# Réseau en ligne avec un remplissage uniTP3 (12%) – Modélisation d’un réseau social

**Consignes :**

1. Ce travail se fait **individuellement**.
2. Le **plagiat** est formellement interdit, les sanctions prévues par le collège s’appliqueront.
3. L’**utilisation de l’intelligence artificielle** pour générer du code, que ce soit partiellement ou en totalité, est formellement interdite. Vous devez en tout temps être capable de justifier votre code.

*Toutefois, l’utilisation de l’intelligence artificielle pour générer de la documentation (javadoc) est permise, mais vous devez le déclarer au début de votre documentation dans chaque classe impactée.*

1. La grille des critères d’évaluation se trouvent à la fin de ce document.

**Mise en contexte**

Pendant une période électorale, le réseau social fictif CivixNet est inondé de publications politiques. Certaines d'entre elles contiennent des informations fausses ou manipulées, volontairement diffusées pour influencer les électeurs. Vous devez concevoir un programme qui simule la circulation de ces publications dans le réseau afin d’en comprendre l’impact.

Les réseaux sociaux sont souvent modélisés dans des objets mathématiques nommés « graphes » ([Théorie des graphes — Wikipédia](https://fr.wikipedia.org/wiki/Th%C3%A9orie_des_graphes)) qui sont constitués de nœuds pour représenter les agents de propagation dans un système et d’arêtes pour représenter les liens entre les agents.

**Partie 0 – Structure du projet et Github**

* Ouvrez le projet fourni dans IntelliJ. Il est normal qu’il y ait des erreurs, certaines classes sont en partie implémentées pour vous, mais ont besoin de d’autres classes que vous ajouterez pour fonctionner. Voir la partie 5 pour l’organisation finale du travail.
* Créez un dépôt nommé « TP3-VotreNom » sur Github et liez-le au code fourni. Ajoutez également un README. Ensuite, tout au long du travail, vous devrez faire des commit + push régulièrement. Lors de la correction, je regarderai l’historique des commit! Assurez-vous également que vos messages de commit sont significatifs.

**Partie 1 – Classe Utilisateur**

Vous devez d’abord créer et coder la classe qui implémente un utilisateur du réseau social fictif CivixNet. Voici les caractéristiques de base la classe :

* Une image contenant texte, capture d’écran, Police, nombre

  Le contenu généré par l’IA peut être incorrect.Un utilisateur a un nom d’utilisateur et un mot de passe.
* Le nom d’utilisateur doit avoir un maximum de 15 charactères.
* Le mot de passe doit avoir un minimum de 12 charactères, incluant au moins 1 chiffre, au moins 1 lettre minuscule et une lettre majuscule.
* Il n’est pas possible de créer un utilisateur par défaut.
* Tous les attributs doivent respecter les principes de l’encapsulation.
* La comparaison se fait via l’ordre alphabétique de *username*.
* Le formattage du toString() est, par exemple :

Username: alice (Mode de passe: Alice123secure)

\*Le diagramme de classe ne montre que les éléments publics, à vous d’implémenter les éléments privés nécessaire pour que toutes les fonctionnalités soient assurées.

**Vous devez entièrement tester les méthodes de votre classe Utilisateur avec JUnit ainsi qu’ajouter la javaDoc à la classe Utilisateur.**

**Partie 2 – Classe CivixNet**

Vous coderez ensuite la classe qui modélise le réseau social fictif CivixNet (ne pas changer les signatures des méthodes dans le fichier CivicNet.java). Voici les caractéristiques de la classe :

* La classe est modélisée avec une TreeMap qui crée une association entre chaque utilisateur et l’ensemble des utilisateurs auxquels il est abonné. L’ordre des abonnées n’a pas d’importance, donc on utiliser un HashSet.
* Puisqu’il s’agit d’un TreeMap, vos utilisateurs doivent implémenter la classe Comparable.
* Il est possible d’ajouter/retirer un abonnement à un utilisateur un à la fois, ou en fournissant une liste d’abonnements.
* Une méthode permet de retrouver un Utilisateur à partir d’un nom d’utilisateur.
* Une méthode permet de déterminer si 2 utilisateurs sont abonnés l’un à l’autre.
* Une méthode permet d’évaluer les utilisateurs impactés par la propagation d’une fausse information (celle-ci sera détaillée à la partie 4, donc attendre avant de la faire).
* Une image contenant texte, Police, capture d’écran, algèbre

  Le contenu généré par l’IA peut être incorrect.Une méthode toString() permet d’afficher les utilisateurs et les gens qu’ils suivent :
* Un fichier de tests vous est fourni pour vous permettre de valider vos méthodes. Celui-ci sera entre-autre utilisé pour la correction.

**Partie 3 – Classe ReseauBuilder**

1. Vous coderez également la méthode statique *chargerDepuisJSON* de la classe ReseauBuilder qui construit un CivixNet à partir d’un fichier json (voir le fichier dans la package donnees). Vous devrez résoudre par vous-même les problèmes de lecture de fichier et tester le bon fonctionnement, donc pensez à :

* Vérifier votre path in et la structure de votre projet
* Utiliser le déboggueur!

