Filière :	Développement des Systèmes d'Information	Durée :	4 Heures
Épreuve :	Développement d'Applications Informatiques – DAI -	Coefficient :	45

ÉTUDE DE CAS : SOCIÉTÉ DE SERVICES EN INGÉNIERIE INFORMATIQUE

DOSSIER 1 : GESTION DES FILIALES *(14 pts)*

1. La classe « **Solution** ».

```
// exeception du constructeur - 0,25pt
class ErreurPrix extends Exception {
    public ErreurPrix(String message) {
        super(message);     }
}

//  la classe abstraite « Solution » - 0,5 pt
public abstract class Solution implements Serializable, Comparable {
private String code;
private double prix;
public abstract double montant();

//  Définir un constructeur avec arguments - 0,75pt
    public Solution(String code, double prix) throws ErreurPrix {
        if (prix < 0) {
            throw new ErreurPrix("Erreur de prix de la solution");  }
        this.code = code;
        this.prix = prix;
    }

//1.3  Définir l'accesseur getPrix()  - 0,5 pt
    public double getPrix() {
        return prix;
    }

//  Donner la définition de la méthode « toString » - 0,5 pt
    @Override
    public String toString() {
        return "code=" + code + ", prix=" + prix;
    }

//  Redéfinir la méthode « compareTo » de l'interface Comparable (0,5 pt)
  @Override
    public int compareTo(Object o) {
        Solution s = (Solution) o;
        if (prix > s.prix) {
            return 1;
        }
        if (prix < s.prix) {
            return -1;
        }
        return 0;
    }

  //  Redéfinir la méthode « equals » - 0,5 pt
  @Override
    public boolean equals(Object obj) {
        Solution other = (Solution) obj;
        return code.equals(other.code);
    }
}
```

2. La classe « **Service** ».

```
// Implémenter la classe « Service ».  (0,5 pt)
public class Service extends Solution {
    private String designation;
    private int dureeAnnee;

//  Proposer un constructeur (0,5 pt)
    public Service(String designation, int dureeAnnee, String code, double prix)
throws ErreurPrix {
        super(code, prix);
        this.designation = designation;
        this.dureeAnnee = dureeAnnee;
    }

// Donner le code de la méthode montant( ) (1 pt)
    @Override
    public double montant() {
        double m = getPrix() * dureeAnnee;
        if (dureeAnnee >= 3) {
            m -= m * 0.2;
        }
        return m;
    }

// Redéfinir la méthode « toString », afin de retourner une chaine porteuse d'informations sur une solution, la chaine aura la
forme suivante :  (0,5 pt)
    @Override
    public String toString() {
        return super.toString()+" ,Désignation : "+designation+", Durée (années) :
"+dureeAnnee;
    }
}
```

3. La classe « **Outil** ».

```
// Implémenter la classe« Outil ». (0,5 pt)
public class Outil extends Solution {
    private String libelle;
    private int nbrLicences;


// Proposer un constructeur (0,5 pt)
    public Outil(String libelle, int nbrLicences, String code, double prix) throws
ErreurPrix {
        super(code, prix);
        this.libelle = libelle;
        this.nbrLicences = nbrLicences;
    }
// Donner le code de la méthode montant( ) permettant de calculer le montant total :  (0,5 pt)
    @Override
    public double montant() {
        double m = getPrix() * nbrLicences;
        if (nbrLicences >= 10) {
            m -= m * 0.1;
        }
        return m;
    }
}
```

4. La classe« **Filiale** ».

```
// Implémenter la classe« Filiale ».(0,5 pt)
public class Filiale {
    private String nom, localite, specialite;
    private Date dateCreation;
    private ArrayList<Solution> listeSolutions;
```

```
//Proposer un constructeur adéquat avec 4 paramètres (0,5 pt)
    public Filiale(String nom, String localite, String specialite, Date
dateCreation) {
        this.nom = nom;
        this.localite = localite;
        this.specialite = specialite;
        this.dateCreation = dateCreation;
        listeSolutions = new ArrayList<Solution>();
  }
// Définir l'accesseur getListeSolutions ().(0,5 pt)
    public ArrayList<Solution> getListeSolutions() {
        return listeSolutions;
    }
// Donner le code des méthodes suivantes :
//  addSolution(Solution) :          (1 pt)

    public boolean addSolution(Solution s) {
        return listeSolutions.add(s);
    }
// delSolution(int) :indispensable.(0,75 pt)
    public Solution delSolution(int i) {
        try {
            return listeSolutions.remove(i);
        } catch (Exception e) {
            return null;          }
    }
// delSolution(Solution) : (0,75 pt)
    public boolean delSolution(Solution s) {
        return listeSolutions.remove(s);
    }
// toString( ) : (0,5 pt)
    @Override
    public String toString() {
        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
        String c = "nom : " + nom + ", date de Creation : " +
sdf.format(dateCreation) + "\n La liste des solutions : \n";
        int i = 1;
        for (Solution listeSolution : listeSolutions) {
            c += i + "-" + listeSolution.toString() + "\n";
        }
        return c;
    }

// saveData(String): (1 pt)
    public void saveData(String f) {
        try {
            ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(f));
            for (Solution listeSolution : listeSolutions) {
                if (listeSolution.getPrix() > 300) {
                    oos.writeObject(listeSolution);
                    oos.flush();
                }
            }
            oos.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
// loadData(String): (1 pt)
    public void loadData(String f) {
        try {
            ObjectInputStream ois = new ObjectInputStream(new FileInputStream(f));
            listeSolutions.clear();
            Solution s;
            while (true) {
                try {
                    s = (Solution) ois.readObject();
                    listeSolutions.add(s);
                } catch (Exception e) {
                    break;  }
            }
            ois.close();
        } catch (Exception e) {
            e.printStackTrace();   }
    }
}
```

