

## ÉTUDE DE CAS : ASSOCIATION DES PERSONNES PORTEUSES DE TRISOMIE 21

ATRI-21 (Association d'aide au handicap Trisomiques 21), a pour but non lucratif affiliée à l'NUPH (Nations Unies pour les Personnes Handicapées). Elle œuvre en faveur des droits des personnes porteuses de trisomie 21 (T21) et de leurs familles, en favorisant leur insertion sociale et professionnelle.

L'ATRI-21 gère plusieurs Établissements d'accueils, d'Hébergements et d'Aide pour Travail (EHAT). Ces établissements permettent aux personnes T21 qui n'ont pas acquis suffisamment d'autonomie pour travailler en milieu ordinaire, d'exercer une activité professionnelle dans un milieu protégé. Chaque établissement EHAT dispose d'un ou plusieurs ateliers, par exemple :

- Un atelier de fabrication de produits cosmétiques ;
- Un atelier restauration-traiteur ;
- Un atelier espaces verts ;
- etc.

Chaque atelier offre des formations au profil des bénéficiaires (*personnes porteuses de trisomie 21*).

L'ATRI-21 dispose de plusieurs services de gestion des formations :

- Service de planification des formations : Application Desktop en VB.NET ;
- Service pour consulter les informations d'une formation : Application client/serveur en JAVA ;
- Service de gestion des formations : Application Web.

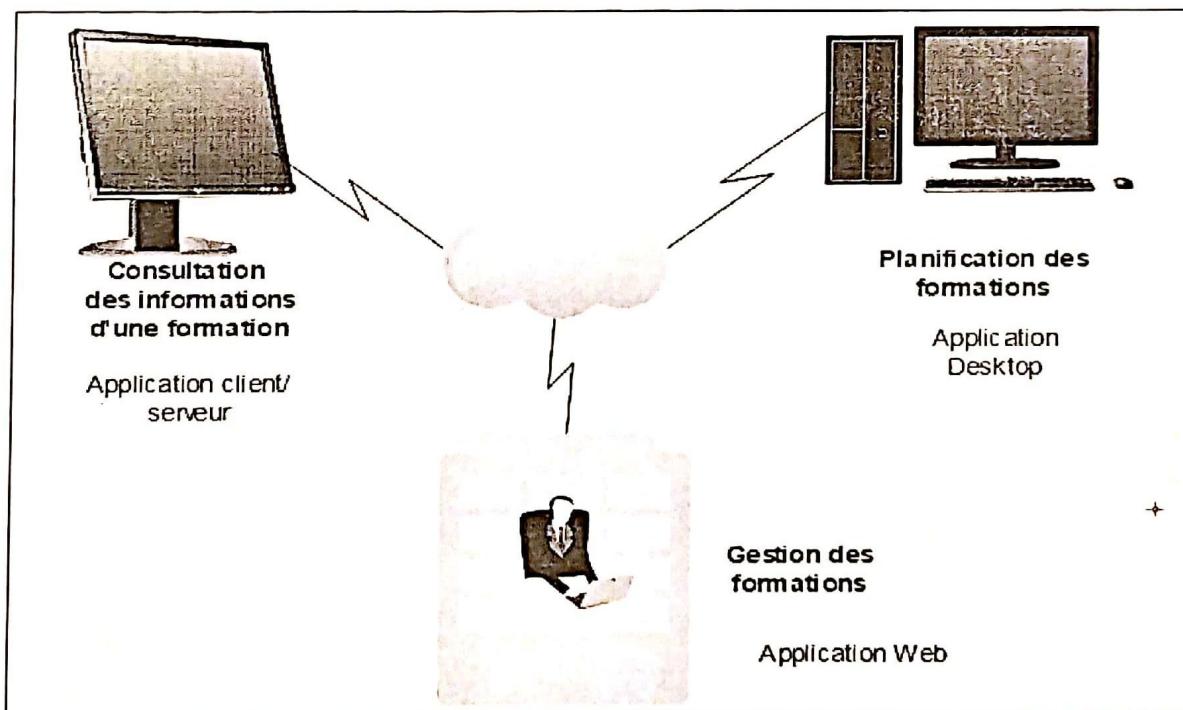


Figure 1 : Schéma structurel des services de l'association

**DOSSIER I : RÉCUPÉRATION DES INFORMATIONS D'UNE FORMATION****❖ SOUS DOSSIER 1-1 : STRUCTURE DE DONNÉES**

(14 pts)

