LaSalle College
Montréal

Évaluation finale : 40 %

Identification du cours

Nom des programmes	TECHNIQUES DE L'INFORMATIQUE SPECIALISATION EN INFORMATIQUE DE GESTION (420.A0) PROGRAMMEUR ANALYSTE EN TECHNOLOGIES DE L'INFORMATION (LEA.3Q) DÉVELOPPEMENT D'APPLICATIONS BASES DE DONNÉES 420-B35-AS
Titre du cours :	INTRODUCTION AU DÉVELOPPEMENT WEB
Numéro du cours :	420-B35-AS
Groupe:	07177
Nom de l'enseignant:	Sehboub Mourad
Durée:	3 périodes (150 minutes)
Semestre:	Automne 2024

Identification de l'étudiant

Nom: _____ Numéro d'étudiant: _____

Date: **2024 – 12 – 17** Résultat: _____

☐ Je déclare qu'il s'agit d'une œuvre originale, et que j'ai crédité toutes les sources de contenu dont je ne suis pas l'auteur (en ligne et imprimé, images, graphiques, films, etc.), dans le style de citation et de citation requis pour ce travail.

Énoncé de la compétence évaluée – Code

Contexte de réalisation propre à la compétence Assurer la qualité d'une application -017B :

- À partir d'une station de travail et des outils de développement propres au système de gestion de la base de données.
- À partir d'une demande de développement d'une application limitée à quelques fonctions.
- À partir des exigences de l'entreprise et des standards de l'informatique.
- À partir des manuels de références techniques appropriés.

- En collaboration avec les personnes en cause dans le développement de l'application.

Consignes :

- Aucune pause ne sera accordée pendant l'examen. Aucun étudiant ne peut quitter la salle d'examen avant que la moitié du temps alloué ne se soit écoulée. L'étudiant qui quitte la salle d'examen ne peut y revenir (PIEA – Article 5.12.4).
- L'enseignant(e) ne répondra pas aux questions.
- Les étudiants ne sont pas autorisés à communiquer entre eux.
- L'enseignant(e) a le devoir d'identifier les erreurs de langue dans les travaux des étudiants. Un pourcentage attribué à la langue va jusqu'à 5 % de la note (PIEA – Article 5.7).
- Le plagiat, la tentative de plagiat ou la coopération à un plagiat lors d'une épreuve sommative entraîne la note zéro (0). Dans le cas de récidive, dans le même cours ou dans un autre cours, l'étudiant se voit octroyer un « 0 » pour le cours concerné. (PIEA – Article 5.16).
- Veuillez écrire de façon lisible.

Répartition des points

Cette évaluation est sur 100 points répartis comme suit :

Partie 1 : Application bibliothèque	**Pour un total de 50 points**
Question 1 : Définition du système	**10 points**
Question 2 : Définition du protocole de communication	**10 points**
Question 3 : Définition d'une collection de données	**10 points**
Question 4 : Validation des entrées/sorties des données	**10 points**
Question 5 : Gestion des exceptions	**10 points**

Partie 2 : Jeux libGDX	**Pour un total de 50 points**
Question 1 : database connexion	**10 points**
Question 2 : ajout utilisateur SGBD	**10 points**
Question 3 : ajout de méthode score	**10 points**
Question 4 : ajout de méthode logique	**10 points**
Question 5 : ajout de méthode mathématiques	**10 points**

TOTAL : 100 POINTS

Enoncé :

La Bibliothèque Municipale de Montréal désire implémenter une application qui lui permettrait de gérer efficacement les informations des abonnés, les livres et les emprunts. Un abonné peut être soit un abonné **standard**, soit un abonné **premium**. Les abonnés standard ont un nombre limité de livres qu'ils peuvent emprunter, tandis que les abonnés premium peuvent emprunter un nombre illimité de livres.

Un abonné doit être caractérisé par un identifiant unique, un nom, un prénom, une adresse, une date d'inscription, un type d'abonnement et une liste d'emprunts effectués.

