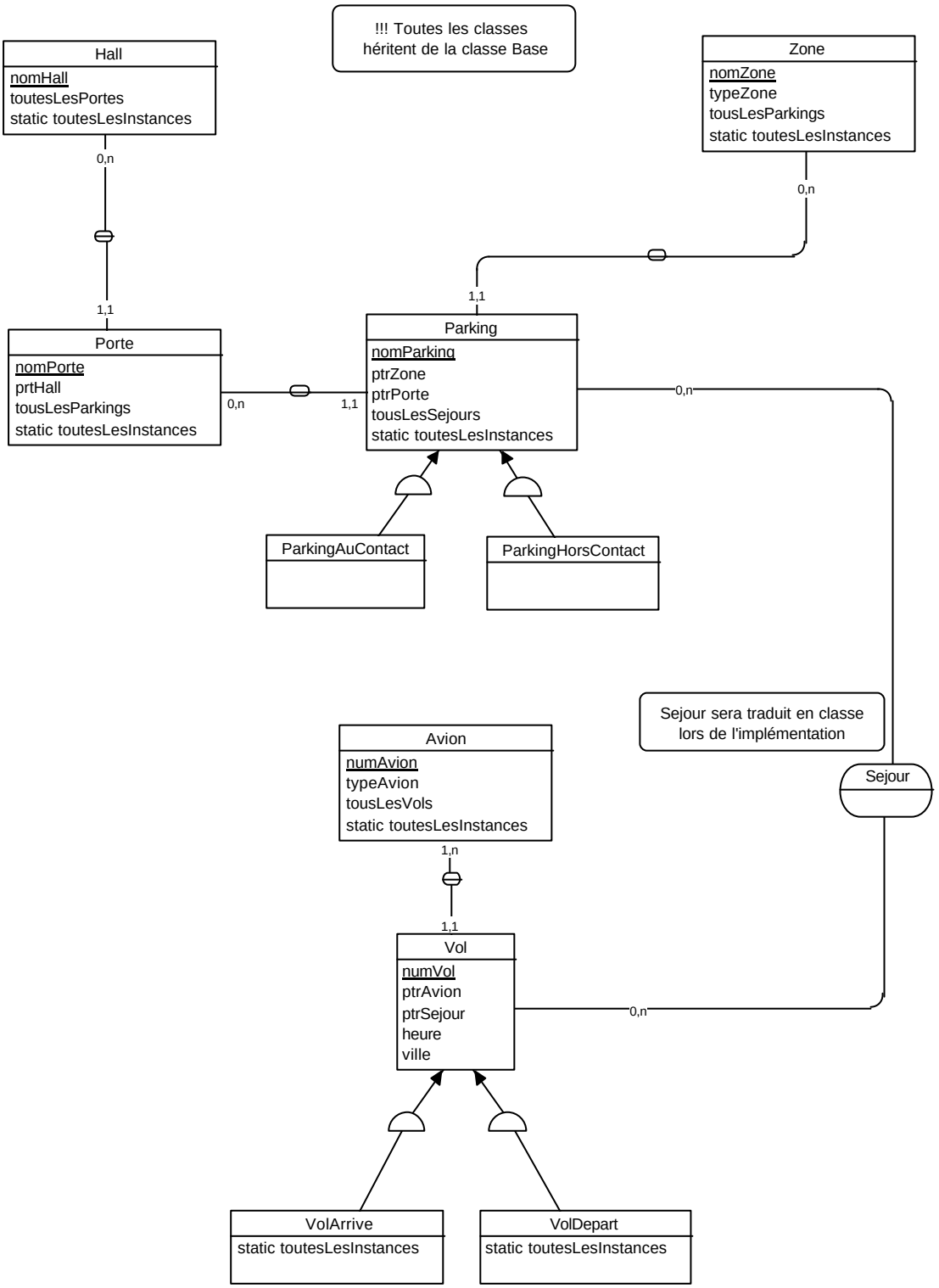
IUP MIAGe - 1999 / 2000

Maya DARRIEULAT - BaBa NGOM - David ROUSSE

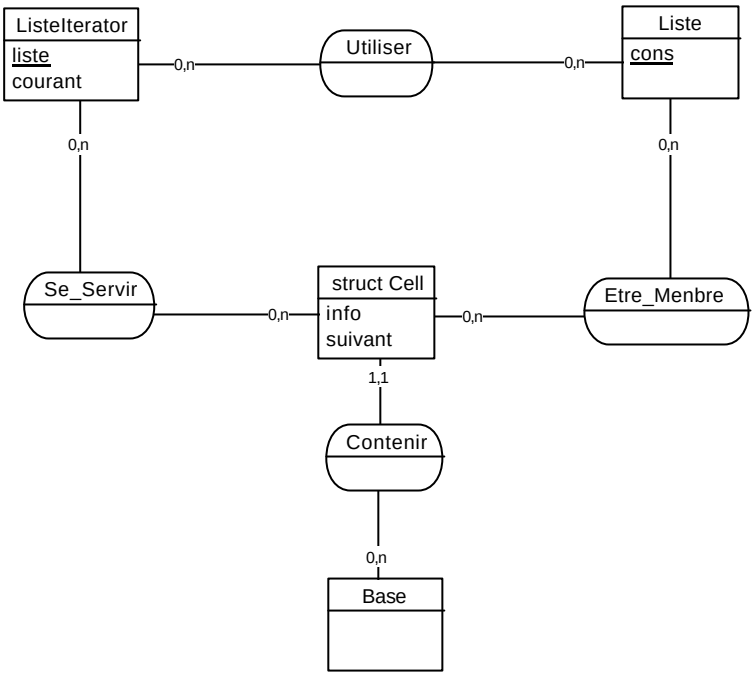
ANALYSE ET CONCEPTION

Avant de rentrer dans le vif du sujet, " la programmation ", nous avons passer du temps (10 heures environ) à comprendre et analyser le projet. Nous avons donc établit dans un premier temps, une ébauche du modèle