CAS ROSSE

Présentation du contexte

Promulguée le 12 juillet 2010, la loi portant engagement national pour l'environnement, dite « Grenelle 2 », est un texte d'application et de territorialisation du Grenelle Environnement et de la loi Grenelle 1. Un des chantiers de cette loi porte sur le développement des véhicules électriques et hybrides rechargeables, en favorisant l'émergence de l'offre industrielle nationale, en stimulant la demande et en encourageant la possibilité de créer et d'entretenir des infrastructures de recharge électrique nécessaires à l'usage de ces véhicules.

Bien que la plupart des infrastructures de recharge va relever de la sphère privée (90%), les bornes de recharge accessibles au public, placées dans des parkings ou sur voirie, offriront l'assurance aux utilisateurs de pouvoir y accéder en dehors de cette sphère privée (domicile, travail) et des stations services. Elles constituent un gage de fiabilité de l'ensemble du système, complément indispensable pour encourager l'utilisation du véhicule électrique.

Les communes sont naturellement impliquées dans le déploiement de ces bornes, en raison du fort impact sur la voirie et les places de stationnement.

La ville de ROSSE fait partie des douze agglomérations pilotes appelées à déployer une première vague d'infrastructures de recharge pour véhicules hybrides et électriques.

Elle a confié à la société prestataire de services Chargéon la mise en place et l'exploitation d'un réseau de points de recharge sous forme de bornes intelligentes standardisées. Soucieuse de se forger une expérience solide sur ce marché émergent, la société Chargéon fait évoluer son système d'information, colonne vertébrale permettant de réaliser les principales opérations nécessaires au bon fonctionnement des différents sous-systèmes de l'infrastructure de recharge.

En tant que développeur d'applications, vous participez aux différentes missions liées à ce projet.

Maintenance préventive des bornes

Documents à utiliser : annexes A, B et C

Les techniciens doivent également mener des actions de maintenance préventive sur les bornes de recharge. Ces révisions sont fonction du type de borne. Elles sont programmées à intervalles de temps réguliers, mais peuvent aussi être déclenchées lors de l'atteinte d'un seuil d'utilisation.

La solution informatique doit permettre chaque mois de répartir équitablement les tâches de maintenance préventive sur l'ensemble des techniciens. Cette partie de l'application doit être réalisée à l'aide d'un langage orienté objet. Un extrait du diagramme de classes utilisé est présenté en ***annexe A***, leur description littérale en ***annexe B***. L'ensemble des objets est instancié à partir de la base de données dès le lancement de l'application.

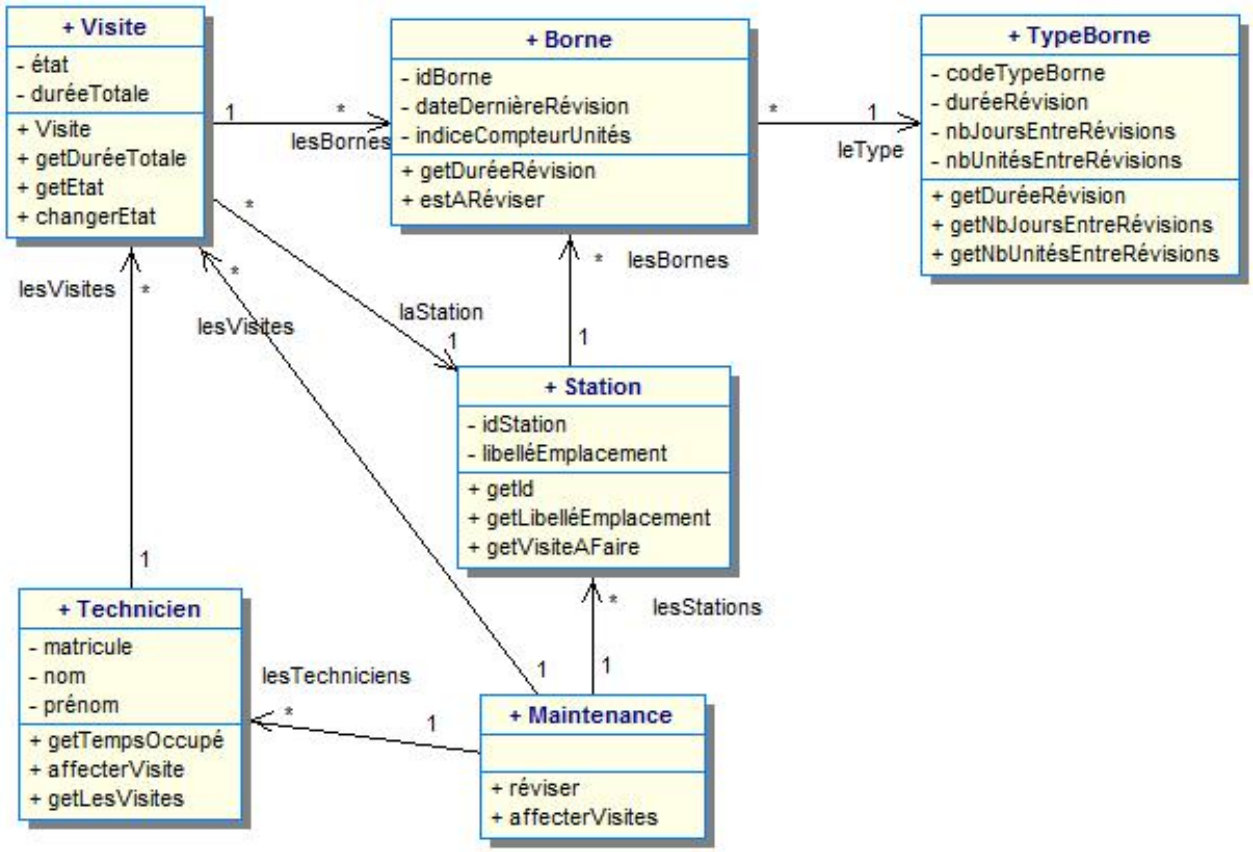
La classe « Maintenance » est chargée de programmer les visites de stations à réaliser dans le mois, puis de les affecter aux techniciens. Une visite concerne une station, et précise la durée totale nécessaire pour réaliser les révisions sur les bornes.

