BE – Programmation Orientée-Objets

Développement d'un Gestionnaire de Véhicules Automobiles réutilisable et extensible

1. *Introduction et motivations*

L'objectif pédagogique du BE est de se familiariser avec les concepts objet, de l'analyse des spécifications d'un problème à la réalisation, y compris en réutilisant des classes existantes.

Placez vous d'abord en tant que concepteur, le résultat tenant plus à l'organisation conceptuelle qu'à la quantité de lignes de code ! Il n'y a donc aucune complexité algorithmique dans le travail proposé.

Les fonctions de calcul de performance proposées sont totalement « fantaisistes », juste là pour donner vie à votre programme. C'est architecture de votre système qui compte, en matière de réponse aux spécifications, en termes d'évolutivité et de stabilité aux modifications.

IMPORTANT : Approche de développement par étapes

a) Analyse du sujet, identification des objets, des classes et de leurs liens

b) Modélisation du système avec des diagrammes de classes au format UML (voir annexe 4)

c) Réalisation et validation (test unitaires et test d'intégration, test fonctionnel du système)

La première étape (a) doit être préparée avant le BE. C'est une appropriation du sujet.
La seconde étape (b) doit être terminée en milieu de première séance de BE.

2. *Objectifs techniques*

L'objectif du travail est de concevoir et de mettre en œuvre un **Gestionnaire de Véhicules Automobiles** (**CMS** - *Car Management System*) fournissant deux services :

- un Configurateur de Modèles (**`Configurator`**).
- un gestionnaire de Base de Données (**`Database`**).

L'ensemble des commandes sera stocké en sortie du *Configurator* dans la base de données *Database*.

La base de données des voitures en commande correspondra à une liste générique d'objets (différentes voitures). On pourra visualiser selon un critère (type, moteur, énergie) des sous-listes de véhicules.

Les services **`Configurator`**, **`Database`** sont les constituants du **`CMS`** et doivent donc être réalisés, en respectant les spécifications qui sont un contrat avec le client.

1) un configurateur de modèles (***Configurator***) permettant à un *utilisateur* de :
   - **a)** **`configurer`** un véhicule selon les caractéristiques de son choix ;
   - **b)** **`afficher`** le véhicule choisi de façon détaillée et d'en calculer le prix ;
2) un gestionnaire de base de données (***DataBase***) permettant à un *utilisateur* de :
   - **a)** **`stocker`** des véhicules créés avec un configurateur ;
   - **b)** **`lister`** les véhicules enregistrés dans la base ;
   - **c)** **`filtrer`** les véhicules en commande correspondant au critère choisi.

3. *Spécifications*

L'application **CMS** *configure* des *voitures*, les stocke dans une *base de données* et fournit un analyseur par critère.

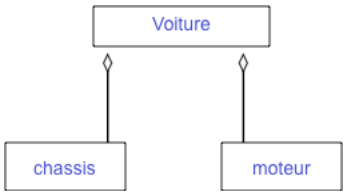
- Un **programme principal** du **CMS** est donné à titre d'exemple en annexe 1.
- Le **processus de développement** se compose d'étapes qui sont rappelé en annexe 2.
- Une **trace d'exécution du programme** est donnée à titre d'exemple, en annexe 3.

3.1. Définition des modèles

Deux types de **modèles** sont proposés : une **berline** et un **coupé**. Pour chacun de ces **châssis**, il existe différents équipements optionnels (**standard** et **luxe**). Une berline peut avoir une spécialisation en version **4x4** ou en version **break**.

Ces deux types de voitures peuvent être équipés de différents types de **moteurs** thermiques (**essence**, **diesel**) ou bien d'un moteur **électrique** seul ou bien **hybride.**

Un modèle de **voiture** est donc composé d'un **châssis** (**berline** et dérivées ou **coupé**) et d'un **moteur** (**essence**, **diesel** ou **électrique** ou encore **hybride**) que l'on peut modéliser selon le schéma ci-contre.



Un moteur thermique peut avoir différentes cylindrées (**1800** et **2200**).

