Projet Base de Données de Films – Travail Étudiant

Cours : Programmation Orientée Objet (POO 1)
Classe : ING 2 Section A
Enseignante : Djeraoui Mouna
Date de remise : …
Note totale : 15 points

1. Aperçu du projet

Objectif :

Les étudiants doivent réaliser une application interactive de **Base de Données Média** permettant à l'utilisateur de :

- Ajouter différents types de médias (films, séries TV, documentaires) à sa collection
- Rechercher et filtrer sa collection
- Noter les médias et les marquer comme visionnés
- Afficher des statistiques sur sa collection

Objectifs pédagogiques :

- Appliquer les concepts de la Programmation Orientée Objet (POO)
- Pratiquer l'héritage et le polymorphisme
- Implémenter la surcharge de méthodes
- Créer une application console interactive
- Utiliser correctement les techniques d'encapsulation

2. Architecture du système

Le système est organisé en trois couches conceptuelles :

a. **Couche Modèles** : Contient les classes représentant les différents types de médias.

b. **Couche Services** : Fournit les fonctionnalités de gestion de la collection de médias.

c. **Couche Interface Utilisateur (UI)** : Gère l'interaction avec l'utilisateur via une interface console.

Relations entre les classes :

- Media est la classe de base.
- Movie, Series et Documentary héritent de Media → démonstration du **polymorphisme**.
- Series contient une liste d'objets Episode.
- MediaDatabase gère une collection d'objets Media et fournit les fonctionnalités de recherche, filtrage et statistiques.
- Application interagit avec l'utilisateur, appelle les méthodes de MediaDatabase et affiche les résultats.

3. Spécifications des classes

a. Media.java (Classe de base)

Rôle : Représente une entrée média générique.

Attributs (tous private) :

- title (String)
- director (String)
- releaseYear (int)
- genre (String)
- rating (double)
- isWatched (boolean)

Constructeurs (surchargés) :

- Media(title, director, releaseYear)
- Media(title, director, releaseYear, genre)

Méthodes :

- Getters et setters pour tous les attributs
- markAsWatched()
- getMediaType()
- displayInfo()
- displayInfo(boolean showRating)

b. Movie.java (Sous-classe de Media)

Constructeur :

- Movie(title, director, releaseYear, genre)

Méthodes :

- Redéfinition de getMediaType() → retourne "Film"
- Redéfinition de displayInfo() et displayInfo(boolean showRating)

c. Series.java (Sous-classe de Media)

Attributs supplémentaires :

- episodes (List)
- nbOfSeasons (int)

Constructeur :

- Series(title, director, releaseYear, genre, episodes, nbOfSeasons)

Méthodes :

- Redéfinition de getMediaType() → retourne "Série TV"
- getTotalEpisodes()
- displaySeriesInfo()
- displaySeriesInfo(boolean showTotalEpisodes)

d. Episode.java (Classe associée à Series)

Attributs (tous private) :

- title (String)
- duration (int) – en minutes
- episodeNumber (int)
- seasonNumber (int)

Constructeur :

- Episode(title, duration, episodeNumber, seasonNumber)

Méthodes :

- Getters et setters
- displayEpisodeInfo()

e. Documentary.java (Sous-classe de Media)

Attributs supplémentaires :

- topic (String)
- isEducational (boolean)

Constructeur :

- Documentary(title, director, releaseYear, topic, isEducational)

Méthodes :

- Redéfinition de getMediaType() → retourne "Documentaire"
- getEducationalValue()

f. MediaDatabase.java (Classe de service)

Attributs :

- mediaList (ArrayList)

Méthodes principales :

Ajout (surchargées) :

- addMedia(Media media)
- addMedia(title, director, year) – crée et ajoute le média

Recherche (surchargées) :

- searchByTitle(String title)
- searchByDirector(String director)
- searchByGenre(String genre)

Gestion :

- markAsWatched(String title)
- rateMedia(String title, double rating)
- getUnwatchedMedia()
- getWatchedMedia()
- displayAllMedia()

Statistiques :

- getAverageRating()

Calcule et retourne la note moyenne de tous les médias dans la collection. Cette méthode parcourt la liste des médias, additionne toutes les notes, divise par le nombre total de médias et retourne la moyenne.

- getMediaCount()

Retourne simplement le nombre total de médias stockés dans la base de données. C'est l'équivalent de mediaList.size().

- getMediaByType(String type)

Retourne une liste de tous les médias d'un type spécifique (ex: "Film", "Série TV", "Documentaire"). Cette méthode parcourt la collection, vérifie le type de chaque média (via getMediaType()), et collecte ceux qui correspondent au type demandé.

g. Application.java (Classe d'interface console)

Attributs :

- database (MediaDatabase)
- scanner (Scanner)
- isRunning (boolean)

Responsabilités :

- Afficher le menu
- Gérer la saisie utilisateur et la validation
- Appeler les méthodes appropriées de MediaDatabase
- Afficher les résultats

Fonctionnalités interactives :

- Ajout de médias (Film, Série, Documentaire)
- Recherche
- Marquer comme visionné
- Noter un média
- Afficher les statistiques et les listes visionnées/non visionnées