conceptuel de données, puis réfléchit aux spécifications relative aux classes.

Au final, nous sommes arrivés au Modèle Conceptuel de Données suivant (réalisé avec AMC Designor) :

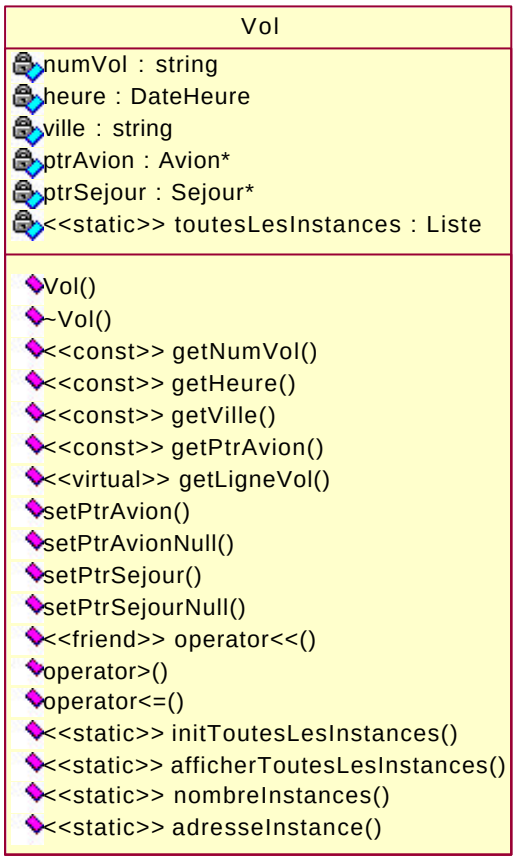


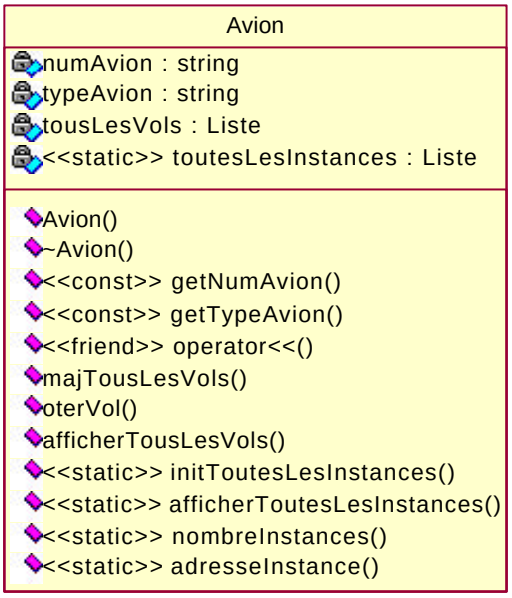
Modèle conceptuel de données		
Projet : PCC		
Modèle : Gestion des parking		
Auteur : Ma-Ba-Da	Version: 2.00	21/03/100