Il n'existe qu'un seul **moteur électrique** dont la puissance est de **70KW** soit **95 cv.** La cylindrée et la consommation de carburant sont nulles dans ce cas.

Un **moteur hybride** combine un **moteur électrique** et un **moteur essence**.

Une **voiture** choisie par un client correspond aux caractéristiques suivantes (Figure 1):

```
Chassis: berline et variantes / coupé
            Nb portes : 4 (berline) / 2 (coupé)
            Option : standard/luxe
            Dimensions : largeur (l), hauteur (h), longueur (L1), longueur toit (L2)
            Cx obtenu selon un calcul différent en fonction du modèle
Moteur : essence/diesel/électrique/hybride
      Moteurs thermiques :
            Cylindrée :   1800/2200 cc /* valeur nulle pour électrique */
            Puissance :   /* la puissance dépend du moteur (cylindrée, énergie) */
            Consommation : /* la consommation dépend de la puissance et du moteur E/D */
      Moteur électrique :
            La puissance du moteur électrique est de 70KW soit 95 cv.
Vitesse max (km/h):  /* la vitesse dépend de la puissance et du cx */
                     /* le cx dépend du type du véhicule – berline ou coupé */
Prix : /* le prix est calculé en fonction du modèle choisi */
```

Figure 1. Résumé des caractéristiques d'une voiture

3.2. Caractéristiques techniques

Les performances (**puissance, vitesse maximum**) d'un modèle se calculent en fonction de paramètres moteur (**cylindrée, énergie**) et de caractéristiques aérodynamique (**cx**).

- Calcul de la puissance moteur : **P = K * cylindrée** *(juste pour l'exemple dans le BE)*
  - **Moteur Essence (CV): K = 0,07**
  - **Moteur Diesel (CV): K = 0,06**
- Calcul de la consommation moteur : **Soit la constante Q = 0,055**
  - **Moteur Essence (CV): C = Q * P** *(pour l'exemple dans le BE)*
  - **Moteur Diesel (CV): C = Q*P / log10 (P/6)** *(pour l'exemple dans le BE)*
- Calcul de la vitesse max : **Vmax = 2 * puissance * (1 – cx)** *(pour l'exemple dans le BE)*
- Calcul du coefficient de caractéristiques aérodynamique, cx : *(pour l'exemple dans le BE)*
  - **Berline :** **cx = R * ((L1 - h/2 + L2)/2 * l * h + (h/2)²)**
  - **Berline-break :** **cx = R * (L1 + L2)/2 * l * h**
  - **Berline-4x4 :** **cx = R * L1 * l * h**
  - **Coupé :** **cx = R * l * L1/2 * h**

Dans tous les calculs de cx, la valeur de la constante R est 0,03.

Attention :

a) *la formule de calcul de la puissance change d'un type de moteur à l'autre.*

b) *la formule de calcul du cx dépend du chassis (berline, 4x4, break ou coupé).*

c) *la formule de calcul de la vitesse dépend du cx du chassis et de la puissance moteur*

Il faut donc tenir compte des formules différentes pour concevoir une solution facilement réutilisable et extensible, c'est-à-dire en utilisant l'héritage et le polymorphisme.

Dans le BE, nous fixons les dimensions pour les différents modèles :

	L1	*L2*	*h*	*l*
Berline	4,6	2	1,4	2,2
Berline-4x4	4,6		1,3	2,5
Berline-break	4,6	3,2	1,4	2,2
Coupé	4,6		1,3	2,2

3.3. Calcul des prix

Le prix de base est fixé à une valeur constante de **25000 €** pour une configuration par défaut **berline-essence-1800-équipement_standard,** mais ajustable en fonction des options (voir tableau des prix - Figure 2).