Pour assurer cette tâche, on exploite plusieurs objets dont la structure est illustrée par le diagramme de classes suivant :

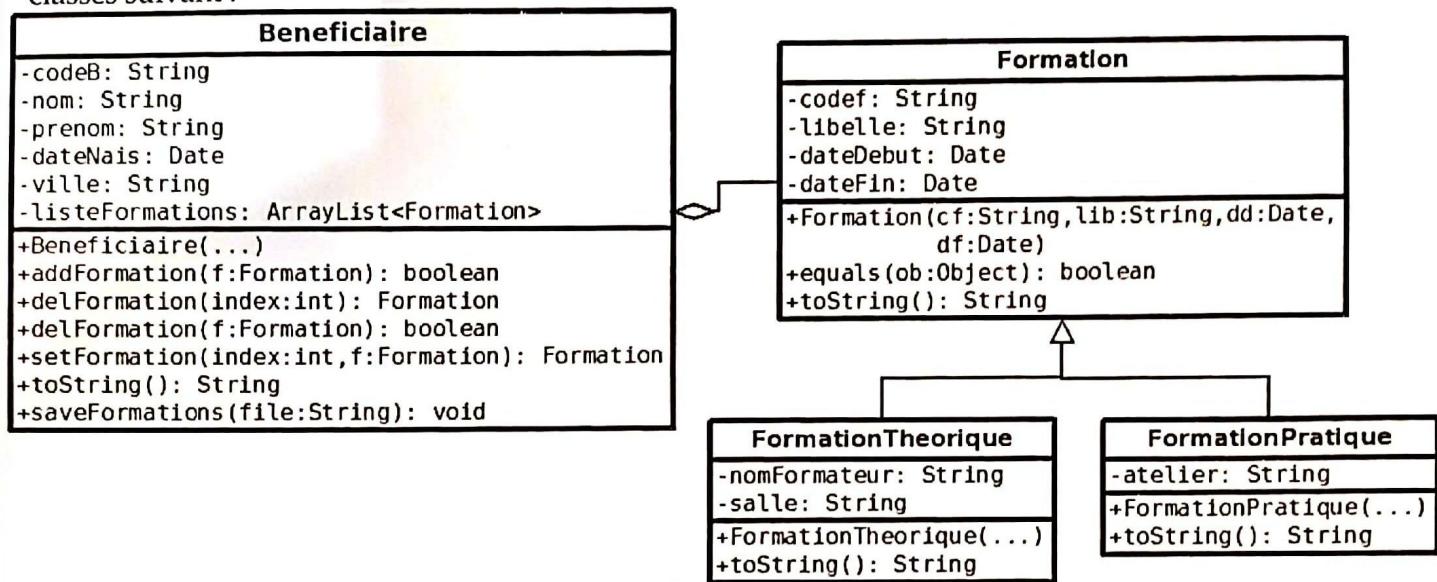


Figure 2 : Diagramme de classe

**1. Implémentation de la classe « Formation » :**

```

public class Formation implements Serializable{
    private String codef, libelle;
    private Date dateDebut, dateFin;

    public Formation(...) {...}
    //-----
    @Override
    public boolean equals(Object ob) { ... }
    //-----
    @Override
    public String toString() { ... }
}
  
```

- Définir un constructeur avec 4 arguments pour initialiser les attributs de cette classe. (1 pt)
- Redéfinir la méthode « **equals** » qui compare deux formations selon leurs codes (*la comparaison est insensible à la casse*). (1 pt)
- Redéfinir la méthode « **toString** », afin de retourner une chaîne porteuse d'informations sur une formation sous la forme : (1 pt)

Code : xxxx, Libellé : xxxx effectuée entre : jj/mm/aaaa et jj/mm/aaaa

**2. Implémentation de la classe « FormationTheorique » :**

```

public class FormationTheorique extends Formation {
    private String nomFormateur;
    private String salle;
    //-
    public FormationTheorique(...) { ... }
    //-
    @Override
    public String toString() { ... }
}
  