Lors de l'inscription, l'abonné doit choisir un type d'abonnement et payer une cotisation annuelle. La cotisation des abonnés standard est différente de celle des abonnés premium. Les frais sont calculés comme suit :

- **Abonné standard** : frais d'inscription annuelle + frais de cotisation annuelle.
- **Abonné premium** : frais d'inscription annuelle + frais de cotisation annuelle premium.

L'application doit permettre de suivre les emprunts de livres par les abonnés, et chaque emprunt a une durée limite. Un emprunt ne peut pas dépasser 30 jours.

Question 1 : Définition du système (20%)

Définissez et implémentez toutes les classes nécessaires pour représenter les abonnés, les livres, et les emprunts.

- La classe **Abonne** doit inclure des attributs tels que l'identifiant, le nom, le prénom, l'adresse, la date d'inscription, le type d'abonnement et une liste des emprunts en cours.

- La classe **Livre** doit inclure des attributs comme un identifiant unique, le titre, l'auteur, et la disponibilité (emprunté ou disponible).

- La classe **Emprunt** doit inclure des informations sur la date de début de l'emprunt et la date de retour prévue.

De plus, assurez-vous qu'on puisse rechercher un abonné à l'aide de son <identifiant unique> ou de son <nom>.

Question 2 : Définition du protocole de communication (10%)

Créez une interface **cotisation** appelée « **Cotisation** » comportant la méthode «**calculCotisation**» qui doit être implémentée par les classes d'abonnés (standard et premium).

Cette méthode permettra de calculer le montant total de la cotisation d'un abonné, en fonction de son type d'abonnement.

Question 3 : Définition d'une collection de données (10%)

Créez une classe permettant d'encapsuler la liste des abonnés. Utilisez une **HashSet** pour la collection des abonnés afin de garantir qu'il n'y ait pas de doublons.

Rappel :

Une **HashSet** en Java est une collection qui fait partie du framework **Collection** et implémente l'interface **Set**. Un HashSet est une **structure de données non ordonnée** qui **ne permet pas les doublons**, ce qui signifie qu'il ne peut contenir qu'une seule occurrence de chaque élément.

Exemple :

```
HashSet<String> set = new HashSet<>();

set.add("Java");

set.add("C#");

set.add("Java");  // Ne sera pas ajouté, car "Java" existe déjà dans l'ensemble

System.out.println(set);  // Affiche [Java, C#]
```

- Définissez la propriété, les constructeurs, les méthodes publiques de lecture et d'écriture, ainsi que la méthode permettant d'afficher l'état de l'objet « liste d'abonnés ».

- Ajoutez des méthodes permettant d'assurer la gestion des abonnés : Ajout, Suppression, Recherche (par identifiant ou nom), Visualisation des abonnés.

- La liste des abonnés doit être ordonnée par identifiant.

Question 4 : Validation des entrées/sorties des données (10%)

Définissez et créez une classe utilitaire appelée « **Terminal** » qui sera utilisée pour effectuer des saisies valides de données (comme le nom, le prénom, etc.) et afficher les résultats à l'écran.

- Cette classe doit permettre de saisir les données depuis le clavier et d'afficher des messages à l'écran de manière conviviale.

- Tous les services de cette classe doivent être partagés pour éviter la duplication de code.

Question 5 : Gestion des exceptions (10%)

Créez une classe d'exception pour contrôler le paiement de la cotisation annuelle qui ne doit pas dépasser un certain montant (par exemple, 200 $ pour un abonné standard et 350 $ pour un abonné premium).

- Créez également une classe d'exception pour contrôler la validité des données saisies par l'utilisateur (par exemple, éviter que le nom ou le prénom soit vide, ou que la date d'emprunt soit dans le futur).

Pour vous aider à créer une base de données complète pour le système de gestion d'une bibliothèque, je vais vous fournir un exemple d'implémentation avec la création de la base de données, des tables associées, les requêtes SQL nécessaires, ainsi que des données d'exemple pour les insérer dans les tables.

Dans ce travail, nous utilisons **MySQL** comme système de gestion de base de données relationnelle (SGBDR) pour stocker et gérer les informations relatives aux abonnés, aux livres et aux emprunts dans la bibliothèque.