Caractéristiques		**Prix additionnel**
Châssis	Coupé	+ 2000 €
	Berline-break	+ 1000 €
	Berline-4x4	+ 3000 €
Equipement luxe		+ 1500€
Moteur	Diesel	+ 2000 €
	Electrique/Hybride	+ 3000 €
Cylindrée 2200		+ 1000 €

Le prix des modèles s'inscrit donc dans l'intervalle **25000** € **- 32500** €

- le moins cher : <**berline, essence 1800** > soit **25000** €
- le plus cher : <**4x4, diesel 2200 ou hybride, luxe**> soit **32500** €

Figure 2. Prix des caractéristiques additionnelles et des véhicules

3.4. Définition du *Configurator*

Le **Configurator** permettra de :

- **Configurer** une voiture choisie par un utilisateur (***sélection interactive de tous les paramètres***).
- **Afficher** toutes les caractéristiques du modèle sélectionné par le client et son prix.

La sortie de la méthode **Configurer** du **Configurator**, qui inclue l'interaction homme-machine, est donc une **voiture** parfaitement définie, dont toutes les caractéristiques peuvent être affichées par la fonction **Afficher**. La **DataBase** stockera les voitures configurées par le **Configurator**.

3.5. Dialogue opérateur

Ce programme est interactif et donc un dialogue operateur sera nécessaire pour inviter l'utilisateur à faire des choix de modèle et d'options.

Ce dialogue opérateur sera effectué en C++ avec les opérateurs de base **cout / cin**. Nous ne chercherons pas dans ce projet à optimiser cet interface utilisateur.

Cependant, il faudra isoler ce dialogue dans une méthode qui pourra être optimisée ultérieurement.

C'est la méthode **Configurer** du **Configurateur** qui hébergera l'implémentation du dialogue opérateur dans votre programme final.

3.6. Définition de la base de données : gestion des listes

Vous réaliserez la **DataBase** à l'aide de classes de gestion de listes d'objets qui sont fournies.

Une liste est composée de cellules, contenant une **information** et un **pointeur** sur la cellule suivante.

La liste fournie permet de stocker dynamiquement n'importe quel type d'objet en modifiant une déclaration, c'est-à-dire **l'élément d'information** stocké dans une cellule de la liste. Dans l'exemple qui est donné cet élément est un **entier**.

Un programme d'utilisation de la liste est aussi donné. Il permet de voir comment **insérer** une nouvelle cellule dans la liste, lister les éléments qui sont dans les cellules de la liste.

http://homepages.laas.fr/fabre/POO/

En s'appuyant sur cet exemple, la classe **DataBase** fournit les services suivants :

- **Stocker** les véhicules choisis en sortie du configurateur
- **Lister** les véhicules enregistrés dans la base
- **filtrer** les véhicules en commande correspondant au critère choisi

Un critère de sélection est un modèle (**berline**, **coupé**..) ou un moteur (**essence**, **diesel**, **électrique**..).

Travail à réaliser

Le travail à réaliser se déroulera en deux phases :

- **Phase 1** : Conception et Modélisation de la solution (en UML)
- **Phase 2** : Réalisation et Validation du Système réalisé

A la fin de la **Phase 1**, une représentation graphique de la conception du système devra être produite et discutée avec les enseignants. Ce document est impératif et servira de base à la **Phase 2**.

1. *Conception : Quels sont les objets, les classes, le fonctionnement*

- Déterminer les objets et concevoir les **classes** correspondantes (sous forme de fichiers **.h**) ;
    - Quels sont les objets utiles pour réaliser le système ?
    - Quels sont leurs attributs, leurs méthodes ?

- Concevoir les différentes **hiérarchies conceptuelles** pour optimiser le développement ;
    - Définir les liens entre les objets (composition, association/utilisation) ;
    - Définir les relations entre classes (héritage, dérivation) ;

- Synthétiser la conception (**graphe de classes**) sous forme graphique ;

2. *Réalisation*

2.1. Implémentation des classes et utilisation des objets

- Développer en C++ les classes (sous forme de fichiers **.cpp**)

- Concevoir la **solution au problème** posé en intégrant ces objets : le programme principal, le main.
    - Service de sélection et de configuration de modèles ;
    - Service d'enregistrement et d'analyse de modèles ;

2.2. Validation: Tester les composants et le programme complet

- Test unitaire des classes : instanciation des classes et leurs composants, calculs des caractéristiques propre à chaque option…
- Test d'intégration dans un programme.

