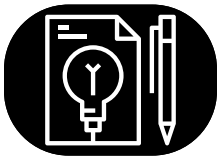
Projet de Programmation

Pour le 22/12/2023



Installation et gestion de bornes électriques

Sujet	Avec la généralisation des véhicules électriques, il est important d'installer des bornes de recharge de véhicules électriques accessibles, mais sans en installer trop (ce qui augmente le coût global de leur installation et entretien). On suppose que le responsable de l'aménagement d'une communauté d'agglomération fait appel `a nous pour réaliser un logiciel qui permettrait de déterminer quelles villes dans la communauté doivent accueillir un parking équipé de bornes de recharge.
Contraintes	• (**Accessibilité)** Chaque ville doit posséder ses bornes, ou être directement reliée à une ville qui possède des bornes. • **(Économie)** Le coût du projet doit être le plus bas possible, ce qui signifie que le nombre de bornes à construire doit être le plus petit possible
Auteurs	**CHAKER** Zakaria, **HAMOUCHI** Nabile, **HULKHOREE** Aaishah

Description

Ce projet vise à développer un système de gestion des zones de recharge pour une communauté d'agglomération, permettant de modéliser les villes, les routes entre les villes (à sens bidirectionnel), ainsi que la gestion des zones de recharge associées à chaque ville.

Structure du Projet

Le programme est organisé en plusieurs classes qui interagissent entre elles, pour permettre la configuration des zones de recharge au sein de la communauté d'agglomération.

Prérequis avant d’utiliser le programme

1. Assurez vous d'avoir Java installé sur votre système.

2. Exécutez la classe Main pour démarrer le programme.

3. Suivez les instructions affichées dans la console pour configurer les zones de recharge et consulter les informations.

Les 5 Classes Principales

CommunauteAgglomera-tion (CA)	Classe qui représente l'ensemble de la communauté d'agglomération, elle gère les villes, les connexions routières entre les villes et les zones de recharge s'il y en a.
Main	Classe principale qui interagit avec l'utilisateur pour gérer les villes, les routes et les zones de recharge.
Ville	Classe qui représente une ville de la CA et gère les informations sur la ville et ses zones de recharge.
ZoneRecharge	Classe générique définissant si une zone est une zone de recharge ou non.
Route	Classe qui modèle une route reliant deux villes et permet d'accéder à des informations sur ces connexions.

Les méthodes et leur description de chaque classe

CommunauteAgglomeration (CA)	
Méthodes	**Description**
setAdjacence((Map<Ville, List<Ville>> adjacence)	Définit la liste d'adjacence pour les villes.
getAdjacence()	Récupère la liste d'adjacence des villes.
ajouterVille(Ville ville)	Ajoute une nouvelle ville à la liste des villes de la communauté d'agglomération.
afficherVille()	Affiche la liste des villes présentes dans l'agglomération
ajouterRoute(Ville villeA, Ville villeB)	Crée une route entre deux villes déjà existantes.
ajouterRoute(Scanner scanner)	Gère l'ajout de routes entre les villes à travers un menu interactif.
afficherListeAdjacence()	Affiche la liste d'adjacence qui représente les connexions entre les différentes villes.
ajouterZoneRecharge(Ville ville)	Ajoute une zone de recharge à une ville, entrée en paramètre.
retirerZoneRecharge(Ville ville)	Supprime la zone de recharge d'une ville entrée en paramètre.
afficherVillesAvecZonesRecharge()	Affiche les villes ayant des zones de recharge.
creerVilles(Scanner scanner)	Permet à l'utilisateur de créer de nouvelles villes pour l'agglomération.

gererRoutes(Scanner scanner)	Permet de gérer les routes entre les villes à l'aide d'un menu interactif
verificationUtilisateurVille(Scanner scanner, String nomVille)	Demande à l'utilisateur de confirmer son choix pour une ville spécifique
ajouterZoneRecharge(Scanner scanner)	Permet d'ajouter une zone de recharge pour une ville spécifiée par l'utilisateur
retirerZoneRecharge(Scanner scanner)	Permet de retirer une zone de recharge à la demande de l'utilisateur.
verifierContrainteAccessibilite()	Vérifie la contrainte d'accessibilité pour toutes les villes.
gererZonesRecharge(Scanner scanner)	Gère les zones de recharge à l'aide d'un menu interactif.
lireFichier(String cheminFichier)	Lit un fichier entré en paramètre pour initialiser les villes et les routes entre elles.
calculerScore()	Calcule le score total représentant le nombre total de zones de recharge dans l'ensemble des villes.
algorithmeContrainteEconomie(int nbLancement)	Propose un algorithme pour optimiser les zones de recharge selon la contrainte d'économie.
resoudreManuellement(Scanner scanner)	Permet de résoudre manuellement la gestion des zones de recharge dans la CA.
valMax()	Renvoie la valeur maximale possible pour les entiers en Java.
resoudreAutomatiquement(Scanner scanner)	Résout automatiquement la gestion des zones de

