project-root/

├── common/

│ ├── model/

│ │ ├── Movie.java

│ │ ├── Coordinates.java

│ │ ├── Person.java

│ │ └── MovieGenre.java

│ └── exchange/

│ ├── CommandRequest.java

│ └── CommandResponse.java

│

├── client/

│ ├── core/

│ │ ├── ClientMain.java

│ │ └── ClientApp.java

│ ├── input/

│ │ ├── MovieInputParser.java

│ │ ├── PersonInputParser.java

│ │ ├── CoordinatesInputParser.java

│ │ └── basic/...

│ └── network/

│ ├── ServerConnection.java

│ ├── RequestSender.java

│ └── ResponseReceiver.java

│

├── server/

│ ├── core/

│ │ ├── ServerMain.java

│ │ └── ServerApp.java

│ ├── commands/

│ │ ├── Command.java

│ │ ├── AddCommand.java

│ │ └── ...

│ ├── collection/

│ │ ├── MovieCollection.java

│ │ └── IdGenerator.java

│ ├── storage/

│ │ └── JsonFileManager.java

│ └── network/

│ ├── ConnectionHandler.java

│ ├── RequestReader.java

│ ├── CommandExecutor.java

│ └── ResponseSender.java

import client.core.MovieCollectionManager;  
  
*/\*\*  
 \* Главная точка входа в программу. Запускает менеджер коллекции фильмов.  
 \*/*public class Main {  
 */\*\*  
 \* Метод main — запускает приложение.  
 \* @param args аргумент командной строки, имя файла с коллекцией  
 \*/* public static void main(String[] args) {  
 if (args.length < 1) {  
 System.*out*.println("Укажите имя файла.");  
 return;  
 }  
 new MovieCollectionManager(args[0]).run();  
 }  
}

package client.core;  
  
import server.collection.MovieCollection;  
import server.commands.Command;  
import server.commands.CommandFactory;  
import server.storage.JsonFileManager;  
import java.util.Scanner;  
  
*/\*\*  
 \* Основной управляющий класс приложения.  
 \* Отвечает за загрузку коллекции фильмов, инициализацию команд,  
 \* и запуск командного цикла для взаимодействия с пользователем.  
 \*/*public class MovieCollectionManager {  
 private final MovieCollection collection;  
 private final CommandFactory commandFactory;  
 private final Scanner scanner;  
 private final JsonFileManager fileManager;  
  
 */\*\*  
 \* Создаёт экземпляр менеджера и инициализирует компоненты приложения.  
 \*  
 \* @param fileName имя JSON-файла, содержащего коллекцию  
 \*/* public MovieCollectionManager(String fileName) {  
 this.collection = new MovieCollection();  
 this.scanner = new Scanner(System.*in*);  
 this.fileManager = new JsonFileManager();  
 this.commandFactory = new CommandFactory(collection, fileManager, fileName, scanner);  
  
 }  
  
 */\*\*  
 \* Запускает главный цикл командной строки для управления коллекцией.  
 \* Обрабатывает пользовательский ввод и выполняет соответствующие команды.  
 \* Завершение возможно через Ctrl+D или команду exit.  
 \*/* public void run() {  
 System.*out*.println("Программа управления коллекцией фильмов");  
 System.*out*.println("Введите 'help' для списка команд");  
  
 while (true) {  
 try {  
 System.*out*.print("> ");  
  
 if (!scanner.hasNextLine()) {  
 System.*out*.println("\nВвод завершён. Завершение программы.");  
 break;  
 }  
  
 String input = scanner.nextLine().trim();  
 if (input.isEmpty()) continue;  
  
 String[] parts = input.split(" ", 2);  
 Command cmd = commandFactory.getCommand(parts[0]);  
  
 if (cmd != null) {  
 cmd.execute(parts.length > 1 ? new String[]{parts[1]} : new String[]{});  
 } else {  
 System.*out*.println("Неизвестная команда. Введите 'help' для справки.");  
 }  
 } catch (Exception e) {  
 System.*out*.println("Ошибка: " + e.getMessage());  
 }  
 }  
 }  
}

package server.commands;  
  
*/\*\*  
 \* Команда завершения работы программы без сохранения.  
 \*/*public class ExitCommand implements Command {  
 */\*\*  
 \* Выполняет команду: завершает выполнение программы.  
 \* @param args не используется  
 \*/* @Override  
 public void execute(String[] args) {  
 System.*out*.println("Завершение программы.");  
 System.*exit*(0);  
 }  
  
