II.1102 : Algorithmique et Programmation
TP 6 : Structures de données

Table des matières

1 Objectifs du TP

— Se familiariser avec les structures de contrôle présentées en cours.

2 Rappels

2.1 Contenant et contenu

Les structures de données présentées en cours ont la particularité d'être des types de variable à part entière. Il faut ensuite définir le type des éléments contenus dans ces structures de données. C'est l'une des principales différences avec les tableaux, dont le type est fortement lié à leurs contenus.

En Java, il existe un grand nombre de structures de données mises à notre disposition. Il est important de les connaître et de savoir les manipuler lorsque nous avons à développer une application en Java (ou dans tout autre langage de programmation). Une compétence importante pour un développeur est donc savoir quand se tourner vers la documentation officielle pour y trouver les informations concernant les méthodes implémentées et les différences entre toutes ces structures de données.

2.2 Piles et files

Les piles et files sont toutes les deux implémentées en Java avec la classe `ArrayDeque`. Ces deux structures de données sont très proches, la seule différence portant sur la façon dont les

éléments sont retirés.

Dans le cas d'une file, on parle de *First-In-First-Out*. Cela signifie que les éléments retirés de la file sont ceux qui y ont été ajoutés en premier. Dans le cas d'une pile, on parle alors de *Last-In-First-Out*. Cela signifie que les éléments retirés de la pile sont ceux qui y ont été ajoutés en dernier.

2.3 Listes

Les listes sont probablement les structures de données les plus proches de tableaux. Plusieurs implémentations existent et il faut donc en connaître les principales différences : la sauvegarde en mémoire, et la complexité en temps pour récupérer un élément à un indice donné.

Encore une fois, il est important de savoir se retourner vers la documentation officielle pour comprendre les subtilités entre toutes les différentes implémentations.

2.4 Ensembles

Les ensembles sont des structures de données parfois sous-estimées. Ces ensembles se différencient des listes de deux façons : il n'y a pas de notion d'ordre dans un ensemble, et les ensembles garantissent l'unicité des éléments.

Les ensembles sont donc particulièrement intéressants lorsque l'on cherche à déterminer si un élément est présent dans une collection puisque la complexité en temps est de $O(1)$ contre $O(n)$ pour les `ArrayList`.

2.5 Dictionnaires

Les dictionnaires utilisent le principe de *(clé, valeur)* pour organiser des données. Il n'y a donc pas de notion d'ordre dans un dictionnaire (sauf si on utilise une `TreeMap`).

Il est donc important de savoir quand utiliser une liste ou un dictionnaire :
— La notion d'ordre est-elle importante pour moi ?
— La notion de clé est-elle importante pour moi ?

Puisque les dictionnaires n'implémentent pas d'indice, le parcours des éléments d'un dictionnaire se fait différemment de celui d'une liste.

2.6 Graphes

Les graphes sont des structures de données n'ayant pas vraiment d'implémentation directe en Java. Pour autant, ces représentations sont si importantes de nos jours qu'il est important de savoir les manipuler.

Les graphes présentant beaucoup de caractéristiques différentes (orienté, pondéré, complet, cyclique, etc.), il est important de choisir la bonne structure de données pour représenter un graphe. L'objectif de ce TP est donc aussi de vous faire réfléchir sur l'utilisation de(s) structure(s) de données adéquate(s) pour résoudre vos problèmes.

3 Création d'une blockchain

Nous découvrirons à travers ce TP les structures de données à travers le cas d'usage des transactions blockchain. Nous allons mettre en place une modélisation simple d'une cryptomonnaie « ISEP Coin ». Le fonctionnement d'une blockchain est le suivant :

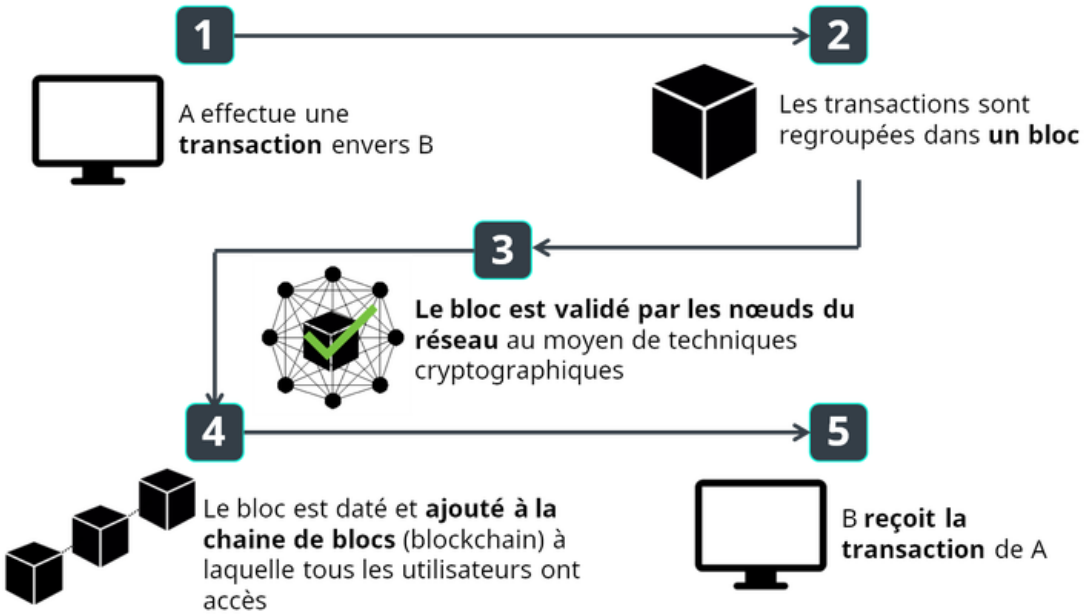

Figure 1 : Fonctionnement d'une blockchain

3.1 Exercice 1 : Modèle de données

Questions :

1. Quelle structure de données est adaptée pour représenter les transactions au sein d'un bloc de la blockchain ? Ces transactions sont à historiser dans l'ordre de leur réalisation.
2. En Java, quelle classe permet de représenter une pile et/ou une file ?

