Projet : Crypto Wallet Simulator

Contexte du projet

Dans l'écosystème des cryptomonnaies, chaque transaction nécessite des frais pour être validée par le réseau. Ces frais varient selon la congestion du réseau et l'urgence souhaitée. Votre mission est de développer une application console en Java 8 qui simule un wallet crypto avec mempool et aide les utilisateurs à optimiser leurs transaction fees.

Concepts blockchain essentiels à comprendre :

- Transaction : Transfert de cryptomonnaie d'une adresse à une autre
- Transaction fees (frais de transaction) : Coût payé aux mineurs pour traiter la transaction. Plus les fees sont élevés, plus la transaction est prioritaire
- Mineurs : Entités qui valident les transactions en échange des fees - ils priorisent les transactions avec les fees les plus élevés
- Wallet (Portefeuille) : Application qui gère vos adresses crypto et vos transactions
- Mempool : File d'attente des transactions en attente de validation. Votre position dépend des fees payés
- Priorité : ECONOMIQUE (lent), STANDARD (moyen), RAPIDE (rapide)

Relations Métier entre les Entités

- Une transaction peut être liée à un wallet spécifique (Bitcoin ou Ethereum)

- Une transaction possède un identifiant unique, des adresses source et destination (format selon le type de crypto : "0x" + 40 caractères pour Ethereum, ou autres formats pour Bitcoin: ça commence par "1", "3", ou "bc1"), un montant, une date de création, et les frais (fees) sélectionnés

- Une transaction possède une priorité de frais (fees) qui détermine sa position dans le mempool. Une priorité de frais (fees) peut être ECONOMIQUE (lent) ou STANDARD (moyen) ou RAPIDE (rapide)

- Une transaction évolue à travers différents statuts (PENDING, CONFIRMED, REJECTED) selon son traitement par les mineurs

- Une transaction contient les informations nécessaires pour calculer ses frais, estimer son temps de confirmation, et afficher ses détails

- Un wallet contient plusieurs transactions et peut créer de nouvelles opérations selon son type de cryptomonnaie (BITCOIN ou ETHEREUM)

- Le mempool regroupe toutes les transactions en attente et calcule leur position selon les frais (fees) payés, créant une file d'attente où les transactions les mieux rémunérées passent en premier

- Chaque type de cryptomonnaie (BITCOIN, ETHEREUM) nécessite sa propre logique de calcul de frais (fees), avec des paramètres spécifiques : Bitcoin utilise taille estimée en bytes × tarif satoshi par byte, tandis qu'Ethereum utilise limite de gas × prix du gas

Note : Utiliser des valeurs fictives cohérentes pour les paramètres de calcul (taille de transaction, tarifs, prix du gas) en respectant les ordres de grandeur réalistes

Structure de l'application :

- Couche de présentation (UI/Menu)
- Couche métier
- Couche de données (Repository Pattern)
- Couche utilitaire
- Vous pouvez ajouter d'autres couches que vous voyez pertinentes

Algorithme principal à implémenter :

Simulateur de Mempool et Fee Calculator :

- Générer un mempool avec 10-20 transactions aléatoires en attente

- Calculer le numéro de position de votre nouvelle transaction selon ses fees

- Estimer le temps : numéro de Position × 10 minutes

Un Menu interactif

Un Menu interactif dans la console à mettre de votre propre manière et contenant les fonctionnalités principales suivantes :

- Créer un wallet crypto
- Créer une nouvelle transaction
- Calculer la position dans le mempool et temps d'attente estimé
- Comparer les 3 fee levels avec position réelle dans la file

Explication des Fonctionnalités Principales :

1. Créer mon wallet

- Choisir le type : BITCOIN ou ETHEREUM
- Générer automatiquement une adresse crypto unique (format selon le type choisi)
- Initialiser le solde à zéro
- Créer le wallet avec un ID unique

2. Créer une nouvelle transaction

- Préciser : adresse source, adresse destination, montant
- Choisir le niveau de frais (fee level) : ÉCONOMIQUE, STANDARD ou RAPIDE
- Effectuer les validations nécessaires
- Calculer les frais (fees) selon le type de cryptomonnaie et la priorité choisie

- Créer une transaction complète avec statut PENDING
- Attribuer un UUID unique à cette transaction

3. Voir ma position dans le mempool

- Demander la position dans la file d'attente
- Calculer la position de la transaction selon ses frais (fees)
- Afficher : "Votre transaction est en position X sur Y"
- Estimer le temps selon la position

4. Comparer les 3 niveaux de frais (fee levels)

- Voir les différences concrètes entre les options (ÉCONOMIQUE, STANDARD ou RAPIDE)
- Calculer la position dans le mempool pour chaque niveau de frais (fee level)
- Montrer le compromis coût/rapidité
- Afficher le tout dans un tableau console avec traits ASCII comparatif avec positions et temps estimés. Une autre maniere d'affichage du tableau est libre à vous de le faire.