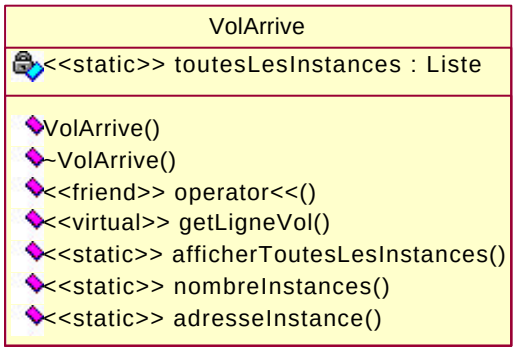


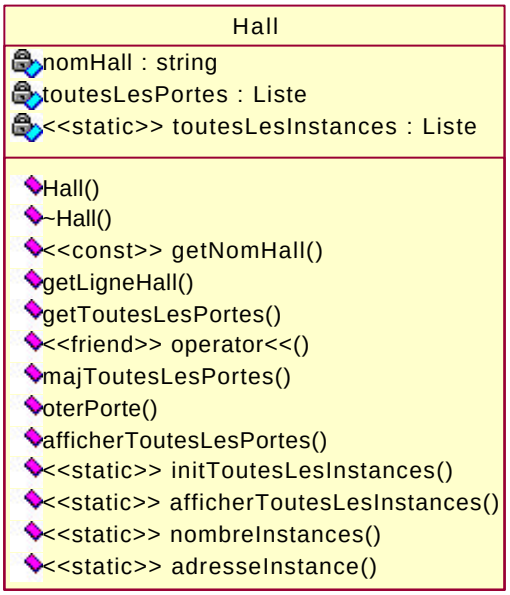
Modèle conceptuel de données		
Projet : PCC		
Modèle : Gestion des parking		
Auteur : Ma-Ba-Da	Version: 2.00	21/03/100

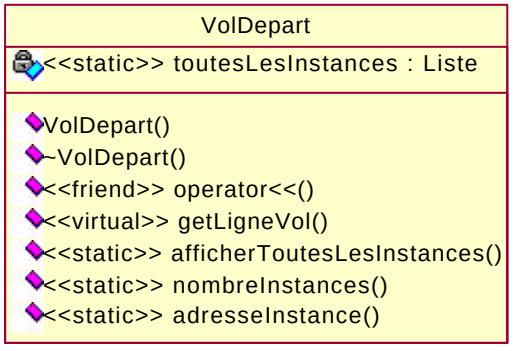
Voici les spécifications des classes (réalisées avec Rational Rose) :

