[Etablissement]
[Adresse]
[CP] [Ville]

Pratique des Techniques Informatiques

BTS IG - Développeur d'Applications

Session 2004-2005

Candidat : **[nom – prénom]**

FICHE DE SYNTHESE n°2

« Réservations de voyages »

OBJECTIF DE L'ACTIVITE

- Implantation de la base de données,
- Création de classes à partir d'un diagramme de classe,
- Application des concepts d'héritage et de polymorphisme,
- Développer une application utilisant des collections d'objets,
- Chargement et sauvegarde des valeurs des collections dans une base de données.

SUPPORT DE L'ACTIVITE

Lieux de réalisation	Outils utilisés
▪ Lycée ***	▪ Delphi, ▪ Access ▪ ODBC

COMPETENCES CONCERNEES

C31 ☑ Gérer un projet de développement de logiciels.
C32 ☑ Développer à l'aide d'un langage de programmation procédural.
C33 ☑ Maquetter une application, la développer à l'aide d'un langage de programmation événementielle.
C34 ☑ Développer à l'aide d'un langage de programmation objet.
C35 ☐ Développer autour d'une base de données relationnelle.
C36 ☐ Développer dans le cadre d'une architecture client-serveur.
C37 ☑ Mettre au point et maintenir une application.
C38 ☐ Développer dans un environnement multimédia.
C39 ☑ Maîtriser le poste de développement et son environnement.

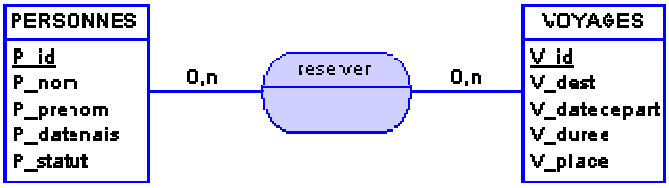
Présentation de l'activité

Cette application gère les réservations de place pour différents voyages. Il y a un programme qui est chargé d'enregistrer les personnes dans la base de données et un autre qui donne la possibilité à ces personnes enregistrées de réserver des places pour des voyages.

Des **collections d'objets** sont chargées au lancement de l'application à partir de la **base de données Access**. Elles sont mises à jour tout au long des opérations effectuées par l'utilisateur ce qui réduit les accès à la base de données pour gérer l'affichage des informations à l'écran.

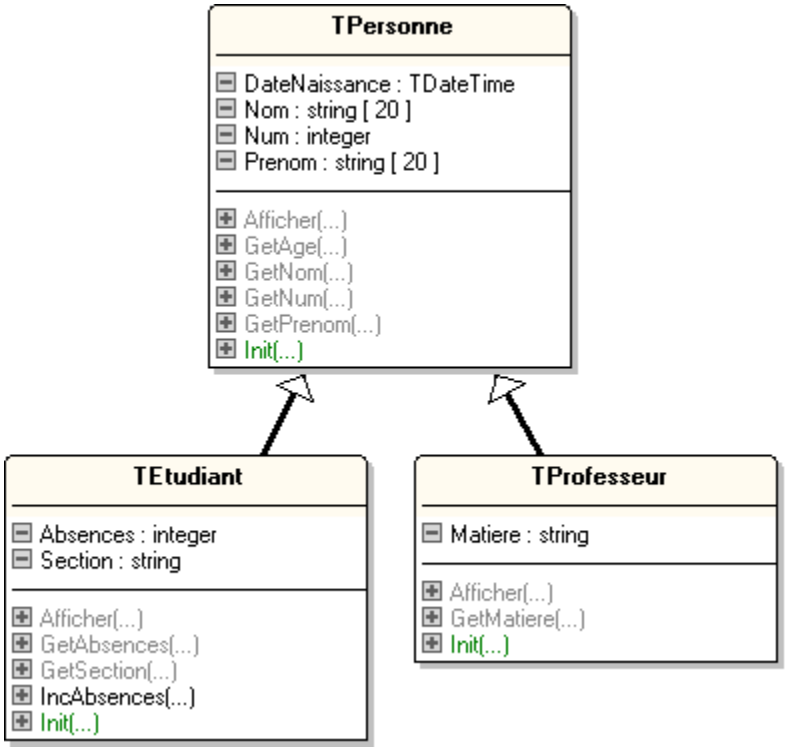
L'intérêt de cette activité réside donc dans l'utilisation des collections d'objets combinée à la programmation orientée objet pour exploiter une base de données.

Modèle Conceptuel des Données



Lorsqu'une personne réserve une place pour un voyage, les identifiants de la personne et du voyage sont enregistrés dans la table « reserver » .

Diagramme de classe



Le diagramme est composé de trois classes. **TPersonne** est une classe générique qui contient des informations communes aux deux sous-classes **TEtudiant** et **TProfesseur**.

