Build & Run

Commencer par lancer le serveur : Dans le dossier cpp, lancer : `make` pour compiler puis `make run` (Parfois il faut supprimer le fichier depend-myprog avant.
On peut faire ceci soit manuellement soit via la ligne de commande en tapant `rm depend-myprog` puis `make clean`)

Le serveur se mettra en écoute sur le port 3331

Puis lancer le client swing : Dans le dossier swing, lancer : make run

Partie C++

Etape 4 :

- **Comment appelle-t-on ce type de méthode ?**

→ Une méthode **virtuelle pure**.

- **Comment faut-il les déclarer ?**

→ En ajoutant = 0 à la déclaration dans la classe de base et en la redéfinissant dans les classes dérivées.

Étape 5 : Traitement générique (en utilisant le polymorphisme)

- **Quelle est la propriété caractéristique de l'orienté objet qui permet de faire cela ?**

→ **Le polymorphisme**.

- **Qu'est-il spécifiquement nécessaire de faire dans le cas du C++ ?**

→ Déclarer les méthodes comme **virtuelles** dans la classe de base.

- **Quel est le type des éléments du tableau : le tableau doit-il contenir des objets ou des pointeurs vers ces objets ? Pourquoi ?**

→ Le tableau doit contenir des **pointeurs vers des objets** pour permettre **le polymorphisme** et éviter le **slicing**.

- **Comparer à Java.**

→ En **Java**, tous les objets sont manipulés via des **références**, donc le problème du slicing ne se pose pas.

Question :
Pourquoi n'est-il plus possible d'instancier des objets de la classe de base ?

Réponse :
Dans notre projet, la classe `BaseMultimedia` (ou `Multimedia`) contient des **méthodes virtuelles pures**, comme `play()`. Cela signifie qu'elle ne fournit pas d'implémentation concrète pour ces méthodes. Une classe contenant au moins une méthode virtuelle pure est une **classe abstraite** et **ne peut pas être instanciée directement**.

Conséquence :
On doit forcément passer par une classe dérivée (`Photo`, `Video`, `Film`, etc.) qui **implémente** ces méthodes.

Étape 6 : Films et tableaux

Question :
Que faut-il faire pour que l'objet `Film` ait plein contrôle sur ses données et que son tableau de durées des chapitres ne puisse pas être modifié ou détruit à son insu ?

Réponse :

- Lorsqu'un tableau de durées des chapitres est fourni à `Film`, **il ne faut pas simplement stocker le pointeur** mais **copier les données** dans un nouveau tableau dynamique.
- L'objet `Film` devient ainsi **propriétaire** des données et peut les gérer sans dépendre de l'extérieur.
- Pour éviter toute modification externe, l'accesseur doit retourner un **pointeur constant (**`const int*`**)** ou un **chapitre spécifique via une fonction** `getChapter(int index)`.
- Un **destructeur** doit être ajouté pour libérer la mémoire lorsque l'objet `Film` est détruit.

Étape 7 : Destruction et copie des objets

Question :
Quelles classes doivent être modifiées pour éviter les fuites mémoire ?

Réponse :
La classe `Film` utilise un tableau dynamique (`int* chapters`) qui est **alloué avec** `new`. Si cet objet est détruit sans libérer cette mémoire (`delete[]`), une **fuite mémoire** se produit.

Solution :

- Ajouter un **destructeur** dans `Film` pour **libérer la mémoire** (`delete[] chapters`).
- Empêcher la **copie d'un** `Film` pour éviter des **problèmes de double** `delete`

Étape 8 : Création des groupes

Explication :

- Un **groupe** est une liste d'objets multimédias (`Photo`, `Video`, etc.).
- On ne peut pas stocker directement des objets `BaseMultimedia` car c'est une **classe abstraite**.
- La liste doit donc **contenir des pointeurs (**`std::shared_ptr<BaseMultimedia>`**)** pour permettre le polymorphisme.

Étape 9 : Gestion automatique de la mémoire

Explication :

- `new` et `delete` doivent être évités autant que possible pour **réduire les risques de fuites mémoire**.
- **Solution :** Utiliser `std::shared_ptr` pour stocker les objets multimédias dans `Database`.
- Un objet sera **automatiquement détruit** dès qu'il ne sera plus référencé par aucun groupe.
- Pour **éviter l'instanciation manuelle des objets**, on peut **rendre le constructeur privé** et utiliser une **fabrique (**`createPhoto()`**,** `createVideo()`**, etc.)**.

Étape 10 : Gestion cohérente des données

Explication :

- Une classe `Database` centralise la gestion des objets multimédias.
- Elle permet de **rechercher, ajouter, supprimer et lister** des objets sans exposer les détails internes.
- Un **map associatif (**`std::map<std::string, std::shared_ptr<BaseMultimedia>>`**)** permet d'accéder aux objets par leur nom.

Étape 11 : Client / Serveur

Explication :

Le serveur C++ est chargé de **recevoir** les requêtes du client (Java Swing), de les **analyser**, puis d'exécuter les actions correspondantes sur la base de données (`Database`). Il est basé sur **TCP/IP** et utilise des **sockets** pour la communication.

`server.cpp` : Gestion de la connexion et des requêtes

- Le serveur démarre en créant un **objet** `TCPServer`, qui écoute sur un **port spécifique** (`3331`).
- Lorsqu'une **requête est reçue**, elle est analysée et **traitée directement dans une fonction lambda**.
- **Chaque commande est interprétée** et exécute une méthode de `Database` pour effectuer l'action demandée.
- La **réponse** est ensuite envoyée au client via la **socket TCP**.

Partie Client (Java Swing)

Le **client Java** joue le rôle d'une **interface graphique** permettant d'envoyer des requêtes au serveur et d'afficher les réponses.

Fonctionnement du client

- **Connexion** au serveur via un **socket TCP** (MediaClientGUI`.java`).
- **Envoi de commandes** sous forme de texte (`displayMedia`, `playMedia`, etc.).
- **Réception des réponses du serveur**, qui sont affichées dans l'interface Swing.
- **Boutons et menus interactifs** pour exécuter des actions sans taper de commande manuellement.

Synchronisation entre le Client et le Serveur

- **Le serveur doit être lancé avant le client** (`make run` côté C++).
- **Le client Swing** envoie une requête sous forme de chaîne (`String`).
- **Le serveur interprète la requête**, exécute l'action et renvoie une réponse.
- **Le client affiche la réponse** dans la zone de texte Swing.

Avec cette structure, **le serveur C++ et le client Java peuvent interagir en temps réel** pour gérer les objets multimédias de manière fluide.