```

- a) Définir un constructeur de 6 arguments pour initialiser tous les attributs de cette classe. (1,5 pt)
- b) Donner la définition de la méthode « **toString** », afin de retourner une chaîne porteuse d'informations sur une formation théorique sous-forme : (1,5 pt)

Code : xxxx, Libelle : xxxx effectuée entre : jj/mm/aaaa et jj/mm/aaaa  
 Formateur : xxxx, salle : xxxx

### 3. Implémentation de la classe « **Beneficiaire** » :

```
public class Beneficiaire{
    private String codeB;
    private String nom, prenom;
    private String ville;
    private Date dateNais;
    private ArrayList<Formation> listeFormations;

    public Beneficiaire(...){ ... }
    //-----
    public boolean addFormation(Formation F){... }
    //-----
    public Formation delFormation(int index){ ... }
    //-----
    public boolean delFormation(Formation F){ ... }
    //-----
    public Formation setFormation(int index, Formation F)
    { ... }
    //-----
    @Override
    public String toString(){ ... }
    //-----
    public void saveFormations(String file){ ... }
}
```

Donner le code des méthodes suivantes :

- › a- Un constructeur avec **5** arguments qui initialise les **5** premiers attributs de la classe « **Beneficiaire** » et instancie la collection « **listeFormations** » ; (1 pt)
- › b- **addFormation(...)** : permet d'ajouter à la collection une nouvelle formation et retourne l'état de l'opération ; (1 pt)
- › c- **delFormation(int)** : permet de supprimer une formation de la collection en se basant sur un index et retourne la formation qui vient d'être supprimée. La vérification de la validité de l'index est indispensable ; (1 pt)
- › d- **delFormation(Formation ..)** : permet de supprimer une formation de la collection en se basant sur un objet « **Formation** » et retourne l'état de la suppression ; (1 pt)
- › e- **setFormation(...)** : permet de modifier une formation déjà existante, la vérification de la validité de l'index est indispensable, elle retourne la formation avant sa modification ; (1 pt)
- › f- **toString ()** : retourne une chaîne de caractères porteuse de toutes les informations d'un bénéficiaire sous-forme : (1 pt)

Code Ben : xxxx, Nom complet : xxxx ayant bénéficié des formations :  
 1- Code : xxxx, Libellé : xxxx effectuée entre : jj/mm/aaaa et jj/mm/aaaa  
 2- Code : xxxx, Libellé : xxxx effectuée entre : jj/mm/aaaa et jj/mm/aaaa  
 ...

- g- **saveFormations (String f)** : permet de sauvegarder toutes les formations théoriques dans un fichier d'objet ; (2 pts)

## ❖ SOUS-DOSSIER 1-2 : ARCHITECTURE CLIENT/SERVEUR (8 pts)

Afin qu'un responsable de l'association puisse récupérer des informations d'une formation, une architecture client/serveur est mise en place. Son schéma est donné par la figure suivante :

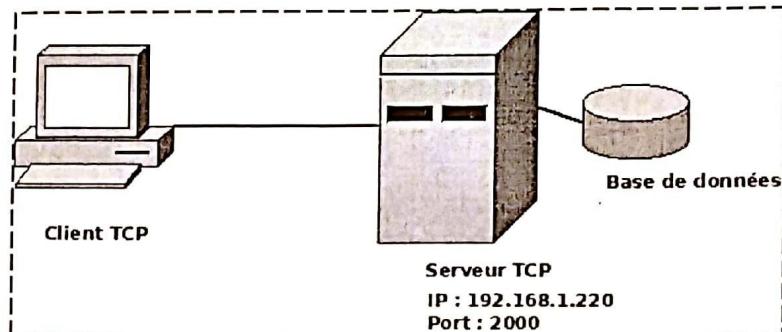


Figure 3 : Architecture client/serveur

1. S'agit-il d'une architecture 2 tiers ou 3 tiers ? justifier votre réponse. (0,5 pt)
2. Donner le modèle de Gartner-group correspondant à cette architecture. (1 pt)
3. Classe : Client TCP (3 pts)