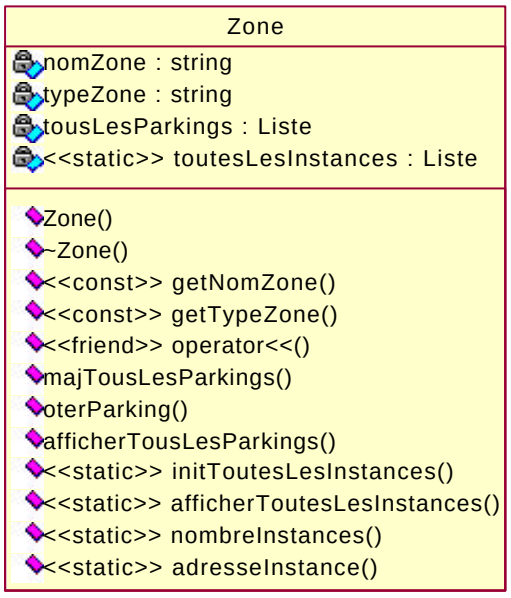


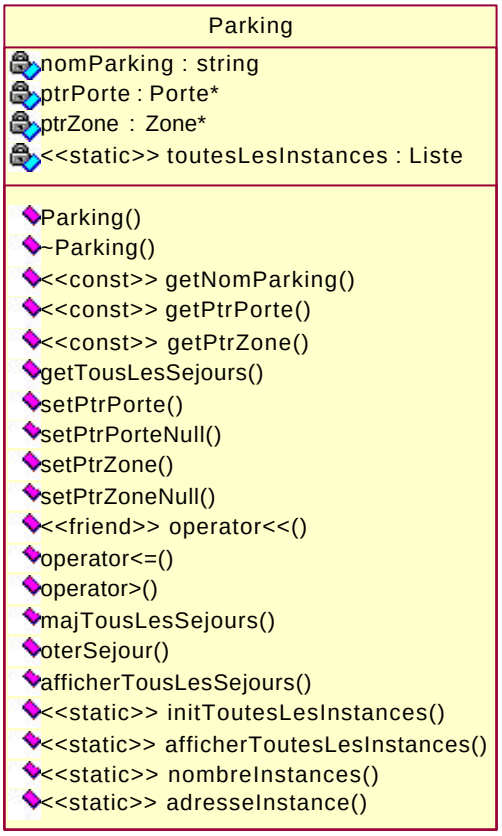


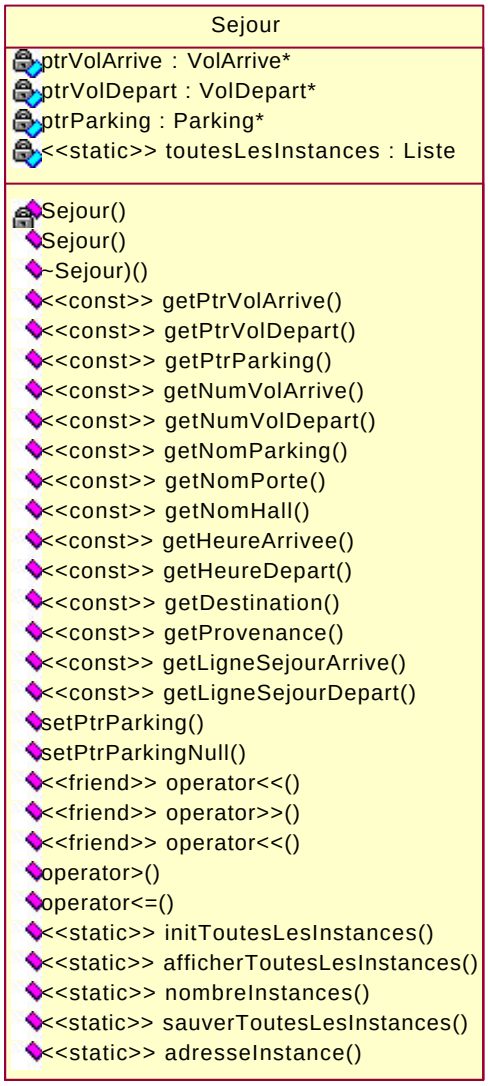




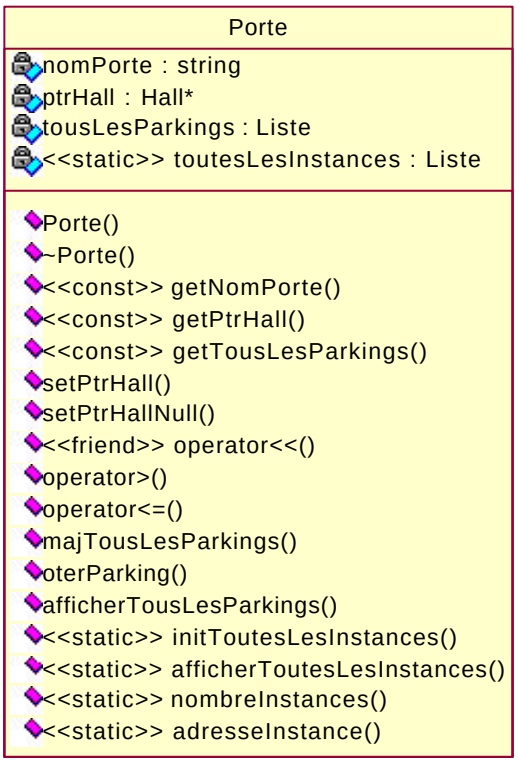








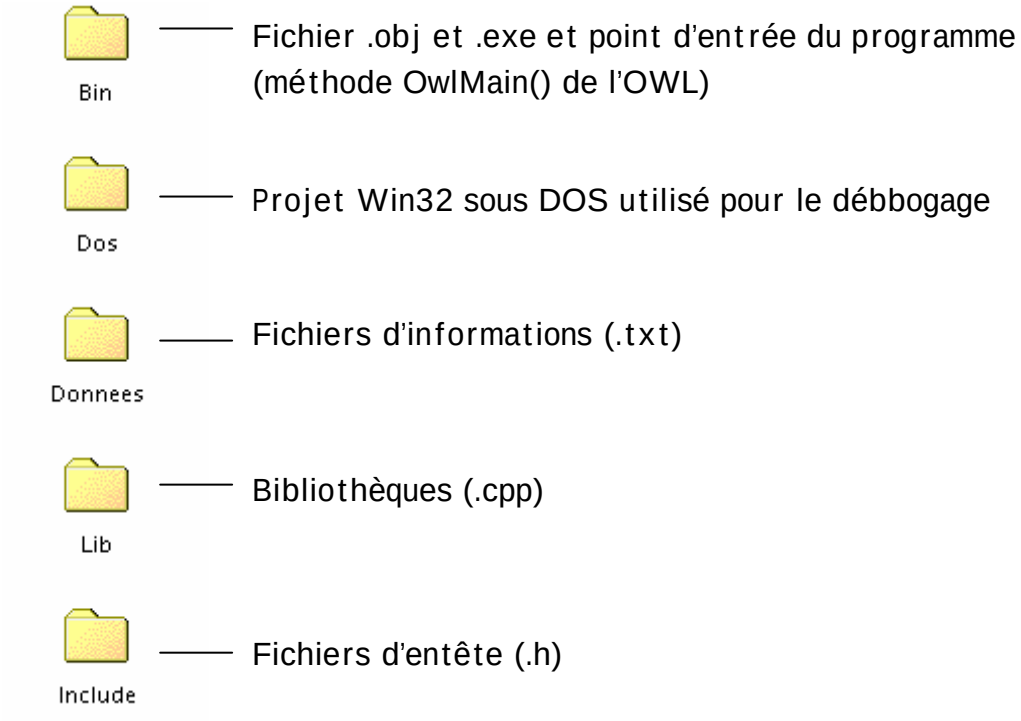
N.b. : bien qu'elles ne soient pas présentées ici, les classes ParkingContact et ParkingHorsContact ont été réalisées.



DEVELOPPEMENT

Suivi des conventions de programmation :

Le programme est organisé selon l'arborescence suivante :



En ce qui concerne les noms utilisés, nous avons toujours imposé le même schéma d'écriture dont voici un exemple :

- Nom de classe : Hall
- Nom de variable : nomHall
- Nom de méthode : getNomHall()

Algorithme de l'affectation des parkings :

On affecte les parkings et on crée les vols lors de l'initialisation des séjours.

```
DEBUT
    Initialisation Hall, Porte, Zone, Parking, Avion

    int nbEnregistrement;
    ouvrir le fichier Séjours.txt ;
```

```
Si Fichier Sejours.txt est bien ouvert Alors
        nbEnregistrement←nombre d'enregistrements qui est inscrit en
        première ligne du fichier ;
        Pour i de 1 à nbEnregistrement avec un pas de 2 Faire
                Création d'un séjour vide ;