Le serveur de base de données MySQL est configuré via **XAMPP**, qui exécute Apache, MySQL et PHP localement sur votre machine. J'ai créé la base de données **bibliothèque** et à partir de DBManager d'Intellij IDE j'ai exécuté toutes les requêtes nécessaires que voici (ce fichier sql vous sera donné)

```
-- Création de la base de données

CREATE DATABASE Bibliotheque;

-- Sélectionner la base de données à utiliser

USE Bibliotheque;

-- Création des tables  « Abonne », « Livre », « Emprunt », et « Cotisation ».

 CREATE TABLE Abonne (
    idAbonne INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(100) NOT NULL,
    prenom VARCHAR(100) NOT NULL,
    adresse VARCHAR(255) NOT NULL,
    date_inscription DATE NOT NULL,
    type_abonnement ENUM('standard', 'premium') NOT NULL,
    cotisation DECIMAL(10, 2) NOT NULL
);
-- Table `Livre` pour stocker les informations des livres
CREATE TABLE Livre (
    idLivre INT AUTO_INCREMENT PRIMARY KEY,
    titre VARCHAR(255) NOT NULL,
    auteur VARCHAR(255) NOT NULL,
```

```
    annee_publication INT NOT NULL,
    disponible BOOLEAN NOT NULL DEFAULT TRUE
)
-- Table `Emprunt` pour enregistrer les emprunts effectués par les abonnés
CREATE TABLE Emprunt (
    idEmprunt INT AUTO_INCREMENT PRIMARY KEY,
    idAbonne INT NOT NULL,
    idLivre INT NOT NULL,
    date_emprunt DATE NOT NULL,
    date_retour DATE,
    FOREIGN KEY (idAbonne) REFERENCES Abonne(idAbonne) ON DELETE CASCADE,
    FOREIGN KEY (idLivre) REFERENCES Livre(idLivre) ON DELETE CASCADE
);

--Insertion de données dans la table `Abonne`
INSERT INTO Abonne (nom, prenom, adresse, date_inscription, type_abonnement, cotisation)
VALUES
(Julien, Paul, '123 Rue des Jasmins, '2024-01-15', 'standard', 50.00),
('Martin', 'Sophie', '456 Rue Laval', '2024-02-20', 'premium', 100.00),
('Durand', 'Dominique', '767, blvd. Bourassa', '2024-03-10', 'standard', 50.00),
('Lemoine', 'Julie', '101 Rue Kepler', '2024-04-05', 'premium', 120.00);

-- Insertion de données dans la table `Livre`
INSERT INTO Livre (titre, auteur, annee_publication, disponible)
VALUES
('Les Misérables', 'Victor Hugo', 1862, TRUE),
('Le Petit Prince', 'Antoine de Saint-Exupéry', 1943, TRUE),
('1984', 'George Orwell', 1949, TRUE),
('La Peste', 'Albert Camus', 1947, TRUE),
```

```
('Les Fleurs du mal', 'Charles Baudelaire', 1857, FALSE);


-- Insertion de données dans la table `Emprunt`
INSERT INTO Emprunt (idAbonne, idLivre, date_emprunt, date_retour)
VALUES
(1, 1, '2024-05-10', '2024-06-10'),
(2, 2, '2024-06-01', '2024-07-01'),
(3, 3, '2024-06-15', '2024-07-15'),
(4, 4, '2024-07-05', '2024-08-05');
-- Requêtes SQL
-- Recherche d'un abonné par identifiant
SELECT * FROM Abonne WHERE idAbonne = 1;
-- Recherche d'un abonné par nom
SELECT * FROM Abonne WHERE nom = 'Durand';
-- Recherche des livres disponibles
SELECT * FROM Livre WHERE disponible = TRUE;
-- Afficher tous les emprunts en cours (livres empruntés et abonnés)
SELECT e.idEmprunt, a.nom, a.prenom, l.titre, e.date_emprunt, e.date_retour
FROM Emprunt e
JOIN Abonne a ON e.idAbonne = a.idAbonne
JOIN Livre l ON e.idLivre = l.idLivre
WHERE e.date_retour >= CURDATE();
-- Afficher l'historique des emprunts d'un abonné
SELECT e.idEmprunt, l.titre, e.date_emprunt, e.date_retour
FROM Emprunt e
JOIN Livre l ON e.idLivre = l.idLivre
WHERE e.idAbonne = 1;
-- Mise à jour de la disponibilité d'un livre (par exemple, un livre qui est retourné)
UPDATE Livre
```