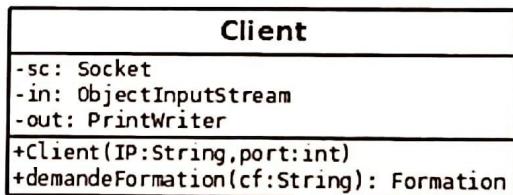


Figure 4 : Classe Client TCP

Implémenter la classe « client » qui contient :

- ✓ 3 attributs ;
- ✓ Un constructeur avec 2 arguments qui crée un socket et initialise les objets d'entrée/sortie **in** et **out** ;
- ✓ Une méthode **demandeFormation** qui envoie au serveur **TCP** le code d'une formation, reçoit et retourne l'objet « **formation** » correspondant à ce code.

```

public class Client {
    private Socket sc=null;
    private ObjectInputStream in = null;
    private PrintWriter out=null;
    public Client(String IP, int Port){...}
    public Formation demandeFormation(String cf)
    {...}
}
  
```

4. Classe : Serveur TCP

(3,5 pts)

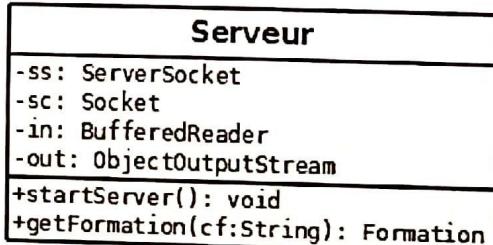


Figure 5 : Classe Serveur TCP

```

public class Serveur {
    private ServerSocket ss;
    private Socket sc=null;
    private BufferedReader in = null;
    private ObjectOutputStream out= null;
    public void startServer()
    {
        try{
            ... [1]
            while(true)
            {
                ... [2]
            }
        }catch(Exception ex) { }
    }
    public Formation getFormation(String cf)
    { /* code non demandé */}
}

```

Compléter la méthode « **startServer** » de la classe **Serveur** de telle manière :

» Dans la partie [1] :

- » - Instancier l'objet « **ServerSocket** » avec un port d'écoute : **2000** ;
- Accepter une demande de connexion du client **TCP** ;
- Préparer les objets d'entrée/sortie : **in** et **out**.

» Dans la partie [2] :

- Lire le code de la formation envoyé par le client **TCP** ;
- Faire appel à la méthode « **getFormation** » pour récupérer l'objet **Formation** correspondant au code de la formation ;
- Envoyer cet objet **Formation** au client **TCP**.

## DOSSIER II : PLANIFICATION DES FORMATIONS.

(10 pts)

Afin de gérer l'attribution des formations aux bénéficiaires, on utilise une application Desktop connectée à une base de données sous SQL-Server. Les caractéristiques de ce serveur sont :

- IP du serveur : **192.168.1.200** ;
- Nom de la base de données : **db\_Formation** ;
- Authentification de Windows.

Le **MLDR** de cette base de données est le suivant :

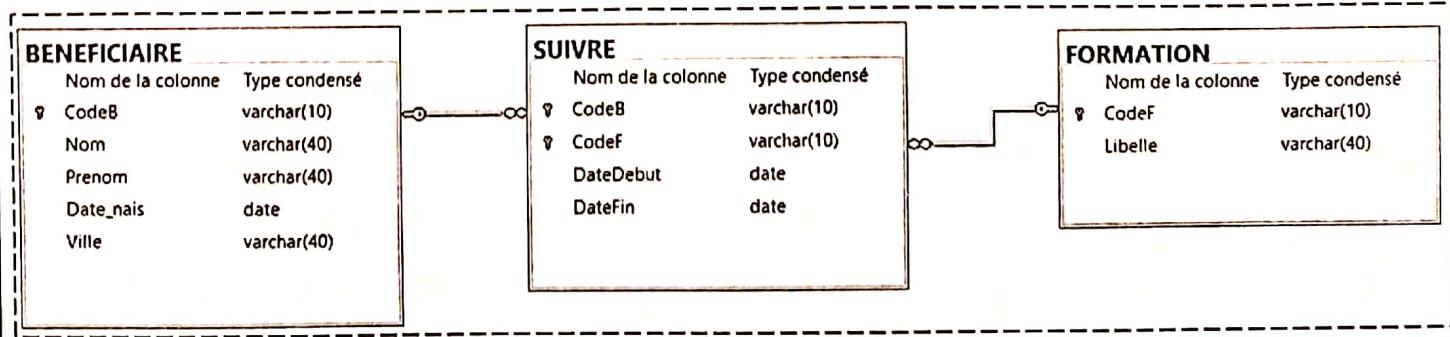


Figure 6 : Modèle logique de données relationnel

L'IHM de cette application est illustrée par la figure suivante :