La répartition des visites aux différents techniciens doit être équitable en affectant chaque nouvelle visite au technicien actuellement le moins occupé en temps total de maintenance préventive.

Les classes techniques « Collection » et « Date » sont présentées en ***annexe C***.

Travail à faire	
1	Écrire la méthode « getDuréeRévision » de la classe « Borne ».
2	Écrire la méthode « estARéviser » de la classe « Borne ».
3	Écrire le constructeur de la classe « Visite ».
4	Écrire la méthode « getVisiteAFaire » de la classe « Station ».
5	Écrire la méthode « affecterVisites » de la classe « Maintenance ». *Le candidat est libre d'ajouter et d'écrire toutes les méthodes (non citées dans les annexes) qu'il juge nécessaires.*

Annexe A – Diagramme partiel des classes métiers



<u>Remarque</u> : Les paramètres des méthodes ne sont pas présentés sur ce diagramme.

Annexe B – Description littérale des classes métiers

Classe Station

Attributs privés :

idStation : Entier *// identifiant de la station*

libelléEmplacement : Chaîne *// libellé de l'emplacement de la station*

lesBornes : Collection de Borne *// les bornes de la station*

Méthodes publiques :

Fonction getId() : Entier *// retourne l'identifiant de la station*

Fonction getLibelléEmplacement() : Chaîne*// retourne le libellé de l'emplacement*

Fonction getVisiteAFaire() : Visite

// retourne une instance de classe Visite recensant toutes les bornes de la station
// qui nécessitent d'être révisées, ou null s'il n'y a aucune borne à réviser

FinClasse

Classe TypeBorne

Attributs privés :

codeTypeBorne : Chaîne *// code du type de borne*

duréeRévision : Entier *// durée en minutes requise pour réaliser*
// la révision sur les bornes de ce type

nbJoursEntreRévisions : Entier
// nombre de jours qui séparent deux révisions successives d'une borne de ce type

nbUnitésEntreRévisions : Entier
// nombre d'unités de recharge au-delà duquel il faut envisager une nouvelle révision

Méthodes publiques :

Fonction getDuréeRévision() : Entier
// retourne la durée en minutes requise pour réaliser la révision sur les bornes de ce type

Fonction getNbJoursEntreRévisions() : Entier
// retourne le nombre de jours au-delà duquel il faut envisager une révision
// sur les bornes de ce type

Fonction getNbUnitésEntreRévisions() : Entier
// retourne le nombre d'unités de recharge au-delà duquel il faut envisager une révision
// sur les bornes de ce type

FinClasse

Classe Borne

Attributs privés :

idBorne : Entier *// identifiant de la borne*

dateDernièreRévision : Date *// date de la dernière révision effectuée sur la borne*

indiceCompteurUnités : Entier
// nombre d'unités de recharge délivrées depuis la dernière révision,
// ce compteur étant remis à zéro suite à chaque révision

leType : TypeBorne *// type de la borne*

Méthodes publiques :