SET disponible = TRUE

WHERE idLivre = 1;

-- Suppression d'un abonné (lorsqu'un abonné est supprimé, ses emprunts doivent également l'être)

DELETE FROM Abonne WHERE idAbonne = 1;

-- Suppression d'un livre (lorsqu'un livre est supprimé, ses emprunts doivent également l'être)

DELETE FROM Livre WHERE idLivre = 1;

Exigences évaluées:

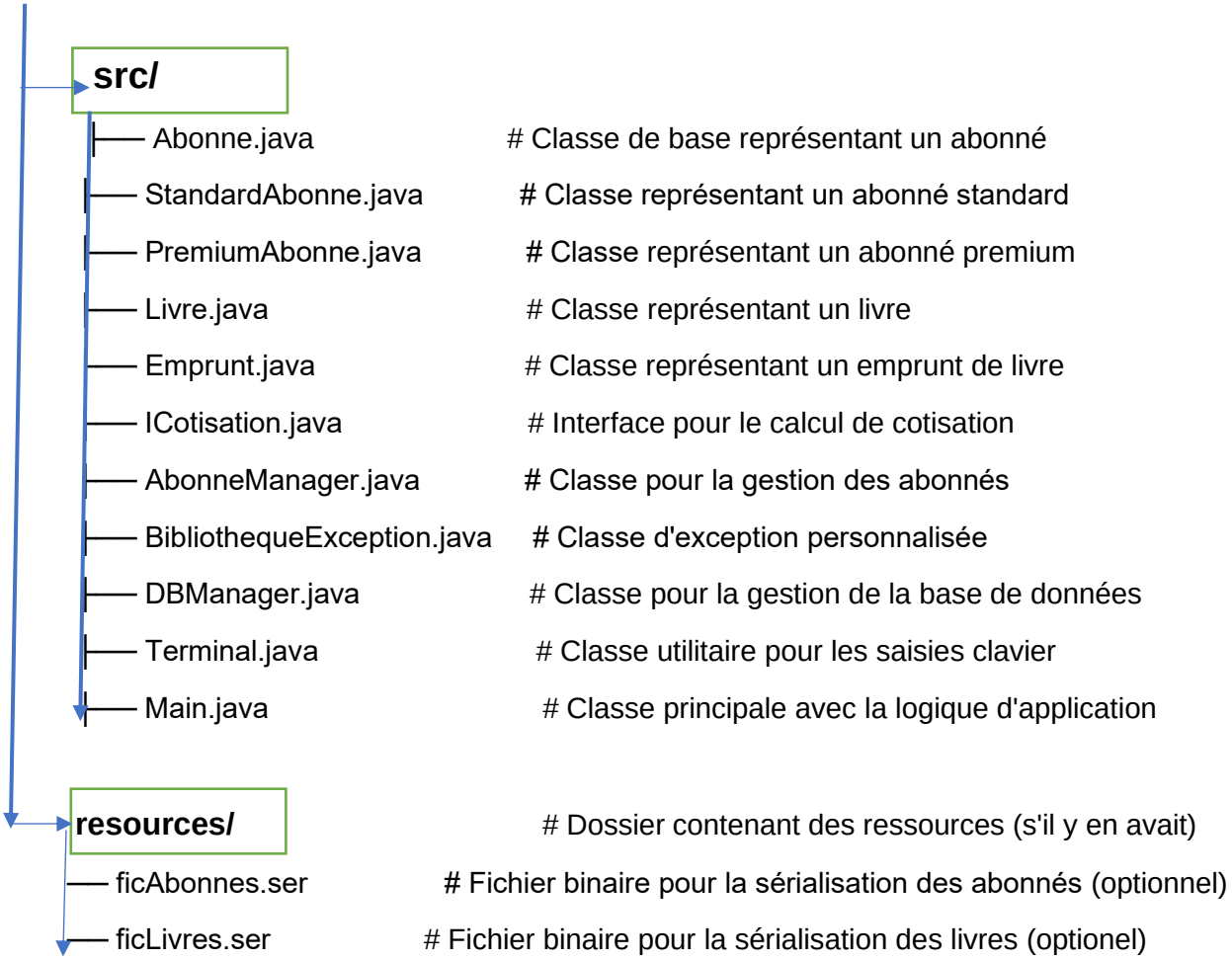
Vous devez élaborer du code Java complet pour implémenter un système de gestion de bibliothèque, conformément à la structure ci-dessous fournie. Le système est structuré autour de plusieurs classes, interfaces et exceptions pour gérer les abonnés, les livres, les emprunts, les cotisations, les sauvegardes et la gestion des erreurs. Chaque partie du système est bien isolée pour répondre aux exigences de votre remise.

