dernière fois et tout était OK, le responsable a donc déclaré que le programme avait atteint sa version finale.