                Séjour vide ← infos présentes dans Séjours.txt ;
                //en même temps sont crées les VolArrive et VolDepart
                //correspondants
        Fin pour ;
       //les Séjours sont maintenant crées et classés par heure de
       //VolArrive
       fermeture fichier Séjours.txt ;
Fin si ;

//on crée un liste de Sejour calquée sur toutesLesInstances
Liste tousLesSejoursDejaAlloues ;
TousLesSejoursDejaAlloues ←  toutesLesInstances;

//variables pour parcourir les donnees
ListeIterator parkingIterator(&Parking::toutesLesInstances);
ListeIterator sejourIterator(&toutesLesInstances);
ListeIterator sejourIteratorDejaAlloue(&tousLesSejoursDejaAlloues);

Sejour * ptrSejourDejaAlloue;
Sejour * ptrSejourAallouer ←  (Sejour*)sejourIterator++;;
Parking * ptrParkingVide ←  (Parking*)parkingIterator++;

int nbAllocation;
Duree delai(0,5); //on prévoit un délai de battement de 5 minutes
                                  //entre chaque vol

//nombre d'allocation de Parking à effectuer
nbAllocation ←  Sejour::nombreInstances();

//création du premier sejour
ptrSejourAallouer->setPtrParking(ptrParkingVide);

//mise à jour des parkings utilisés
ptrParkingVide ←  (Parking*)parkingIterator++;
```