Héritage et polymorphisme

TEtudiant **hérite des attributs et méthodes** de la classe mère et la spécialise avec l'ajout de nouveaux attributs et méthodes. TProfesseur est aussi une sous-classe et hérite de la même manière des attributs de TPersonne tout en ajoutant un nouvel attribut `Matière`.

Le mécanisme du polymorphisme permet à une sous-classe de **redéfinir une méthode** dont elle a hérité tout en gardant le même prototype de la méthode héritée. Ainsi, on remarque que chaque sous-classe possède une méthode `Init()`, héritée de TPersonne, qui permet d'initialiser les valeurs de leurs attributs.

Méthode `Init()` de la classe mère TPersonne.

```
   constructor TPersonne.Init(aNum: integer; aNom: string; aPrenom: string; aDate:
TDateTime);
   begin
        Num := aNum;
        Nom := aNom;
        Prenom := aPrenom;
        DateNaissance := aDate;
   end;
```

- Les paramètres passés au constructeur servent à valoriser les attributs de la classe.

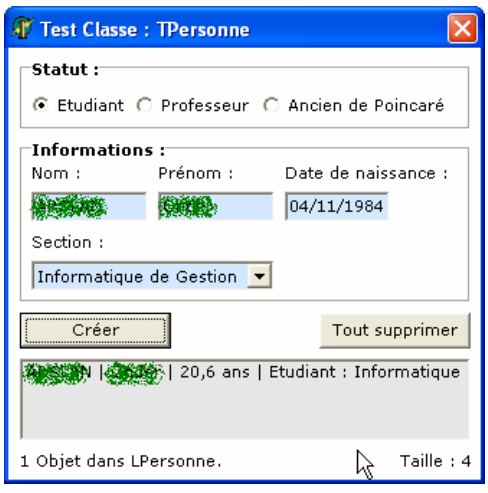
Méthode `Init()` de la sous-classe TProfesseur.

```
   constructor TProfesseur.Init(aNum: integer; aNom: string; aPrenom: string;
aDate: TDateTime; aMat: string);
   begin
        inherited Init(aNum, aNom, aPrenom, aDate);
        Matiere := aMat;
   end;
```

- Le mot clé **inherited** va permettre au programme d'appeler la méthode `Init()` de la classe mère pour valoriser les attributs communs à la classe mère et à la sous-classe.
- La ligne `Matiere := aMat;` valorise l'attribut spécifique à la sous-classe.

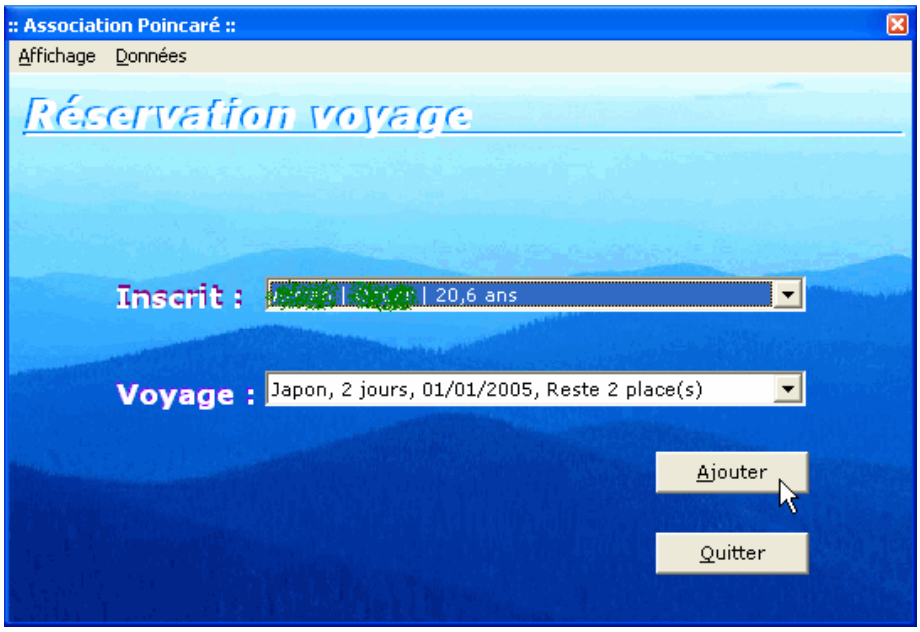
Aspect fonctionnel et technique

L'enregistrement de personne se fait à l'aide d'un programme dans lequel il suffit de saisir les informations dans les champs. Ensuite un bouton permet de valider et d'enregistrer la personne dans la base de données. *(cf. image ci-contre)*



La réservation de place pour un voyage se fait dans un autre programme qui exploite la même base de données. *(cf. image ci-dessous)*

Au lancement de l'application, les zones de liste ainsi que les collections d'objets contenant les personnes et les voyages sont chargés depuis la base de données à l'aide de requêtes SQL exécutées par le composant **TQuery**.