Structure de votre Workspace :

- `Abonne.java` : Classe représentant un abonné.
- `Livre.java` : Classe représentant un livre.
- `Emprunt.java` : Classe représentant un emprunt.
- `ICotisation.java` : Interface pour les cotisations.
- `StandardAbonne.java` et `PremiumAbonne.java` : Classes représentant les types d'abonnés.
- `AbonneManager.java` : Classe pour gérer la collection des abonnés.
- `Terminal.java` : Classe utilitaire pour gérer les saisies de l'utilisateur.
- `BibliothequeException.java` : Classe d'exception pour gérer les erreurs liées aux cotisations et aux entrées.
- `DBManager.java` : Classe pour gérer les connexions à la base de données et les requêtes SQL.
- `Main.java` : Application principale pour tester le système.

Votre travail doit structurer les classes nécessaires pour gérer les abonnés, les livres, les emprunts et les cotisations. Les exceptions sont gérées pour les erreurs de cotisation et de validation des données. Le gestionnaire d'abonnés permet d'ajouter, de supprimer et de rechercher des abonnés. Le terminal permet des saisies interactives.

BibliothequeGestion/



Partie II (50 points)

Le jeu dont je vous inclue le code consiste à enregistrer et gérer les scores des joueurs dans une base de données. Chaque joueur peut être identifié par un courriel et doit avoir la possibilité d'ajouter des scores à son actif. Le but est de gérer ces scores en permettant de :

- Ajouter de nouveaux joueurs.

- Ajouter des scores pour chaque joueur.

- Récupérer les scores d'un joueur particulier.

- Calculer le score moyen d'un joueur.

- Afficher les cinq meilleurs scores de tous les joueurs.

Vous serez amené à compléter le code existant, en vous concentrant principalement sur la gestion de la base de données et l'interaction avec celle-ci via JDBC (Java Database Connectivity).

Vous devez compléter et adapter plusieurs parties du code, notamment :

1. Classe `**DatabaseConnection.java**` : Cette classe sera responsable de la gestion de la connexion à la base de données. Vous devrez implémenter la méthode de connexion à une base de données en utilisant JDBC. Le code actuel est une esquisse, et vous devez y ajouter la logique nécessaire pour établir une connexion à une base de données (par exemple, MySQL, PostgreSQL).

2. Classe `**Application.java**` : Cette classe contient un `main` qui utilise le service `**JeuService**`. Vous devrez compléter les appels de méthodes pour :

- Ajouter un utilisateur à la base de données.

- Ajouter des scores pour cet utilisateur.

- Récupérer et afficher les scores d'un utilisateur donné.

- Calculer et afficher le score moyen d'un joueur.

- Afficher les cinq meilleurs scores de tous les joueurs.

Sections à compléter

Question 1 (10%) : Compléter la connexion à la base de données (`**DatabaseConnection.java**`)

La méthode de connexion à la base de données doit être définie. Cela implique généralement l'utilisation de l'API JDBC pour établir une connexion avec un serveur de base de données en utilisant les informations nécessaires (URL, utilisateur, mot de passe).

Dans la méthode `main`, vous devrez compléter selon les questions suivantes :

Question 2 (10%) : Ajouter un utilisateur : Créez un objet `Utilisateur` et ajoutez-le à la base de données via la méthode `**ajouterUtilisateur**` de la classe `**JeuService**`.