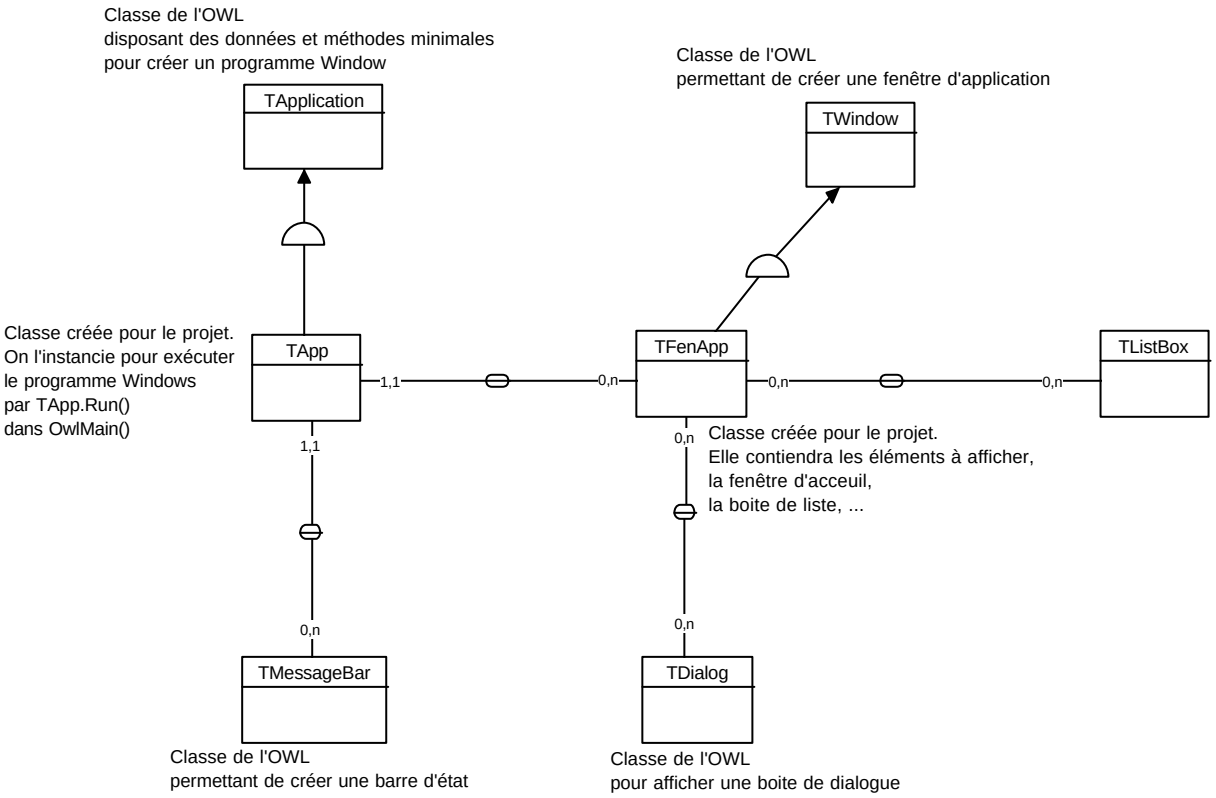
```
//mise à jour de nombre d'allocation restants à faire
nbAllocation--;

//allocation des nbEnregistrement-1 Sejours restants
Pour i de 1 à nbAllocation Faire
    //mise à jour des Sejour restants à allouer
    ptrSejourAallouer ← (Sejour*)sejourIterator++;

    //retour en début de liste de Sejour deja alloués
    sejourIteratorDejaAlloue.reinit();

    //on se place sur le premier Sejour à prendre en compte
    ptrSejourDejaAlloue ← (Sejour*)sejourIteratorDejaAlloue++;

    //parcours des Séjours déjà alloués en recherchant un Sejour
    //dont l'heure de Depart soit < = à l'heure d'arrivée du Sejour à
    //allouer
    Tant que le pointeur de séjour n'est pas NULL
    Et que l'heure de départ du séjour pointé + le délai est > à l'heure
    d'arrivée du séjour à allouer Faire
        ptrSejourDejaAlloue ←(Sejour*)sejourIteratorDejaAlloue++;

        Si ptrSejourDejaAlloue n'est pas NULL Alors
            on a trouvé un Sejour dont l'heure de Depart est
            <= à l'heure d'arrivée du Sejour à allouer
            on affecte donc un sejour à un parking deja utilise

            //on met à jour l'indice des sejours déja alloués
            tousLesSejoursDejaAlloues.oter(ptrSejourDejaAlloue);
        Sinon
            //on affecte un parking non alloue
            ptrSejourAallouer->setPtrParking(ptrParkingVide);

            //mise à jour des parkings utilisés
            ptrParkingVide = (Parking*)parkingIterator++;
        Fin si;
    Fin tant que;
Fin pour ;
FIN.
```