5. Consulter l'état actuel du mempool

- Avant affichage générer automatiquement des transactions aléatoires pour simuler l'activité réseau
- Afficher la liste des transactions en attente dans le mempool
- Montrer les frais (fees) payés par chaque transaction
- Mentionner ma transaction dans le pool

exemple :

=== ÉTAT DU MEMPOOL ===

Transactions en attente : 18

Transaction (autres utilisateurs)	Frais
0x742d35... (anonyme)	8.50$
0x8a5f92... (anonyme)	5.20$
0x1c4e67... (anonyme)	2.80$
>>> VOTRE TX: 0x9b2a14...	2.00$
0x3f8d91... (anonyme)	1.50$

Spécifications techniques :

Utiliser ArrayList/HashMap pour stockage et recherche rapide

Enums obligatoires pour représenter les constantes métier (priorités, statuts, types)

Intégrer Java Time API pour la gestion des dates et durées

Gestion des exceptions (try-catch) pour les cas d'erreur

Les données persistent en base de données postgres via le pilote JDBC de postgres

Validations obligatoires : montants positifs, format des adresses, niveaux valides, ...etc

Logging : Utiliser java.util.logging, SLF4J ou Logback pour logger les erreurs, calculs de fees, et opérations critiques. System.out.println autorisé UNIQUEMENT pour l'affichage du menu et interactions utilisateur, INTERDIT pour les erreurs

Stream API : Utilisation obligatoire de map, filter, reduce, Optional

Débogage : Maîtriser les techniques de débogage pour investiguer et résoudre les problèmes

Gestion de projet :

- Utiliser Git avec stratégie de branches multiples

- Utiliser JIRA pour la planification

- IDE au choix : Eclipse, VSCode, IntelliJ IDEA, ...

- Plugin à utiliser : SonarLint et un UML generator plugin

- Raccourcis clavier essentiels à maîtriser :

  - Run project, Debug project, Rechercher fichier, Rechercher dans fichiers, Refactoring, Aller à la déclaration, d'autres raccourcis si vous voyez pertinents

Bonus (si vous avez terminé ce qui est au-dessus) :

- Couverture de tests via des tests unitaires : mettre au minimum 2 tests unitaires en créant "src/test/java" et travailler avec JUnit sachant qu'il faut importer manuellement le jar nécessaire. Maven autorisé si les choses sont difficiles à faire.

- Export des transactions en CSV

- Docker : Containerisation de l'application avec Dockerfile. Maven autorisé si les choses sont difficiles à faire.

Exigences :

- JDK 8 : Interdit de travailler avec une version autre que 8

- Maven interdit : Utiliser uniquement javac et java

- Implémenter les principes SOLID

- Design Patterns obligatoires : Singleton, Repository Pattern

- Interfaces obligatoires : Identifier et implémenter des interfaces appropriées pour respecter les principes SOLID

- Héritage obligatoire : Identifier et implémenter une hiérarchie de classes logique

Anti-patterns à éviter :

- God Class

- Mélanger la logique métier avec l'affichage

- Couplage fort

- Violation de l'encapsulation

- Code dupliqué

- Magic numbers sans constantes

- Utilisation de String/int au lieu d'enums pour les constantes

Modalités pédagogiques :

Projet individuel

Durée : 7 jours

Début : 22/09/2025, Deadline : 30/09/2025 avant minuit

Modalités d'évaluation

- Présentation client et technique

- Démonstration des fonctionnalités de l'application

- Évaluation des savoirs (Q/A)

- Mise en situation

- Review de code et bonnes pratiques

Livrables

Lien GitHub contenant :

- Le code source complet sur un dépôt Git avec branches multiples

- Le fichier JAR exécutable de l'application

- Le diagramme de classe UML

- Script SQL

- Le fichier README.md contenant :

- Description du projet
- Technologies utilisées
- Structure du projet
- Prérequis et installation
- Guide d'utilisation
- Captures d'écran

Critères de performance

- L'application doit être développée avec Java 8 exclusivement
- L'application doit être fonctionnelle et répondre à toutes les exigences
- La structure en couches doit être respectée
- Le code doit être propre, bien commenté et suivre les conventions Java
- Logs appropriés avec distinction System.out (affichage) vs Logger (erreurs/opérations)
- Utilisation correcte des enums pour éviter les magic strings/numbers
- L'utilisation de Git doit montrer une progression avec branches multiples
- Le README.md doit être clair et complet
- Le diagramme UML doit refléter fidèlement la structure
- Respect des principes SOLID et des Design Patterns demandés