Question 3 (10%) : Ajouter un score pour l'utilisateur : Créez un objet `Score` et ajoutez-le pour l'utilisateur via la méthode `**ajouterScore**`.

Question 4 (**10%):** Calculer et afficher le score moyen : Utilisez la méthode `**getScoreMoyen**` de la classe `**JeuService**` pour obtenir la moyenne des scores d'un utilisateur.

Question 5 (10%) : Afficher les cinq meilleurs scores : Appelez la méthode `**getTop5Scores**` pour afficher les cinq meilleurs scores de tous les utilisateurs.

Voici le code de l'application en question et la structure de la base de données

Pour que cet exercice fonctionne correctement, il est important que la base de données soit structurée avec deux tables :

1. Table `utilisateurs` : Contient les informations des joueurs (nom, prénom, email).

- `id` (int, clé primaire)
- `nom` (varchar)
- `prenom` (varchar)
- `email` (varchar)

2. Table `scores` : Contient les scores des joueurs.

- `id` (int, clé primaire)
- `id_utilisateur` (int, clé étrangère référencée à `utilisateurs.id`)
- `score` (int)
- `date_jeu` (timestamp)

Exemple de sortie attendue

Si le code est bien complété, l'exécution du programme pourrait donner une sortie comme celle-ci :

Score : 350 le 2024-11-10 12:34:56

Score : 450 le 2024-11-11 14:23:45

Score moyen : 400.0

Top 5 Scores :

Score : 900 le 2024-11-10 12:12:45

Score : 850 le 2024-11-11 10:10:33

Score : 750 le 2024-11-11 11:11:21

Score : 650 le 2024-11-10 09:09:11

Score : 600 le 2024-11-11 08:08:08

Avant d'implémenter l'application Java, voici les requêtes SQL nécessaires pour créer les tables dans votre base de données MySQL score.db

```
CREATE TABLE utilisateurs (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(100) NOT NULL,
    prenom VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE
);
```

Crée la table des utilisateurs

```
CREATE TABLE scores (
    id INT AUTO_INCREMENT PRIMARY KEY,
    id_utilisateur INT,
    score INT NOT NULL,
    date_jeu DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (id_utilisateur) REFERENCES utilisateurs(id)
);
```

Crée la table des scores

Voici le code java complet du jeux libGDX , vous devez compléter le code manquant.

//Classe de connexion à la base de données (`DatabaseConnection.java`). Cette classe gère la connexion JDBC à la base de données.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseConnection {
        // à compléter
}
```

// Classe `Utilisateur` pour représenter un joueur. Elle contient les informations d'un joueur.

```
public class Utilisateur {
    private int id;
    private String nom;
    private String prenom;
    private String email;

    // Constructeurs, getters et setters
    public Utilisateur(int id, String nom, String prenom, String email) {
        this.id = id;
        this.nom = nom;
        this.prenom = prenom;
        this.email = email;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getNom() {
        return nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
    public String getPrenom() {
        return prenom;
    }
    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }
    public String getEmail() {
      return email;
    }
    public void setEmail(String email) {
        this.email = email;
```

```
    }
}


//Classe `Score` pour représenter un score (score.java)
import java.util.Date;
public class Score {
    private int id;
    private int utilisateurId;
    private int score;
    private Date dateJeu;

    // Constructeurs, getters et setters
    public Score(int id, int utilisateurId, int score, Date dateJeu) {
        this.id = id;
        this.utilisateurId = utilisateurId;
        this.score = score;
        this.dateJeu = dateJeu;
    }

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public int getUtilisateurId() {
        return utilisateurId;
    }
    public void setUtilisateurId(int utilisateurId) {
        this.utilisateurId = utilisateurId;
    }
    public int getScore() {
        return score;
    }
    public void setScore(int score) {
        this.score = score;
    }

    public Date getDateJeu() {
        return dateJeu;
    }

    public void setDateJeu(Date dateJeu) {
        this.dateJeu = dateJeu;
    }
}
//Classe `JeuService` pour la logique de gestion des utilisateurs et des scores(jeuservices.java)
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
public class JeuService {
    // Ajouter un utilisateur
     public void ajouterUtilisateur(Utilisateur utilisateur) throws SQLException {
```