3. *Rapport : Dossier de développement*

Un rapport sera demandé quelques jours après la fin du BE.

Le plan sera donné ultérieurement et couvrira les points suivants : Conception, Implémentation, Validation et Extensions de la solution.

ANNEXE 1: Programme principal du CMS

Votre programme principal ressemblera à ceci :

```
main()
        configurator CF();      // static creation of the Configurator
        dataBase DB();          // static creation of the database
        car* vp;                // local reference (a pointer) for a car

        // car selection and storage
    while (true) {
        vp=CF.config();         // car model selection
        CF.display(vp);         // display of the selected car
        DB.insert(vp);          // insert this car into the database
        cout <<"Continue? " ; cin >> rep; if (rep !='y') break;     // do you want to continue?
    }
        // car list full display
        DB.list();      // display of the full list of selected car stored into the database

        // car list analysis
    database* temp_DB ; // pointer to a database

    DB.select(); // 1: berline; 2: 4x4; 3: break; 4: coupe; 5: HDI; 6: ESS; 7 : ELEC

    temp_DB=new(database);          // temporary database creation
    DB.filter(DB, temp_DB) ; // car sublist according to selected criteria, loading temp_DB
    temp_DB->list();  // sublist display
    // the temporary database can be selected as an input database for multiple criteria selection
    }
```

IMPORTANT: Ceci est un exemple donnant le **niveau d'abstraction** du programme principal attendu. La solution au problème utilise des objets complexes qui utilisent des objets plus élémentaires. La solution, son exécution, peut donc se lire de façon macroscopique.

Bien entendu, ceci reste un exemple qui peut servir de guide de réflexion et de mise en œuvre.

En fonction de vos choix conceptuels et d'implémentation, des **différences sensibles peuvent apparaître** dans votre solution, et c'est normal.

Le programme n'est donc pas imposé…. C'est la méthode de développement qui l'est !

Comme vous le voyez sur cet exemple, le programme final doit être compact et facilement lisible car bien structuré grâce à l'approche objet.

ANNEXE 2: Processus de développement

- Quels sont les objets ? C'est-à-dire ceux dont on aura besoin dans le système/programme.
- Quelle est la fonctionnalité finale, l'algorithme du main, la logique des actions ?
- Quelles sont les classes associées ?

  - –Les attributs, les composants (les attributs peuvent être des objets) ;
  - –Les attributs, les associations (références sur d'autres objets) ;
  - –Les méthodes.

- Peut-on factoriser les classes ? Hiérarchies conceptuelles et héritage.

  - –Les attributs ;
  - –Les méthodes ;
  - –Des questions à se poser :

    - Où les attributs doivent-ils être définis ?
    - Où doivent-ils être initialisés ?
    - Où les méthodes doivent-elles être définies ?
    - Où doivent-elles être implémentées ?
    - Doivent-elles être re-définies ? Où ?

- Les classes abstraites permettent de profiter du polymorphisme.
- La mise en œuvre :

  - –Réalisation des classes : procéder de façon incrémentale, faire une 1ère version des classes et les compléter ;
  - –Tester les classes de façon unitaire, puis le système par parties :

    - –Test unitaire : chaque classe est testée indépendamment en créant des objets ;
    - –Test d'intégration : composition, agrégation de classes = parties du système.

- Réalisation du programme complet à l'aide des classes et des sous-systèmes validés.

Livraison…. Et APRES

- Maintenance : correction de défauts, optimisation d'implémentation.
- Réutilisation :

  - –Directes : des classes utiles à d'autres applications ;
  - –Indirectes : des classes dérivées à partir de classes existantes.

- Evolution : Ajout de fonctionnalités, le design doit l'anticiper…. Autant que faire se peut !