Pour vérifier notre affectation des parkings, nous avons crée un méthode permettant d'écrire automatiquement dans un fichier la liste des affectations par séjour.
En appelant Sejour::sauverToutesLesInstances(); on génère ce fichier (voir annexe).

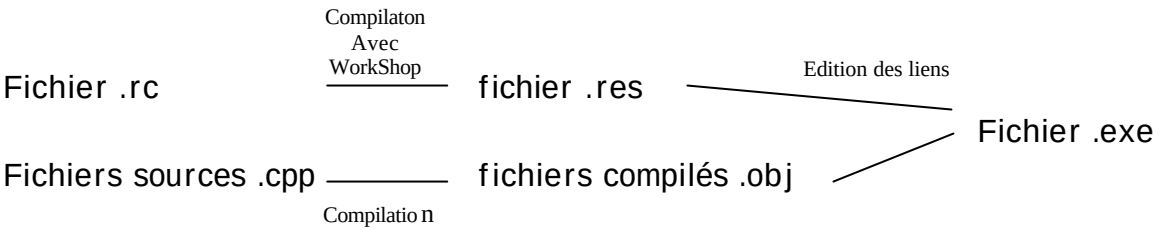
Code Source : cf Listing

INTERFACE GRAPHIQUE

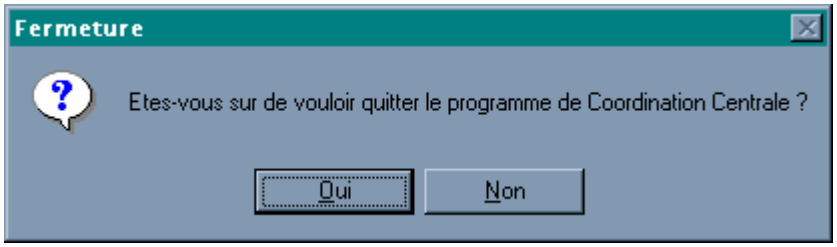
L'OWL (Object Windows Library) est une bibliothèque de classes (équivalente aux MFC de Microsoft) fournie par Borland pour faciliter la création d'un logiciel pour MS-Windows. Notamment l'ensemble des appels aux API WIN32 sont encapsulés au sein de classes de l'OWL facilitant ainsi l'accès aux fonctions avancées de l'OS tout en programmant en objet.
Pour créer une application Windows, les classes minimales à utiliser sont Tapplication (pour créer un programme Windows) et Twindow (pour créer la fenêtre du programme),
Dans le projet, voici les classes de l'OWL que nous avons utilisées :



Nous avons également utilisé dans notre projet un fichier de ressources. Les ressources Windows sont constitués de l'ensemble des éléments graphiques utilisés par le programme. On trouve dans un fichier de ressources des MENU, ICONE, BITMAP, DIALOG, … Le principe est de mettre en place l'équation Bloc de données + bloc de code = Programme. **Le bloc de données** contient toutes les ressources de chaîne de l'interface utilisateur mais aucun code. À l'inverse, **le bloc de code** ne contient que le code de l'application.



Pour créer notre fichier de ressources resgui.rc, nous avons eu recours au Worshop de Borland : c'est un éditeur de ressources qui permet de créer, modifier, visualiser des ressources Windows. Nous l'avons utilisé notamment pour créer la fenêtre d'accueil.


Fermeture
Etes-vous sur de vouloir quitter le programme de Coordination Centrale ?
Oui
Non

ANNEXES

On présente en annexes le fichier généré par notre application lors de l'affectation des parkings aux séjours puis le code source du programme.