```
    String query = "INSERT INTO utilisateurs (nom, prenom, email) VALUES (?, ?, ?)";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(query)) {

        stmt.setString(1, utilisateur.getNom());
        stmt.setString(2, utilisateur.getPrenom());
        stmt.setString(3, utilisateur.getEmail());
        stmt.executeUpdate();
    }
}

// Ajouter un score pour un utilisateur
public void ajouterScore(Score score) throws SQLException {
    String query = "INSERT INTO scores (id_utilisateur, score, date_jeu) VALUES (?, ?, ?)";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(query)) {
         stmt.setInt(1, score.getUtilisateurId());
        stmt.setInt(2, score.getScore());
        stmt.setTimestamp(3, new Timestamp(score.getDateJeu().getTime()));
        stmt.executeUpdate();
    }
}

// Récupérer tous les scores d'un utilisateur
public List<Score> getScoresByUtilisateur(String email) throws SQLException {
    String query = "SELECT s.id, s.id_utilisateur, s.score, s.date_jeu " +
                   "FROM scores s JOIN utilisateurs u ON s.id_utilisateur = u.id WHERE u.email = ?";
    List<Score> scores = new ArrayList<>();
         try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(query)) {
         stmt.setString(1, email);
        ResultSet rs = stmt.executeQuery();
         while (rs.next()) {
            Score score = new Score(
                rs.getInt("id"),
                rs.getInt("id_utilisateur"),
                rs.getInt("score"),
                rs.getTimestamp("date_jeu")
            );
            scores.add(score);
        }
    }
         return scores;
}

// Calculer le score moyen d'un utilisateur
public double getScoreMoyen(String email) throws SQLException {
    String query = "SELECT AVG(score) AS moyenne FROM scores s " +
                   "JOIN utilisateurs u ON s.id_utilisateur = u.id WHERE u.email = ?";
    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(query)) {
```

```
            stmt.setString(1, email);
            ResultSet rs = stmt.executeQuery();

            if (rs.next()) {
                return rs.getDouble("moyenne");
            }
        }
            return 0;
    }

    // Récupérer les 5 meilleurs scores
    public List<Score> getTop5Scores() throws SQLException {
        String query = "SELECT s.id, s.id_utilisateur, s.score, s.date_jeu " +
                       "FROM scores s ORDER BY s.score DESC LIMIT 5";
        List<Score> topScores = new ArrayList<>();

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(query)) {

            ResultSet rs = stmt.executeQuery();

            while (rs.next()) {
                Score score = new Score(
                    rs.getInt("id"),
                    rs.getInt("id_utilisateur"),
                    rs.getInt("score"),
                    rs.getTimestamp("date_jeu")
                );
                topScores.add(score);
            }
        }

        return topScores;
    }
}
```

// Utilisation de l'application ou classe principale, main.java

```
import java.util.Date;

import java.util.List;

public class Application {

    public static void main(String[] args) {

        JeuService service = new JeuService();