The screenshot shows a Windows application window titled "Attribution des formations". At the top left, it says "Attribution des modules". The main area has several input fields and dropdown menus:

- Bénéficiaire: KADIRI Amine
- Date début: mercredi 17 juin 2020
- Code Formation: F1
- Date fin: mercredi 17 juin 2020
- Libellé Formation: Fabrication des produits cosmétiques

At the bottom right are two buttons: "Ajouter" (Add) and "Fermer" (Close).

Below this, there is a section titled "Liste des formations d'un bénéficiaire" (List of trainings for a beneficiary). It contains a table with the following data:

Code Formation	Libellé	Date Début	Date Fin
F1	Fabrication des produits cosmétiques	15/06/2020	24/07/2020
F2	Restauration	09/03/2020	30/09/2020

Figure 7 : IHM des attributions des formations

Objet	Name	Texte
Boutons de commandes	CmdAjouter	Ajouter

Objet	Name
Label	LblFormation
DataGridView	DGVListe
DateTimePicker	dtpDebut
	dtpFin
ComboBox	cmbBen
	cmbCodeF

### TRAVAIL DEMANDÉ

Remarque : Vous avez le choix d'utiliser le mode connecté ou non connecté.

- 1) Dans un module, déclarer les objets de connexion et créer la procédure « Connexion() » pour établir une connexion avec le serveur de base de données. Il faut gérer les exceptions. (1,5 pt)

Signature de la procédure

```
Public Sub Connexion()
    ...
End Sub
```

- 2) Dans la classe de l'IHM, Écrire le code de la procédure « charger\_CodeForm() » qui charge le comboBox « cmbCodeF » par les codes de toutes les formations. (1,5 pt)

Signature de la fonction

```
Private Sub charger_CodeForm()
    ...
End Sub
```

- 3) Écrire le code de la fonction « getLibelleForm(...) » qui reçoit, en argument, le code d'une formation et retourne son libellé. (1,5 pt)

Signature de la fonction

```
Private Function getLibelleForm(ByVal CodeF As String) As String
    ...
End Sub
```

- 4) Écrire le code de la procédure « `affiche_ListeForm()` » qui liste toutes les formations correspondantes à un bénéficiaire dans l'objet `DataGridView` nommé « `DGVListe` ». La procédure prend en argument le code du bénéficiaire. Voici un aperçu de l'objet `DataGridView`. (2 pts)

Code Formation	Libellé	Date Début	Date Fin
----------------	---------	------------	----------

Signature de la procédure

```
Private Sub affiche_ListeForm(ByVal codeBen As String)
    ...
End Sub
```

- 5) Écrire le code de la fonction « `keyExist(...)` » qui vérifie l'existante de la clé primaire de la table « `SUIVRE` ». Le code du bénéficiaire et celui de la formation sont donnés en arguments. (1,5 pt)

Signature de la procédure

```
Private Function keyExist(ByVal CodeF As String, ByVal codeB As String) As Boolean
    ...
End Function
```

- 6) On suppose que l'objet `comboBox` « `CmbBen` » est rempli par les noms complets de bénéficiaires et que les dates de début et de fin de la formation sont valides. (2 pts)

Donner le code de la procédure événementielle du bouton « `cmdAjouter` » permettant de :

- ✓ Vérifier l'unicité de la clé primaire de la table « `SUIVRE` » ;
- ✓ Insérer un nouvel enregistrement dans cette table en se basant sur les données offertes par le formulaire ;
- ✓ Actualiser l'affichage dans `DataGridView`.

Signature de la procédure

```
Private Sub cmdAjouter_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles cmdAjouter.Click
    ...
End Sub
```

### DOSSIER III : GESTION DES FORMATIONS.

(8 pts)

On souhaite créer une application Web, permettant à l'administrateur de visualiser les formations disponibles et de lister les bénéficiaires de chaque formation.