1. En plus de charger depuis un JSON un ReseauBuilder peut aussier sérialiser et désérialiser un objet CivixNet (format .ser, voir les notes de cours 8.1) afin de pouvoir sauvegarder un réseau complet. Les chaînes de charactères dans les méthodes serialise et deserialise sont le pathIn et le pathOut. Le fichier Main.java vous donne quelques tests.

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Une fois les 3 premières partie terminée, vous pouvez exécuter la classe CivixNetSwingUI dans le package app pour voir le résultat du graphe construit et valider le bon fonctionnement de votre code. Voici le résultat pour le fichier reseau.json.

Une image contenant capture d’écran, diagramme, texte, cercle

Le contenu généré par l’IA peut être incorrect.

En cliquant sur chacun des nœuds du graphe, on peut voir les informations de l’utilisateur sélectionné. Des flèches indiquent le sens de la connexion (qui est abonné à qui).

**Partie 4 – Propagation fausse information**

Il est maintenant temps de coder la méthode propagationFausseInformationRecursive! Cette méthode **récursive** prend en entrée le username d’un utilisateur qui commence la propagation d’une fausse information et retourne un ArrayList de tous les utilisateurs affectés (la personne qui démarre la propagation est considérée comme affectée également). Voici les principe de propagation :

* La fausse information se propage de la source (niveau 0) aux utilisateurs directement connectés (niveau 1)
* De ces utilisateurs, elle se propage encore une fois aux utilisateurs directement connectés à ceux-ci (niveau 2), mais la propagation s’arrête ensuite.
* Un utilisateur ne peut pas être ajouté deux fois à la liste.
* La liste finale est triée en ordre alphabétique inverse de username (z 🡪 a).

**Partie 5 – Remise**

ATTENTION LA REMISE COMPORTE 2 PARTIES : UNE SUR LÉA ET UNE SUR GITHUB

**Sur LÉA :**

*Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.*À remettre (dans un dossier compressé .zip) :

* Votre projet IntelliJ au complet

Où remettre?

* Sur LÉA dans *Travaux > TP3 – Modélisation d’un réseau*

**Sur GitHub :**

* Ajoutez-moi comme collaboratrice (voir Formatif 3).

**Quand remettre?**

* Date limite : lundi 21 avril 23h59

|  |  |  |  |
| --- | --- | --- | --- |
| Capacité | Critères d’évaluation | Éléments évalués | Points (sur 60) |
| 1  (20 pts) | Utiliser des interfaces pour des manipulations de données | * Les classes pour lesquelles c’est pertinent implémentent l’interface Comparable. * Les classes pour lesquelles c’est pertinent implémentent l’interface Serializable. | /4 |
| Utiliser l’encapsulation au maximum | * La classe Utilisateur respecte les principes de l’encapsulation et présente tous les éléments demandés. | /4 |
| Utiliser la mécanique de gestion des exceptions | * Les exceptions de type approprié sont lancées aux endroits appropriés. | /3 |
| Programmer du code OO fonctionnel et optimisé | * **L’application CivixNetSwingUI peut être exécutée sans erreur.** * **Le code est propre et optimisé.** | /8 |
| Programmer du code OO complétement documenté | * La JavaDoc est présente et complète pour la classe Utilisateur. | /1 |
| 2  (30pts) | Développer des algorithmes fonctionnels permettant de parcourir des structures de données.  Programmer des algorithmes fonctionnels qui impliquent des collections imbriquées et hétérogènes.  Utiliser les algorithmes de tri et sélection proposés par les librairies du langage de programmation | * La classe CivixNet est entièrement fonctionnelle et respecte la documentation fournie. ***Validez vos fonctionnalités avec les tests unitaires fournis!*** | /12 |
| Récupérer et emmagasiner de manière fonctionnelle les données nécessaires dans des fichiers | * Le chargement depuis un fichier JSON est fonctionnel | /10 |
|  | Sérialiser et désérialiser efficacement les données pertinentes d’un objet | * Implémentation fonctionnelle de la sérialisation et désérialisation | /6 |
|  | Utiliser continuellement les types génériques avec les différentes collections | * Utilisation des types génériques lors de la déclaration de List, Set, Map | /2 |
| 3  (10pts) | Configurer le gestionnaire de versions  Partager son travail avec une équipe  Écrire des messages adéquats dans le gestionnaire de versions | * Le projet est correctement configuré et partagé sur Github. * Présence du README. * Historique qui montre des commit réguliers et une évolution logique du code. | /6 |
| Produire des tests complets et pertinents en utilisant un outil de test unitaire. | * Les tests (JUnit) pour la classe Utilisateur sont présents, fonctionnels et couvrent suffisamment de cas possibles. | /4 |