        try {

            // Ajouter un utilisateur

            // à compléter

            // Ajouter des scores pour l'utilisateur
```

```
            // à compléter
        // Récupérer tous les scores d'un utilisateur
        List<Score> scores = service.getScoresByUtilisateur("jean.dupont@example.com");
        for (Score score : scores) {
            System.out.println("Score : " + score.getScore() + " le " + score.getDateJeu());
        }
        // Calculer le score moyen d'un utilisateur
                // à compléter
        // Récupérer les 5 meilleurs scores
                // à compléter
        }
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
}
```

Grille d’évaluation partie 1 (au total 50 points)

Grille d’évaluation (Pour un total de 10 points) : Question 1

Élément de compétence Contexte de réalisation propre à la compétence Assurer la qualité d'une application -017B :					
Critères de rendement	**Très satisfaisant**	**Satisfaisant**	**Insatisfaisant**	**Très insatisfaisant**	**Total**
1.2 Utilisation appropriée des techniques de collecte de l’information.	**10**	**06 – 08**	**01 – 05**	**0**	**/10**

Grille d’évaluation (Pour un total de 10 points) : Question 2

Élément de compétence Contexte de réalisation propre à la compétence Assurer la qualité d'une application -017B :					
Critères de rendement	**Très satisfaisant**	**Satisfaisant**	**Insatisfaisant**	**Très insatisfaisant**	**Total**
4.1 Modélisation appropriée des données.	**10**	**06 – 08**	**01 – 05**	**0**	**/10**

Grille d’évaluation (Pour un total de 10 points) : Question 3 et Question 4

Élément de compétence Contexte de réalisation propre à la compétence Assurer la qualité d'une application -017B :					
Critères de rendement	**Très satisfaisant**	**Satisfaisant**	**Insatisfaisant**	**Très insatisfaisant**	**Total**
.5 Création correcte des tables.	**10**	**06 – 08**	**01 – 05**	**0**	**/10**

Grille d’évaluation (Pour un total de 10 points) : Question 5

Élément de compétence Contexte de réalisation propre à la compétence Assurer la qualité d'une application -017B :					
Critères de rendement	**Très satisfaisant**	**Satisfaisant**	**Insatisfaisant**	**Très insatisfaisant**	**Total**
2.3 Justification des choix en fonction des	**10**	**06 – 08**	**01 – 05**	**0**	**/10**

priorités et des exigences.					

Grille d'évaluation partie 2 (au total 50 points)

Grille d'évaluation (Pour un total de 10 points) : Question 1

Élément de compétence Contexte de réalisation propre à la compétence Assurer la qualité d'une application -017B :					
Critères de rendement	**Très satisfaisant**	**Satisfaisant**	**Insatisfaisant**	**Très insatisfaisant**	**Total**
4.5 Création correcte des tables.	**10**	**06 – 08**	**01 – 05**	**0**	**/10**

Grille d'évaluation (Pour un total de 10 points) : Question 2 et Question 3

Élément de compétence Contexte de réalisation propre à la compétence Assurer la qualité d'une application -017B :					
Critères de rendement	**Très satisfaisant**	**Satisfaisant**	**Insatisfaisant**	**Très insatisfaisant**	**Total**
1.5 Représentation globale des fonctionnalités.	**10**	**06 – 08**	**01 – 05**	**0**	**/10**

Grille d'évaluation (Pour un total de 10 points) : Question 4 et Question 5

Élément de compétence Contexte de réalisation propre à la compétence Assurer la qualité d'une application -017B :					
Critères de rendement	**Très satisfaisant**	**Satisfaisant**	**Insatisfaisant**	**Très insatisfaisant**	**Total**
3.3 Appropriation de l'environnement de développement de l'application.	**10**	**06 – 08**	**01 – 05**	**0**	**/10**

Grille d'évaluation (Pour un total de 10 points) : Question 5

Élément de compétence Contexte de réalisation propre à la compétence Assurer la qualité d'une application -017B :					
Critères de rendement	**Très satisfaisant**	**Satisfaisant**	**Insatisfaisant**	**Très insatisfaisant**	**Total**

2.3 Justification des choix en fonction des priorités et des exigences.	**10**	**06 – 08**	**01 – 05**	**0**	**/10**

Grille de correction de la langue

Communication claire	Communication le plus souvent claire	Communication floue	Communication difficile
-0	-0,5	-1,5	-2
Utilisation d'un vocabulaire précis et varié	Utilisation d'un vocabulaire précis	Utilisation d'un vocabulaire imprécis	Utilisation d'un vocabulaire inapproprié
-0	-0,5	-1,5	-2
Respect des normes (type de travail)	Respect de la plupart des normes (type de travail)	Non-respect des normes (type de travail)	Situation de communication inappropriée (type de travail)
-0	-0,5	-1,5	-2
Respect du code linguistique (≤2 fautes /page)	Respect de la plupart des règles du code linguistique (3-7 fautes /page)	Respect de la plupart des règles du code linguistique (8-10 fautes /page)	Respect de la plupart des règles du code linguistique (>10 fautes /page)
-0	-0,5 -2.5	-2.5 – 3.5	-4