DOSSIER 2 : SERVEUR D'APPLICATION *(8 pts)*

A. Programmation Serveur

```
public class Connect {
    private Connection cn;   //objet de connexion
    private static Connect A = null;
// 1   Donner le code du constructeur permettant d'établir une connexion (1pt)
    private Connect() {
        try {
            cn = DriverManager.getConnection("jdbc:mysql://81.172.0.2: 3306/
SolutionsInfo", "Serveur", "112233");
        } catch (Exception e) {
            e.printStackTrace();        }
    }

public static Connect getInstance() {
        if (A == null) {
            A = new Connect();          }
        return A;
    }
```

// 2. Une méthode « lireServices» permet de retourner un objet « ResultSet » contenant tous les services ayant le prix reçu comme argument. En cas d'échec, elle retournera « null ». (1 pt)

```
public ResultSet lireServices(double prix) {
    try {
            String req = "select * from service where prix=" + prix;
            Statement stm = cn.createStatement();
            return stm.executeQuery(req);
        } catch (Exception e) {
            return null;
        }
    }
```

//3. Une méthode « addService» permet d'ajouter un enregistrement à la table service. En cas d'échec, un message d'erreur sera affiché. (1 pt)

```
public void addService(String code, double Prix, String Des, int duree) {
   try {
      String req = "insert into service values('" + code + "', " + Prix + ", '" + Des
+ "'," + duree + ")";
            Statement stm = cn.createStatement();
            stm.executeUpdate(req);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
//4.  Une méthode « toArray» permet de (1 pt)
    public ArrayList<String> toArray(ResultSet rs) {
        try {
            ArrayList<String> arrS = new ArrayList<>();
            while (rs.next()) {
                arrS.add(rs.getObject(1) + " : " + rs.getObject(3) + " ,
                 " + rs.getObject(2));
            }
            return arrS;
        } catch (Exception e) {
            return null;  }
    }

}
```

//5. (0,5 pt)

L'instanciation de la classe « **Connect** » est faite une seule fois. Car l'attribut statique « **A** » n'est instancié que lorsqu'il est différent du **nul**. Cette condition est assurée par la méthode statique « **getInstance** ».

B. Programmation client

```
public class Client {
    private String IPServeur;
    private int PortServeur;
    private Socket s = null;
    private PrintWriter pw = null;
    private ObjectInputStream ois = null;

    public Client(String IP, int Port) {
        this.IPServeur = IP;
        this.PortServeur = Port;
    }
```

//1. Écrire une méthode connecter (1pt)

```
public boolean connecter() {
        try {
            s = new Socket(IPServeur, PortServeur);
            pw = new PrintWriter(s.getOutputStream());
            ois = new ObjectInputStream(s.getInputStream());
            return true;
        } catch (Exception e) {
            return false;
        }
    }
```

//2. Écrire la méthode demandeListeServices (1 pt)

```
    public ArrayList<String> demandeListeServices(double prix) {
        try {
            pw.println(prix);
            return (ArrayList<String>) ois.readObject();
        } catch (Exception e) {
            return null;  }
    }
}
```

//3. *(1.5 pt)*

```
Private void cmdEnvoyerActionPerformed(java.awt.event.ActionEvent evt) {
      ArrayList<String>
    arrD=clt.demandeListeServices(Double.parseDouble(txtPrix.getText()));
      liste.setListData((String[]) arrD.toArray());
    }
```

DOSSIER 3 : SUIVI DES FORMATIONS. *(10 pts)*

1. Le code de la procédure « **Connexion** » *(1.5 pts)*

```
Public CN as SqlConnection
Sub Connexion()
  Try
    CN=new SqlConnection("Initial Catalog= BD_Formations ;
       Data Source= Srv-FormatNet; Integreted Security=true; ")
    CN.open()
  Catch
     MessageBox.Show("Erreur de connexion")
  End Try
End Sub
```