 */\*\*  
 \* @return описание команды  
 \*/* @Override  
 public String getDescription() {  
 return "завершить программу (без сохранения)";  
 }  
}

package server.commands;  
  
import server.collection.MovieCollection;  
import server.storage.JsonFileManager;  
  
import java.io.IOException;  
  
*/\*\*  
 \* Команда сохранения коллекции фильмов в файл.  
 \*/*public class SaveCommand implements Command {  
 private final MovieCollection collection;  
 private final JsonFileManager fileManager;  
 private final String fileName;  
  
 */\*\*  
 \* Конструктор.  
 \* @param collection коллекция фильмов  
 \* @param fileManager менеджер для сохранения в файл  
 \* @param fileName имя файла для сохранения  
 \*/* public SaveCommand(MovieCollection collection, JsonFileManager fileManager, String fileName) {  
 this.collection = collection;  
 this.fileManager = fileManager;  
 this.fileName = fileName;  
 }  
  
 */\*\*  
 \* Сохраняет коллекцию в JSON-файл.  
 \*  
 \* @param args не используются  
 \* @param data не используется  
 \* @return сообщение об успешном или неудачном сохранении  
 \*/* @Override  
 public Object execute(String[] args, Object data) {  
 try {  
 fileManager.saveToFile(fileName, collection.getMovies());  
 return "Коллекция успешно сохранена.";  
 } catch (IOException e) {  
 return "Ошибка при сохранении: " + e.getMessage();  
 }  
 }  
  
  
 */\*\*  
 \* @return описание команды  
 \*/* @Override  
 public String getDescription() {  
 return "сохранить коллекцию в файл";  
 }  
}

package server.commands;  
  
import java.io.File;  
import java.io.FileNotFoundException;  
import java.util.\*;  
  
*/\*\*  
 \* Команда для выполнения команд из указанного файла (скрипта).  
 \*/*public class ExecuteScriptCommand implements Command {  
 private final CommandFactory commandFactory;  
 private final Set<String> executingScripts = new HashSet<>();  
  
 */\*\*  
 \* Конструктор.  
 \* @param commandFactory фабрика команд для создания нужных экземпляров  
 \*/* public ExecuteScriptCommand(CommandFactory commandFactory) {  
 this.commandFactory = commandFactory;  
 }  
  
 */\*\*  
 \* Поменять под сервер!!!!!!!!!!  
 \*/* @Override  
 public void execute(String[] args) {  
 if (args.length < 1) {  
 System.*out*.println("Не указано имя файла скрипта");  
 return;  
 }  
  
 String scriptName = args[0];  
  
 if (executingScripts.contains(scriptName)) {  
 System.*out*.println("Скрипт '" + scriptName + "' уже выполняется. Рекурсивный вызов запрещён.");  
 return;  
 }  
  
 File scriptFile = new File(scriptName);  
 if (!scriptFile.exists() || !scriptFile.isFile()) {  
 System.*out*.println("Файл скрипта не найден: " + scriptName);  
 return;  
 }  
  
 executingScripts.add(scriptName);  
  
 try (Scanner fileScanner = new Scanner(scriptFile)) {  
 while (fileScanner.hasNextLine()) {  
 String line = fileScanner.nextLine().trim();  
 if (line.isEmpty() || line.startsWith("#")) continue;  
  
 System.*out*.println("Выполнение команды: " + line);  
 String[] parts = line.split(" ", 2);  
 String commandName = parts[0];  
 String[] commandArgs = parts.length > 1 ? new String[]{parts[1]} : new String[]{};  
  
 Command command = commandFactory.getCommandWithCustomScanner(commandName, fileScanner);  
 if (command != null) {  
 try {  
 command.execute(commandArgs);  
 } catch (Exception e) {  
 System.*out*.println("Ошибка при выполнении команды '" + commandName + "': " + e.getMessage());  
 }  
 } else {  
 System.*out*.println("Неизвестная команда: " + commandName);  
 }  
 }  
 } catch (FileNotFoundException e) {  
 System.*out*.println("Файл скрипта не найден: " + scriptName);  
 } finally {  
 executingScripts.remove(scriptName);  
 }  
 }  
  
 */\*\*  
 \* @return описание команды  
 \*/* @Override  
 public String getDescription() {  
 return "считать и исполнить скрипт из указанного файла";  
 }  
}