Requête SQL exécutée pour le chargement des personnes
`SELECT * FROM personnes WHERE P_statut <> 0 ORDER BY P_id`

Chargement des voyages

```
aVoyage := TVoyage.Init({liste des paramètres});
LVoyage.Add(aVoyage);
```

- `LVoyage` est une collection d'objet qui enregistre chaque objet `aVoyage` de type `TVoyage` initialisé.

- Ensuite, pour chaque voyage enregistré, une requête SQL va récupérer la liste des personnes ayant réservé une place :

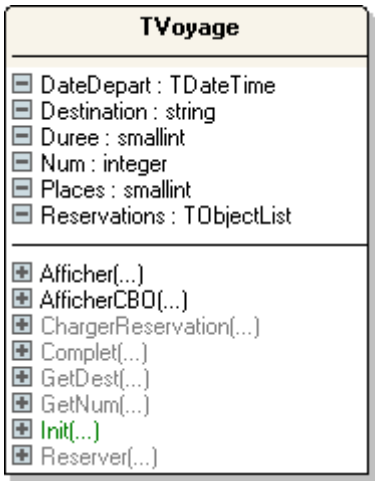
```
SELECT * FROM reserver
WHERE R_V_id = :num_v
ORDER BY R_V_id, R_P_id;
```

ⓘ **`:num_v`** est une variable dite hôte qui va contenir le numéro du voyage pour lequel la requête va rechercher les réservations.

- Puis lorsque nous avons la liste des personnes, il suffit de charger cette liste dans la collection `Reservations` du voyage concerné.

```
aVoyage.ChargerReservation(aPersonne : TPersonne);
```

La classe **TVoyage** permet d'instancier tous les voyages enregistrés dans la base de données.



Pour enregistrer une réservation, l'utilisateur doit sélectionner une personne et un voyage proposé dans les listes. Lorsqu'il valide son choix :

- Le programme définit la requête SQL, dans le composant TQuery nommé `SQL_Add`, qui va enregistrer la réservation. Il définit également les valeurs des deux variables hôtes nécessaires à l'enregistrement de la réservation dans la table « reserver ».
- Avant d'exécuter la requête, le programme vérifie qu'il reste des places pour le voyage.
- Si c'est le cas, la requête est exécutée et la méthode `Reserver` de l'objet TVoyage concerné est appelée avec en paramètre la personne qui a réservée sa place.
- Sinon un message d'erreur est affiché.

Extrait du code d'enregistrement d'une réservation

```
SQL_Add.SQL.add('INSERT INTO reserver VALUES (:num_voyage,:num_personne)');

SQL_Add.Params[0].AsInteger := TVoyage(LVoyage.Items[cboVoyage.ItemIndex]).GetNum;
SQL_Add.Params[1].AsInteger := TPersonne(LPersonne.Items[cboMembre.ItemIndex]).GetNum;

if TVoyage(LVoyage.Items[cboVoyage.ItemIndex]).Complet = false then
begin
      SQL_Add.ExecSQL;
      TVoyage(LVoyage.Items[cboVoyage.ItemIndex]).Reserver(
                              TPersonne(LPersonne.Items[cboMembre.ItemIndex]));
end
else
     self.MessageBox('Toutes les places sont réservées !', 'Erreur',0);
```

ⓘ Chaque fois qu'on souhaite travailler sur un objet de la collection, on doit « dire » au programme que l'objet est de *tel* type. On parle alors de **transtypage**. En effet, les objets stockés dans une collection peuvent être de n'importe quel type, d'où la nécessité de préciser.

Exemple : `TVoyage(LVoyage.Items[cboVoyage.ItemIndex]).GetNum;`

- `cboVoyage.ItemIndex` : retourne le numéro du voyage sélectionné dans la zone de liste.
- `LVoyage.Items[x]` : retourne le x-ème *(ici `cboVoyage.ItemIndex`)* objet de la collection.
- `TVoyage(objet).GetNum` : appel de la méthode `GetNum` de l'`objet` de type `TVoyage`. L'objet en question est bien évidemment celui retourné par `LVoyage.Items[x]`.

Conclusion

Cette activité fut très intéressante à développer car elle m'a permis d'appliquer et de comprendre les concepts intrinsèques à la programmation orientée objet, c'est à dire l'encapsulation, l'héritage et le polymorphisme. J'ai surtout apprécié le développement objet pour ses apports certains en terme de facilité d'utilisation et de sécurité.

L'activité démontre également les avantages de travailler avec des collections d'objets qui sont certainement la facilité, la rapidité et surtout l'efficacité.

Le dernier point important de cette activité est résolument l'intégration des requêtes SQL aux évènements de l'application. Elles permettent en effet de traiter les données facilement en un minimum de code.