Implémentation :

Pour commencer, initialisez un nouveau module Java (TP6) avec un fichier Main et les classes suivantes :

- Wallet (Portefeuille)
    - owner : type texte, ne pouvant être modifié, correspondant au nom du propriétaire du portefeuille (getter)
    - token : type entier, ne pouvant être modifié, correspondant au numéro du portefeuille, définit aléatoirement à la création du portefeuille (getter)
    - isepCoins : type entier, correspondant au solde du portefeuille, initialisé à 0 à la création du portefeuille (getter et setter)
- Transaction
    - originWallet : type Wallet, correspondant au numéro du portefeuille émetteur de la transaction, ne pouvant être modifié (getter)
    - destinationWallet : type Wallet, correspondant au numéro du portefeuille

destinataire de la transaction, ne pouvant être modifié (getter)

- isepCoins : type entier, correspondant au montant de la transaction, ne pouvant être modifié (getter)
- payed : type booléen, indique si la transaction a été réalisée ou non (getter)

- Block
  - transactions : type pile / file de transactions, correspondant à un bloc de transaction qui sera stocké dans la blockchain (getter)

3.2 Exercice 2 : Initialisation

Questions :

1. Quelle structure de données est adaptée pour stocker les portefeuilles des membres de la blockchain ?
2. En Java, quelle structure permet de représenter un dictionnaire ?

Implémentation : vous pouvez implémenter les fonctionnalités suivantes au sein du fichier *Main.java* de votre module

3. Créez un tableau des noms de 5 élèves de votre classe.
4. Implémentez une fonction afin de créer un dictionnaire de 5 portefeuilles blockchain (Wallet) à partir du tableau de 5 élèves ; et attribuez 10 ISEP Coins à chaque portefeuille.
   - La clé du dictionnaire correspond au token du portefeuille, et la valeur associée à cette clé correspond au portefeuille correspondant.

3.3 Exercice 3 : Simulation

Questions :

1. Quelle structure de données est adaptée pour stocker les blocs de la blockchain ?
2. En Java, quelles sont les structures permettant de représenter des listes ? Quelles sont leurs caractéristiques respectives ?

Implémentation

1. Initialisez la blockchain sous le format d'une liste de blocs, à sa création la blockchain ne comporte aucun bloc.
2. Implémentez une méthode « pay » au sein de la classe Transaction qui permet de transférer des isepCoins d'un portefeuille à un autre (en se basant sur la variable statique wallets stockée dans Main). Attention les transactions sont refusées si le solde du débiteur devient négatif après la transaction. Vérifiez également que votre blockchain ne permette pas le transfert de fond d'un portefeuille vers ce même portefeuille.
3. Implémentez une méthode « add » au sein de la classe Block : la méthode ajoute une transaction à ce même bloc (et return ce bloc) ; si la taille du bloc atteint 10, la méthode utilise la fonction pay de la classe Transaction pour solder les 10 transactions en attente et créé un nouveau bloc à la fin de la liste chaînée statique blockchain.
4. Implémentez une méthode « Simulation » au sein de laquelle sont créées 55 transactions aléatoires qui sont ajoutées systématiquement au dernier bloc de la

blockchain. A l'aide de la fonction précédente, l'enchaînement des blocs et la réalisation des transactions en attente est automatique.

3.4 Exercice 4 : Analyse de la blockchain

Vous avez été hacké ! Un développeur malveillant a propagé un virus à travers votre réseau blockchain qui n'était malheureusement pas encore sécurisé avant d'être déployé. Il s'agit du premier étudiant auquel vous avez créé un portefeuille.

Implémentation :

1. Créez une fonction qui permet de visualiser (de manière antéchronologique) les N derniers bloc de transactions de la blockchain afin d'inspecter manuellement les dernières opérations et mesurer l'impact du désastre. Pour ce faire vous pourrez définir une méthode toString au niveau de Transaction et de Block.
2. Créez une fonction qui génère à partir de l'historique de la blockchain un dictionnaire qui associe chaque utilisateur au nombre de transactions qu'il a réalisé, et identifiez l'activité en nombre de transactions et en nombre de ISEP Coins de votre étudiant malveillant.
3. **Bonus** Créez une fonction qui génère un graphe qui permet de visualiser quel portefeuille a fait des échanges avec quel autre portefeuille, et identifiez ainsi tous vos étudiants impactés par l'étudiant malveillant. Pour ce faire vous pourrez notamment calculer une matrice d'adjacence entre les différents portefeuilles (dont le poids peut-être le nombre de transactions ou le volume d'ISEP Coins) ; puis dessiner ce graphe afin d'identifier tous les étudiants impactés.