	recharge.
resoudreAutomatiquement()	Version surchargée de la méthode resoudreAutomatiquement()
sauvegarderSolution(String cheminFichier)	Sauvegarde la solution actuelle dans un fichier spécifié en format texte.
verifierFichierValide(String cheminFichier)	Vérifie si un fichier existe, qu'il soit lisible et valide.
afficherMenuIDE(Scanner scanner)	Affiche un menu interactif pour permettre au concepteur de mieux maintenir le programme en marche sur un IDE.
trouverVille(String nomVille)	Recherche une ville dans la liste en fonction de son nom entré en paramètre.
trouverIndiceVille(List<Ville> listeVilles, String nomRecherche)	Trouve l'indice d'une ville dans la liste des villes en fonction de son nom.
afficherMenu(Scanner scanner, String chemin)	Sert à afficher un menu interactif dans la console de votre terminal avec les commandes que vous retrouverez dans la section **UTILISATION DU PROGRAMME** et à gérer les actions associées à chaque choix.
routeExisteDeja(Ville ville1, Ville ville2)	Vérifie si une route entre deux villes existe déjà.

Route: La classe Route permet de représenter et de gérer les connexions entre deux villes. Elle est utilisée pour définir les villes associées à une route spécifique.	
Méthodes	**Description**
getNom()	Retourne le nom de la route.
getVilleA()	Retourne la première ville reliée par la route
setVilleA(Ville villeA)	Définit la première ville reliée par la route.
getVilleB()	Retourne la deuxième ville reliée par la route.
setVilleB(Ville villeB)	Définit la deuxième ville reliée par la route.
toString()	Redéfinition de la méthode toString() pour obtenir une représentation textuelle de la route.

Main: La classe principale du programme.	
Méthodes	**Description**
main	Point d'entrée principale du programme qui permet à l'utilisateur de gérer et configurer les villes, les routes et les zones de recharge au sein de la communauté d'agglomération.

ZoneRecharge	
Méthodes	**Description**
getZoneRecharge()	Retourne l'état de la zone de recharge (si elle existe ou pas).
setZonesRecharge(boolean zonesRecharge)	Définit l'état de la zone de recharge.

Ville : La classe Ville permet de représenter une ville avec ses attributs et la gestion des zones de recharge associées à cette ville.	
Méthodes	**Description**
getNom()	Retourne le nom de la Ville.
ajouterZoneRecharge()	Active la zone de recharge pour la ville prise en charge.
retirerZonesRecharge()	Désactive la zone de recharge pour la ville prise en charge.
setZonesRecharge(boolean zonesRecharge)	Définit l'état des zones de recharge.
getZonesRecharge()	Retourne l'état actuel des zones de recharge.
hashCode()	Retourne le code de hachage de l'objet basé sur le nom de la ville.
equals(Object obj)	Vérifie si deux objets Villes sont égaux en comparant leurs noms de ville.

Utilisation du programme

- Sur le terminal de votre PC:

- Il faudra d'abord aller à la racine du projet. Puis exécuter la commande suivante sur le terminal de votre PC :
  - Sous Windows , il faut taper les commandes:

javac -d bin src\projet\CommunauteAgglomeration.java src\projet\Main.java src\projet\Ville.java src\projet\Route.java
(1)

Puis, entrez, ensuite si vous avez des erreurs, celles-ci vous serrons informés. Ici, dans notre code, vous retrouverez des erreurs d'encodages UTF-8 qui n'interfère pas sur la résolution en bon et due forme de notre programme.

Après avoir analysés ces potentielles erreurs, tapez la commande suivante:

java -cp bin projet.Main [Le chemin absolu du fichier à lire] (2)

- Sous Linux/MacOs:

Reprenez les mêmes indications que sous Windows en modifiant les commandes (1) et (2) par celles-ci:

javac -d bin src/projet/CommunauteAgglomeration.java src/projet/Main.java src/projet/Ville.java src/projet/Route.java (1)

java -cp bin projet.Main [Le chemin absolu du fichier à lire] (2)

Fonctionnalités non implémentées

Tout au long de notre projet, nous avons rencontré de nombreuses difficultés dans la réalisation d'un programme efficace et qui fonctionne correctement en respectant les contraintes imposées (Accessibilité et Économie). Comme vous le constaterez, nous avons mis l'accent sur la gestion des erreurs et la mise en œuvre d'une interface utilisateur/machine opérationnelle. Les fonctionnalités qui n'ont pas été implémenté sont les suivantes:

- Pour la partie **Interface**, nous aurions aimé ajouter:
  - Boites de dialogues simples avec boutons et autres: Cela aurait permis à l'utilisateur une gestion de sa communauté beaucoup plus agréable et facile d'utilisation.
  - Représentation de la communauté d'agglomération sous forme d'un graphe simple au niveau du Bilan.
  - Concernant la partie de **Sauvegarde des fichiers**, nous aurions aimé ajouter:
    - Un ***explorateur de fichiers*** pour que l'utilisateur puisse, dans un premier temps choisir son fichier à résoudre que ce soit manuellement ou

automatiquement puis le sauvegarder plus facilement à l'aide de cet outil n'importe où.

- Pour la partie **Code**, nous aurions aimé ajouter:
  - Plus de méthodes de contrôle et de confirmation auprès de l'utilisateur sur les informations saisies sur la console (confirmez vous les informations saisie? o/n...) ; il y a des méthodes qui en sont dotées...