4. Exigences d'implémentation

Phase 1 – Classes de base :

- Implémenter Media avec encapsulation complète
- Implémenter Movie, Documentary, Series, Episode
- Tester l'héritage, le polymorphisme et la surcharge de méthodes
- Implémenter la classe MediaDatabase

Phase 2 – Application interactive :

- Créer la classe Application avec interface console basée sur Scanner
- Implémenter toutes les options du menu
- Ajouter la validation des saisies
- Tester le flux complet utilisateur

Phase 3 – Améliorations (Optionnelles) :

- Ajouter la persistance dans des fichiers (sauvegarde/chargement)
- Ajouter des filtres supplémentaires, tri, et statistiques avancées

Projet de Base de Données de Formule 1 – Devoir Étudiant

Cours : Programmation Orientée Objet (POO 1)
Classe : ING 2 Section A
Enseignante : Djeraoui Mouna
Date de Rendu :
Total des Points : 15 Points

1. Aperçu du Projet

- **Objectif** :

Les étudiants ont pour tâche de créer une application interactive de Base de Données de Formule 1 où les utilisateurs peuvent :

  - Ajouter différents types d'entités de F1 (pilotes, équipes, courses) à leur collection
  - Rechercher et filtrer les entités selon divers critères (ex. : nom du pilote, équipe, année de la course)
  - Enregistrer les résultats des courses et les statistiques des pilotes/équipes
  - Consulter les statistiques et les classements

- **Objectifs d'Apprentissage** :
  - Appliquer les concepts de Programmation Orientée Objet (POO)
  - Pratiquer l'héritage et le polymorphisme
  - Implémenter la surcharge de méthodes
  - Créer une application console interactive
  - Utiliser des techniques d'encapsulation appropriées

2. Architecture du Système

Le système est organisé selon les couches conceptuelles suivantes :

a. **Couche Modèles** : Contient les classes représentant les différentes entités de Formule 1.

b. **Couche Services** : Fournit les fonctionnalités de gestion pour la base de données F1.

c. **Couche Interface Utilisateur (UI)** : Gère l'interaction avec l'utilisateur via une interface console.

Relations entre les Classes :

a. **F1Entity** est la classe de base.

b. **Driver**, **Team** et **Race** héritent de **F1Entity**, illustrant le polymorphisme.

c. **Driver** contient une liste d'objets **RaceResult**.

d. **Race** contient une liste d'objets **RaceResult**.

e. **F1Database** gère les collections de **Driver**, **Team** et **Race**.

f. **Application** interagit avec l'utilisateur, appelle les méthodes de **F1Database** et affiche les résultats.

3. Spécifications des Classes

a. F1Entity.java (Classe de Base)

Objectif : Représente une entité générique de Formule 1.
Attributs (tous privés) :

- name (String)
- country (String)
- activeYears (String)

Constructeurs (Surchargés) :

- F1Entity(name, country)
- F1Entity(name, country, activeYears)

Méthodes :

- Getters et setters pour tous les attributs
- getEntityType() → Retourne 'Entité F1 Générique' (à surcharger)
- displayInfo()

b. Driver.java (Sous-classe de F1Entity)
Objectif : Représente un pilote de Formule 1.
Attributs Supplémentaires :

- team (Team)
- number (int)
- raceResults (List<RaceResult>)

Méthodes :

- getEntityType()
- addRaceResult(RaceResult result)
- getTotalPoints()
- displayDriverInfo(boolean showResults)

c. Team.java (Sous-classe de F1Entity)
Objectif : Représente une équipe de Formule 1.
Attributs Supplémentaires :

- drivers (List<Driver>)
- championshipPoints (int)

Méthodes :

- getEntityType()
- addDriver(Driver driver)
- calculateTeamPoints()
- displayTeamInfo()

d. Race.java (Sous-classe de F1Entity)
Objectif : Représente une course de Formule 1.
Attributs Supplémentaires :

- year (int)
- location (String)
- raceResults (List<RaceResult>)

Méthodes :

- Surcharger getEntityType() → 'Course'
- addRaceResult(RaceResult result)
- displayRaceResults()

e. RaceResult.java (Associé à Race et Driver)
Objectif : Représente le résultat d'un pilote dans une course.
Attributs (tous privés) :
- driver (Driver)
- team (Team)
- position (int)
- qualifyingPosition (int)
- points (int)

Méthodes :
- Getters et setters
- displayResult()

f. F1Database.java (Classe de Service)
Objectif : Gère les collections d'entités F1.
Attributs :
- drivers (List<Driver>)
- teams (List<Team>)
- races (List<Race>)

Méthodes :
- Méthodes d'ajout (surchargées) pour Driver, Team, Race
- Méthodes de recherche par nom, pays ou année
- Obtenir les classements des pilotes et des équipes
- Afficher toutes les entités
- Enregistrer les résultats de courses et calculer les statistiques

g. Application.java (Classe UI/Interactive)
Objectif : Fournit une interface console pour l'utilisateur.
Attributs :
- database (F1Database)
- scanner (Scanner)
- isRunning (boolean)

