**Annexes à utiliser : annexes a, b et c²**

**Suivi des entretiens dans une agence**

Chaque agence dispose d’un parc de véhicules dont elle doit assurer l’entretien courant : vidanges, contrôle du liquide de freins, niveau d’huile de la boîte de vitesse, etc.

L’agence de Bordeaux, du fait de son envergure, possède un atelier dans lequel six employés assurent l’entretien de la centaine de véhicules rattachés à l’agence.

Les distances parcourues par les véhicules loués peuvent être très importantes ; il n’est pas rare de trouver une voiture ayant un « kilométrage compteur » dépassant 50 000 km après trois mois d’utilisation. Le suivi des entretiens, essentiellement ceux devant être réalisés en fonction du kilométrage, est donc une priorité pour maintenir le parc automobile dans un état satisfaisant.

Les véhicules subissent des entretiens types suivant le nombre de kilomètres effectués depuis le dernier entretien. Ces entretiens types ont lieu pendant toute la période de vie du véhicule à des intervalles de kilomètres particuliers, par exemple tous les 5 000 km, 10 000 km, 50 000 km, avec une certaine marge de tolérance. Ainsi, un véhicule peut subir l’entretien type des 50 000 km entre 47 000 km et 53 000 km puisque la tolérance est de 3 000 km pour ce type d’entretien. Plusieurs entretiens peuvent être réalisés à une même date pour le même véhicule : ainsi l’entretien type effectué tous les 50 000 km implique aussi l’entretien type effectué tous les 10 000 km.

Une application développée, implémentée et exploitée localement dans l’agence de Bordeaux doit permettre le suivi des entretiens. Un langage de programmation objet est utilisé. Un paquetage de classes techniques permet d’accéder à la base de données. Cet accès aux données ne fait pas partie de l’étude.

Un diagramme des classes partiel, réalisé pour l’application, est présenté en **annexe a** et la description des classes implémentées est fournie dans l’**annexe b**.

Afin de réaliser les associations entre les classes, une classe technique Collection est utilisée ; celle-ci est présentée en **annexe c**.

|  |  |
| --- | --- |
| **TRAVAIL À FAIRE** | |
| **4.1** | Reprendre les éléments utiles du diagramme de classes de l’**annexe a** pour faire figurer dans le nouveau diagramme obtenu la classe Entretien dont l’implémentation est présentée en **annexe b**. |

# Classe Entretien

|  |  |
| --- | --- |
| **TRAVAIL À FAIRE** | |
| **4.2** | Écrire le constructeur de la classe Entretien. |

# Classe Véhicule

|  |  |
| --- | --- |
| **TRAVAIL À FAIRE** | |
| **4.3** | Écrire l’algorithme de la procédure **ajouteEntretien de la classe Véhicule** qui permet d’ajouter un entretien à un véhicule. |

# Classe Atelier

On désire savoir si un véhicule doit faire l’objet d’un entretien type. Ce sera le cas si le kilométrage effectué depuis le dernier entretien de ce véhicule est compris dans l’un des intervalles de tolérance correspondant aux entretiens types de la classe Atelier.

|  |  |
| --- | --- |
| **TRAVAIL À FAIRE** | |
| **4.4** | Écrire l’algorithme de la fonction **nécessiteEntretien** de la classe Atelier qui indique si un véhicule doit faire l’objet d’un nouvel entretien. |
| **4.5** | En utilisant la méthode précédente, écrire la fonction **véhiculesAEntretenir** qui retourne tous les véhicules devant faire l’objet d’un entretien. |

Annexe a – Diagramme de classes partiel à compléter



**Commentaires** :

* La classe EntretienType permet de recenser tous les types d’entretiens et ceci quels que soient les véhicules.
* Les rôles indiqués sur les associations (*lesTypesExistants*, *lesVéhicules*) se traduiront par des attributs dans les classes concernées par ces rôles.

Annexe b – Les classes implémentées (seuls les attributs et méthodes utiles sont représentés)

|  |  |
| --- | --- |
| **public** **class** Atelier{  **private** Collection<Véhicule> lesVéhicules =  **new** Collection<Véhicule>() ;  **private** Collection<EntretienType> lesTypesExistants  =  **new** Collection< EntretienType >() ;  // une fonction indique si le véhicule passé en paramètre doit  // bénéficier d’un entretien  **public** **boolean** nécessiteEntretien(Vehicule  unVehicule)  {  **boolean** resultat ;  …  return resultat ;  }  //Une fonction qui retourne tous les véhicules  //devant bénéficier d’un entretien  **public** Collection<Vehicule> vehiculesAEntretenir()  {  Collection<Vehicule> resultat = **new** Collection<Véhicule>();  …  **return** resultat ;  }  }//finClasse | public classe EntretienType{  **private int** code;  **private int** nbKm;  **private int** nbKmToléré ;  …  // retourne le kilométrage standard du // type d’entretien courant  **public int** getNbKm() {  **return** nbKm;  }  // retourne la marge de tolérance en km // majorant ou minorant le kilométrage // standard  **public int** getNbKmToléré(){  **return** nbKmToléré;  }  **}//finClasse** |
| public classe Vehicule{  **//Attributs**  **private** String numImma ;  **private** **int** nbKmActuel ;  **private** Collection <Entrtien> mesEntretiens = **new** Collection<Entretien> ;  // retourne le kilométrage du véhicule courant  **public int** getNbKmActuel() **{**  **return** nbKmActuel;  **}**  // retourne l’entretien dont le rang est égal à l’index  **public** Entretien getUnEntretien(**int** index){  **retrun** mesEntretiens.donnerObjet(index) ;  }  // retourne le nombre total d’entretiens effectués sur le  // véhicule courant  **public** **int** getNbEntretiens(){  **return** mesEntretiens.cardinal() ;  **}**  // ajoute un entretien au véhicule courant  **public void ajouteEntretien**(Date uneDate, String unCommentaire, EntretienType unType){  **…**  }  **}//finClasse** | public classe Entretien{  // Cette classe recense tous les entretiens réellement effectués, plusieurs entretiens peuvent avoir lieu à la même date. L’attribut nbKmCompteur correspond au kilométrage du véhicule au moment de l’entretien.  **private** date dateEntretien ;  **private** **int** nbKmCompteur ;  **private** Sting commentaire ;  **private** EntretienType leType ;  **// Constructeur**  **public Entretien**(Date uneDate ,  String unCommentaire, **int** unnNbKm,  EntretienType unType){  // valorise les attributs privés  …  }  **public int** getNbKmCompteur(){  // retourne le kilométrage du véhicule // pour l’entretien courant  **return** nbKmCompteur ;  }  **}//finClasse** |

Remarque : Tous les arguments des méthodes sont en « entrée ».

Annexe c***: Classe Collection***

|  |
| --- |
| **publicclasse Collection {**  // classe générique : un objet de la classe Collection permet de  // gérer un ensemble d'objets de même classe.  **public int cardinal(){**  // renvoie le nombre d'éléments de la collection  }  **public Object donnerObjet(int** index**) {**  // retourne l’objet qui se trouve à l’index passé en paramètre }  **public void ajouter(Object unObjet){**  // ajoute un objet à la collection  }  } |
| Pour instancier une collection :  **Collection<Classe>** uneCollection **= new Collection de <Classe>() ;**  // La collection instanciée contiendra des objets de la classe <Classe>  Les méthodes présentées sont toutes publiques.  L'avantage de cette classe est de proposer des services d'ajout et de suppression plus simples que la gestion d'un tableau. En outre, le problème du dimensionnement de la structure n'est pas à la charge du développeur. |