2. Écrire le code de la procédure« **Lister_demandes** » *(2 pts)*

```
Sub Lister_demandes(ByVal Etat As String)
  Dim req As String="select N_Demande As [N° Demande], Date_Demande As
  [Date], Titre, Nom+'  '+Prenom As [Nom & Prénom Emplyé], Raison_Sociale
  As [Société], Etat_Demande as Etat from Demande D, Employe E, Societe S,
  Formation F where D.Matricule=E.Matricule and D.Formation=F.Numero and
  E.Societe = S.ID_STE and D.Etat=' "+Etat+" ' "
  Dim cmd as New SqlCommand(req,CN)
  SqlDataReader rd=cmd.ExecuteReader( )
  Dim T as New DataTable
  T.load(rd)
  rd.Close()
  DGListe.DataSource=T
End Sub
```

3. Écrire le code de la procédure« **Supprimer_demande** » *(1 pts)*

```
Public Sub Supprimer_Demande (ByVal N_Demande As Integer)
   Dim req As String="delete from Demande where N_Demande= "+
   N_Demande.ToString
   Dim cmd as New SqlCommand(req,CN)
   cmd.ExecuteNonQuery( )
End Sub
```

4. On considère l'interface suivante, qui permet d'ajouter une nouvelle demande.

a) Le code d'une fonction nommée « **Generer_Code** » *(1 pts)*

```
Public Function Generer_Code() As Integer
   Dim req As String="select max(N_Demande) from Demande"
   Dim cmd as New SqlCommand(req,CN)
   Return (cmd.ExecuteScalar()+1)
End Function
```

b) Le code permettant d'afficher, dans l'objet « **LblDescription** », la description de la formation sélectionnée dans le **Combobox** « **CmbTitre** ». *(1 pt)*

```
Sub Afficher_Description()
   Dim req As String="select Description from Formation where
   Titre='"+cmbTitre.SelectedValue+"'"
   Dim cmd as New SqlCommand(req,CN)
   Dim res As String= cmd.ExecuteScalar()
     LblDescription.Text=res
End Sub
```

c) Le code de la procédure « **Remlpir_Cmb_Societe** » *(1 pt)*

```
Sub Remlpir_Cmb_Societe ()
    Dim req As String="select * from Societe"
    Dim cmd as New  SqlCommand(req,CN)
    SqlDataReader rd=cmd.ExecuteReader( )
    Dim T as New DataTable
    T.load(rd)
    rd.Close()
    CmbSociete.DisplayMember="Raison_sociale"
    CmbSociete.ValueMember="ID_Ste"
    CmbSociete.DataSource=T
End Sub
```

d) Le code correspond à l'événement **Clic** du bouton **Enregistrer** permettant d'ajouter une nouvelle demande. *(2 pt)*

On suppose que le Combobox « ***Cmbtitre*** » est déjà rempli.

```
Private Sub CmdEnregistrer_Click(.....)
   Dim req As String="insert into Demande values ("+LblNum_Demande+",
   '"+Date_Demande.Value+"', '"+CmbMatricule.SelectedValue+"', null,
   "+CmbTitre.SelectedValue+", 'En Cours')"
   Dim cmd as New  SqlCommand(req,CN)
   cmd.ExecuteNonQuery( )
End Sub
```

DOSSIER 4 : SITE WEB DYNAMIQUE DE LA SOCIÉTÉ *(8 pts)*

1. Fichier« **connexion.php**» qui permet la connexion à la base de données «**gssii**». *(1 pt)*

$c=mysqli_connect("localhost","root","","gssi");

Le code de la page d'accueil « **index.php** » permettant charger la liste déroulante par les noms des entreprises à partir de la table **Entreprise**. *(2 pts)*

```
<form id="f" name="f" method="post" action="filiale.php">
   <table> <tr><td>
    <b>Sélectionner l'entreprise :</b></td></tr>
    <tr><th>
    <select name="ide" id="ide">
    <?php
    include_once("Connexion.php");
    $r=mysqli_query($c,"select * from entreprise");
    while($tab=mysqli_fetch_row($r)){
     echo "<option value='$tab[0]'>$tab[1]</option>";
     }
     ?>
    </select>
    </th></tr>
    <tr><th><input type="submit" value="Afficher liste des filiales"
    id="bouton"></th></tr>
   </table>
</form>
```

2. *(3 pts)*

```
<?php
   //Code PHP à compléter
   $x=$_POST['ide'];
   include_once("Connexion.php");
   $r=mysqli_query($c,"select * from filiale where ide=$x");
   while($tab=mysqli_fetch_row($r)){
     echo "<tr>";
     for($i=0;$i<count($tab)-1;$i++) echo "<td>$tab[$i]</td>";
       echo "<th><a  href='supprimer.php?w=$tab[0]'> Supprimer</a> </th>
   </tr>";
      }
      $n=mysqli_num_rows($r);
    echo "<tr><th colspan='6'>Nombre de Filiales : $n</th></tr>";
?>
```

3. *(2 pts)*

```
$x=$_GET['w'];
include_once("Connexion.php");
mysqli_query($c,"delete from filiale  where idf=$x");
```