Sous MySQL, on implémente une copie de la base de données `db_Formation` dont le MLDR est :

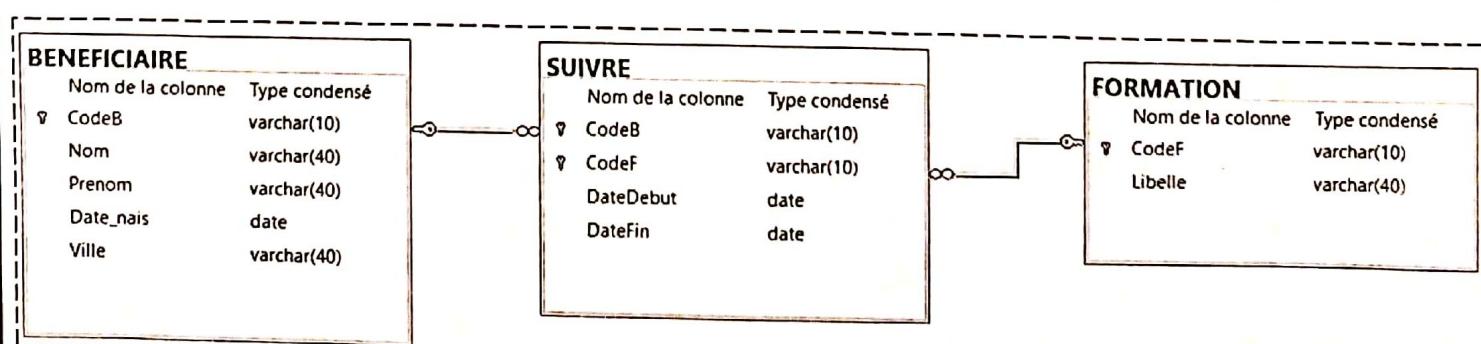


Figure 8 : Modèle logique de données relationnel

## TRAVAIL DEMANDÉ

1) Donner le code du script « connexion.php » permettant la connexion à la base de données « db\_formation ». (2 pts)

On donne :

- Nom du serveur : **Srv\_ATRI21**
- Nom utilisateur : **association**
- Mot de passe : **association123**

2) La page d'accueil : « index.php » affiche la liste des formations : (3 pts)

Liste des formations		
Code Formation	Libelle	
f1	fabrication de produits cosmétiques	<a href="#">Liste bénéficiaires</a>
f2	Jardinage	<a href="#">Liste bénéficiaires</a>

Figure 9 : Page d'accueil : index.php

Le script « index.php » est le suivant :

```

<html>
    <head>
        <title> tableau de bord administrateur </title>
        <style type="text/css"> ... </style>
    </head>

    <body>
        <?php include('connexion.php'); ?>
        <fieldset>
            <h1> Liste des formations </h1>
            <table>
                <tr>
                    <th> Code Formation </th>
                    <th> Libellé </th>
                    <th> </th>
                </tr>
                <?php
                    // partie à compléter
                ?>
                </table>
            </fieldset>
        </body>
    </html>

```

Compléter la partie PHP du script ci-dessus pour lister toutes les formations.

Le lien hypertexte permet la redirection vers la page « `Istben.php` » en envoyant le code de la formation via URL.

- 3) La page « `Istben.php` » est la suivante : (3 pts)

Liste des bénéficiaires de la formation : fabrication de produits cosmétiques				
Code Beneficiaire	Nom	Prenom	Date Naissance	Ville
b1	MAHMOUDI	Asmae	2000-02-10	Rabat
b2	SALHI	Mohamed	2002-04-15	CASA
<a href="#">Retour</a>				

Figure 10 : Liste des bénéficiaires : `Istben.php`

Le script « `Istben.php` » est le suivant :

```
<html>
<body>
    <?php include('connexion.php') ; ?>
    <!-- partie à compléter -->
    <center><a href="index.php">Retour</a></center>
</body>
</html>
```

Compléter le script ci-dessus pour lister tous les bénéficiaires de la formation sélectionnée au niveau de la page d'accueil.

Le titre de cette page doit mentionner le libellé de la formation choisie.