Fonction getDuréeRévision() : Entier
// retourne la durée en minutes requise pour réaliser la révision sur la borne,
// cette durée étant fonction du type de la borne

Fonction estARéviser() : Booléen
// retourne vrai lorsque la borne doit être révisée, soit parce que le temps qui sépare
// deux révisions pour ce type de borne s'est écoulé depuis la date de la dernière
// révision, soit parce que le nombre d'unités de recharge délivrées par la borne
// depuis la dernière révision a atteint le seuil établi pour ce type de borne ;
// retourne faux sinon

FinClasse

Classe Visite

Attributs privés :

état : Caractère
// état de la visite : 'p' pour programmée, 'a' pour affectée, 'r' pour réalisée

duréeTotale : Entier
// durée totale en minutes requise pour réaliser l'ensemble des révisions
// prévues sur les bornes de la station

laStation : Station
// la station concernée par la visite

lesBornes : Collection de Borne
// la collection des bornes de laStation concernées par la visite

Méthodes publiques :

Visite(lesBornesAVisiter : Collection de Borne, uneStation : Station)
// constructeur valorisant les attributs privés de la classe Visite, y compris l'état et la
// durée totale de la visite

Fonction getDuréeTotale() : Entier
// retourne la durée totale en minutes requise pour réaliser l'ensemble
// des révisions prévues sur les bornes de la station

Fonction getEtat() : Caractère
// retourne l'état de la visite

Procédure changerEtat()
// modifie l'état de la visite, de 'p' programmée à 'a' affectée, ou de 'a' affectée à 'r' réalisée

FinClasse

Classe Technicien

Attributs privés :

matricule : Entier — *// matricule du technicien*
nom : Chaîne — *// nom du technicien*
prénom : Chaîne — *// prénom du technicien*
lesVisites : Collection de Visite — *// ensemble des visites affectées au technicien*

Méthodes publiques :

Fonction getTempsOccupé() : Entier
// retourne la durée totale en minutes des visites affectées au technicien

Procédure affecterVisite(uneVisite : Visite)
// ajoute la visite uneVisite dans les visites affectées au technicien

Fonction getLesVisites() : Collection de Visite
// retourne l'ensemble des visites affectées au technicien

FinClasse

Classe Maintenance

Attributs privés :

lesStations : Collection de Station *// l'ensemble des stations*

lesTechniciens : Collection de Technicien *// l'ensemble des techniciens*

lesVisites : Collection de Visite *// l'ensemble des visites à réaliser*

Méthodes publiques :

Procédure réviser()

// Etablit l'ensemble des visites à réaliser sur les stations

Procédure affecterVisites()

// Affecte les visites à réaliser aux techniciens, en répartissant équitablement le travail
// entre les techniciens. Chaque visite est affectée au technicien le moins occupé en
// temps total de maintenance préventive. L'état de chaque visite doit être mis à jour.

FinClasse

Annexe C – Description des classes techniques

Classe Collection de <nom de la classe>

Méthodes publiques

Fonction cardinal() : Entier

// Renvoie le nombre d'objets de la collection

Fonction obtenirObjet(unIndex : Entier) : Objet de la classe

// Retourne l'objet d'index unIndex, le premier objet de la collection a pour index 1

Procédure ajouter(unObjet : Objet de la classe)

// Ajoute un objet à la collection

FinClasse

Pour instancier une collection :

uneCollection : Collection de <classe>

uneCollection = new Collection() de <classe>

Pour parcourir par itération les éléments d'un objet Collection, il est possible d'utiliser :

Pour chaque <objet> dans <collection> faire

// instructions avec <objet>

FinPour

Classe Date

Attributs privés :

année : Entier

mois : Entier

jour : Entier

Méthode publique à portée classe :

fonction aujourdhui() : Date *// renvoie la date du jour*

// Exemple d'appel : an = Date.aujourdhui().année()

// la variable entière an reçoit l'année de la date du jour

Méthodes publiques :

fonction année() : Entier *// renvoie l'année*

fonction mois() : Entier *// renvoie le mois*

fonction jour() : Entier *// renvoie le jour*

fonction différence(uneDate : Date) : Entier

// renvoie le nombre de jours de différence entre l'objet Date courant et le paramètre uneDate
// si l'objet Date courant correspond à une date postérieure au paramètre uneDate,
// le nombre de jours retourné est positif.
// Dans le cas contraire, le nombre de jours retourné est négatif.

Fin Classe Date