Options du Menu Principal :
- Ajouter un nouveau pilote/équipe/course
- Rechercher des entités
- Voir tous les pilotes
- Voir toutes les équipes
- Voir toutes les courses

- Enregistrer les résultats d'une course
- Voir le classement des pilotes
- Voir le classement des équipes
- Quitter

4. Exigences d'Implémentation

Phase 1 – Classes de Base :

- Implémenter F1Entity avec une encapsulation complète
- Implémenter Driver, Team, Race, et RaceResult
- Tester l'héritage, le polymorphisme et la surcharge de méthodes
- Implémenter la classe F1Database

Phase 2 – Application Interactive :

- Créer la classe Application avec une interface utilisateur basée sur Scanner
- Implémenter toutes les options du menu
- Ajouter la validation des entrées
- Tester le flux utilisateur complet

Phase 3 – Améliorations (Optionnel) :

- Ajouter la persistance des données (sauvegarder/charger les données)
- Ajouter des filtres supplémentaires, du tri et des statistiques avancées
-

=== MENU BASE DE DONNÉES F1 ===
Entrez votre choix : 1
Que souhaitez-vous ajouter ?
1. Pilote
2. Équipe
3. Course
Entrez votre choix : 1
Entrez le nom du pilote : Lewis Hamilton
Entrez le pays : Royaume-Uni
Entrez le numéro du pilote : 44
Sélectionnez l'équipe : Mercedes
Pilote ajouté avec succès !

1. Listes en Java : L'Explication Simple

Considérez une **ArrayList** comme une **liste de courses redimensionnable** en Java.
C'est un conteneur qui peut contenir plusieurs objets (comme des objets Driver, Team ou Race dans votre projet), et qui grandit automatiquement lorsque vous ajoutez plus d'éléments.

a. Déclaration de Base :

```
java
import java.util.ArrayList;import java.util.List; // Créer une liste qui
contient des objets DriverList<Driver> drivers = new ArrayList<>();
```

Méthodes Essentielles :

- add(element) → Ajouter un élément à la fin : drivers.add(new Driver("Lewis Hamilton"));
- get(index) → Obtenir un élément par position (0=premier) : Driver d = drivers.get(0);
- remove(index) → Supprimer par position : drivers.remove(0);
- size() → Obtenir le nombre d'éléments : int count = drivers.size();
- isEmpty() → Vérifier si vide : if (drivers.isEmpty()) { ... }

Boucle Simple :

```
java
// Afficher tous les pilotesfor (Driver driver : drivers)
{    System.out.println(driver.getName());}
```

1. Installer Git sur votre PC

Étape 1 : Téléchargez Git depuis https://git-scm.com/downloads
Étape 2 : Installez Git

- Exécutez l'installateur et sélectionnez les options par défaut.
- Important : Choisissez "Utiliser Git depuis l'invite de commande Windows".
- Gardez VS Code comme éditeur par défaut si disponible.

Étape 3 : Vérifiez l'installation
Ouvrez un terminal et exécutez :

```
git –version
```

Vous devriez voir un numéro de version.

2. Créer un Compte GitHub

Étape 1 : Allez sur https://github.com et inscrivez-vous.
Étape 2 : Entrez votre email, nom d'utilisateur et mot de passe. Vérifiez votre email.
Étape 3 : Optionnel : configurez les clés SSH (nous utiliserons HTTPS pour plus de simplicité).

3. Lier GitHub à VS Code

Étape 1 : Installez VS Code depuis https://code.visualstudio.com
Étape 2 : Installez l'extension GitHub

- Ouvrez Extensions (Ctrl+Shift+X) et installez "GitHub Pull Requests and Issues".

Étape 3 : Connectez-vous à GitHub dans VS Code

- Ouvrez la Palette de Commandes (Ctrl+Shift+P) → GitHub: Sign in → Suivez les instructions.

4. Configurer Git Localement

```
git config --global user.name "Votre Nom"git config --global user.email "votre.email@exemple.com"
```

Vérifiez la configuration :

```
git config –list
```

5. Créer un Dépôt sur GitHub

Étape 1 : Allez sur GitHub → Nouveau Dépôt

- Nom : nom-du-projet
- Définir la visibilité du projet sur **Privé**
- Optionnel : Initialiser avec un README

Étape 2 : Copiez l'URL HTTPS (exemple : https://github.com/username/nom-du-projet.git)

6. Connecter le Projet Local à GitHub

Projet Existant :

```
git initgit add .git commit -m "Premier commit"git remote add origin https://github.com/username/nom-du-projet.gitgit branch -M maingit push -u origin main
```

7. Travailler sur le Projet

1. Effectuez des modifications dans VS Code.
2. Ajoutez et validez les changements :

```
git add .git commit -m "Décrivez les changements"
```

3. Poussez vers GitHub :

```
git push origin main
```

4. Tirez les derniers changements si vous collaborez :

```
git pull origin main
```

8. Soumission du Travail

1. Allez dans les paramètres de votre dépôt.
2. Collaborateurs.
3. Ajoutez les personnes, moi-même (https://github.com/MonaMo02) et Dr. Mekahlia (https://github.com/Fmekahlia) en tant que collaborateurs.