ANNEXE 3 : Exemple d'exécution du CMS

(Exemples de traces d'exécution d'une implémentation simplifiée du CMS)

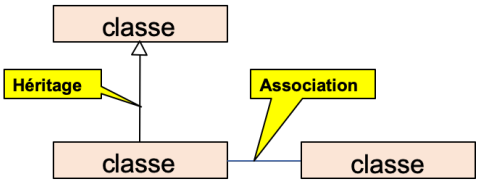
Remarque : *les performances ne sont données ici qu'à titre d'exemple et les calculs ne correspondent pas tout à fait à ceux demandés dans le BE.*

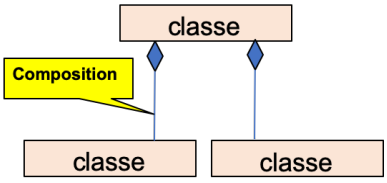
```
STARTING THE CAR MANAGEMENT SYSTEM
DATABASE CREATED
CONFIGURATOR CREATED


*** Debut sélection des modèles ***
Selection d'un modele********** (Y/N) : Y
Type? (1-BL ; 2-4x4 ; 3-Break ; 4-CC ) :         1
Voiture! Vous avez choisi une Berline!
Modele selectionne' :
Option? (1-STRD : 2-LUXE)          2
Energie? (1-HDI ; 2-ESS ; 3-ELC)   1
Cylindree? (1-C1800 2-C2200)       2


BL 4 P LUXE C2200 HDI 132 CV 188.16 km/h  * 19500 Euros
```

```
*** Debut Analyse des commandes ***
Liste des modeles en commande**********
BL 4 P STRD C1800 HDI 108 CV 153.95 km/h  27000 €
CC 2 P LUXE C2200 ESS 154 CV 247.20 km/h  29500 €
CC 2 P LUXE C2200 HDI 132 CV 211.90 km/h  31500 €
BL 4 P LUXE C1800 HDI 108 CV 153.95 km/h  28500 €
BL 4 P STRD ***** ELC  95 CV 135.42 km/h  27000 €

CHOIX CRITERE: essence - 1 ; diesel - 2 ; LUXE - 3 ; FIN - F: 2
Liste des modeles diesel en commande**********
BL 4 P STRD C1800 HDI 108 CV 153.95 km/h
CC 2 P LUXE C2200 HDI 132 CV 211.90 km/h
BL 4 P LUXE C1800 HDI 108 CV 153.95 km/h

CAR MANAGEMENT SYSTEM: BYE, BYE...
```

ANNEXE 4 : Rappel de quelques notations UML

Voir chapitre 5 du cours pour plus d'information.





Héritage : relation de spécialisation/généralisation entre classes

Composition : relation d'inclusion d'objets dans d'autres objets

Association : relation d'utilisation d'un objet indépendant

Fig. 1. Architecture logicielle – Diagramme de classes

Fig. 2. Descriptif des classes

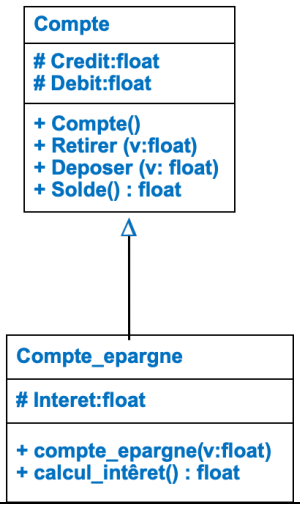


Fig. 3. Prototype (.h) des classes en C++

```
class compte {
protected :
      float debit,credit;

public :
            compte();
      void deposer(float);
      void retirer(float);
      float solde();
};

class compte_epargne:public compte {
protected :
      float interet;

public :
            compte-epargne(float);
      float calcul_interet();
};
```

Fig. 4. Modélisation de la visibilité

- \- private
- \# protected
- \+ public

Fig. 5. Codage (.cpp) des classes en C++

```
#include "compte.h"

compte::compte(){
      credit=0;
      debit=0;
}
void compte::deposer(float v){
      credit+=v; cout << "Depot de : " << v << "\n" ;
      cout << "Somme credits : " << credit << "\n" ;
 };
```

Implémentation de toutes les méthodes de la classe compte (compte.h) et de toutes ses classes dérivées dans le fichier compte.cpp