Київський нацiональний унiверситет iменi Тараса Шевченка Факультет комп’ютерних наук та кiбернетики

**ЛАБОРАТОРНА РОБОТА №8**

«Побудова клієнт-серверної архітектури на базі протоколу TCP / IP. Клієнт-серверна взаємодія на основі технології віддаленого виклику процедур»

Виконала:

Студентка групи ІПС-31

Ящук Ангеліа

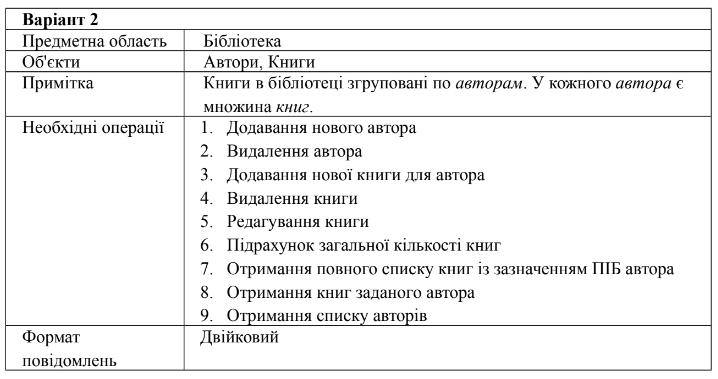
Київ, 2023

**ЛАБОРАТОРНА РОБОТА №8**

**«Побудова клієнт-серверної архітектури на базі протоколу TCP / IP»**

**Мета:** на основі програми, розробленої в рамках лабораторної роботи No2, створити серверну програму, що забезпечує виконання віддалених запитів з управління об'єктами. Відомості про об'єкти повинні зберігатися в базі даних. Розробити клієнтську програму, що відправляє серверу запити на введення, редагування і отримання інформації про об'єкти. Взаємодія між клієнтом і сервером має здійснюватися по протоколу TCP / IP.

**Варіант завдання:**



**Лістинг розробленої програми**

package org.labs;  
  
import java.io.\*;  
import java.net.\*;  
import java.sql.\*;  
  
public class ClientHandler extends Thread {  
 private Socket clientSocket;  
 private PrintWriter writer;  
 private BufferedReader reader;  
  
 private Connection connection;  
  
 public ClientHandler(Socket socket, Connection connection) {  
 this.clientSocket = socket;  
 this.connection = connection;  
 try {  
 writer = new PrintWriter(socket.getOutputStream(), true);  
 reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 @Override  
 public void run() {  
 try {  
 String request = reader.readLine();  
  
 String response = processRequest(request);  
 writer.println(response);  
  
 reader.close();  
 writer.close();  
 clientSocket.close();  
  
 } catch (IOException | SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private String processRequest(String request) throws SQLException {  
 String[] parts = request.split(",");  
 String operation = parts[0];  
  
 switch (operation) {  
 case "ADD\_AUTHOR":  
 String authorName = parts[1];  
 addAuthor(authorName);  
 return "Author added successfully.";  
  
 case "REMOVE\_AUTHOR":  
 int authorIdToRemove = Integer.*parseInt*(parts[1]);  
 removeAuthor(authorIdToRemove);  
 return "Author removed successfully.";  
  
 case "ADD\_BOOK":  
 String bookTitle = parts[1];  
 String bookISBN = parts[2];  
 int authorIdForBook = Integer.*parseInt*(parts[3]);  
 addBook(bookTitle, bookISBN, authorIdForBook);  
 return "Book added successfully.";  
  
 case "REMOVE\_BOOK":  
 int bookIdToRemove = Integer.*parseInt*(parts[1]);  
 removeBook(bookIdToRemove);  
 return "Book removed successfully.";  
  
 case "EDIT\_BOOK":  
 int bookIdToEdit = Integer.*parseInt*(parts[1]);  
 String newBookTitle = parts[2];  
 String newBookISBN = parts[3];  
 int newAuthorIdForBook = Integer.*parseInt*(parts[4]);  
 editBook(bookIdToEdit, newBookTitle, newBookISBN, newAuthorIdForBook);  
 return "Book edited successfully.";  
  
 case "COUNT\_BOOKS":  
 int totalBooks = countBooks();  
 return "Total number of books: " + totalBooks;  
  
 case "GET\_ALL\_BOOKS\_WITH\_AUTHORS":  
 String booksWithAuthors = getAllBooksWithAuthors();  
 return booksWithAuthors;  
  
 case "GET\_BOOKS\_BY\_AUTHOR":  
 int authorId = Integer.*parseInt*(parts[1]);  
 String booksByAuthor = getBooksByAuthor(authorId);  
 return booksByAuthor;  
  
 case "GET\_ALL\_AUTHORS":  
 String allAuthors = getAllAuthors();  
 return allAuthors;  
  
 default:  
 return "Invalid operation.";  
 }  
 }  
  
 private void addAuthor(String authorName) throws SQLException {  
 String sql = "INSERT INTO authors (name) VALUES (?)";  
 try (PreparedStatement statement = connection.prepareStatement(sql)) {  
 statement.setString(1, authorName);  
 statement.executeUpdate();  
 }  
 }  
  
 private void removeAuthor(int authorId) throws SQLException {  
 String sql = "DELETE FROM authors WHERE id = ?";  
 try (PreparedStatement statement = connection.prepareStatement(sql)) {  
 statement.setInt(1, authorId);  
 statement.executeUpdate();  
 }  
 }  
  
 private void addBook(String bookTitle, String bookISBN, int authorId) throws SQLException {  
 String sql = "INSERT INTO books (title, isbn, author\_id) VALUES (?, ?, ?)";  
 try (PreparedStatement statement = connection.prepareStatement(sql)) {  
 statement.setString(1, bookTitle);  
 statement.setString(2, bookISBN);  
 statement.setInt(3, authorId);  
 statement.executeUpdate();  
 }  
 }  
  
 private void removeBook(int bookId) throws SQLException {  
 String sql = "DELETE FROM books WHERE id = ?";  
 try (PreparedStatement statement = connection.prepareStatement(sql)) {  
 statement.setInt(1, bookId);  
 statement.executeUpdate();  
 }  
 }  
  
 private void editBook(int bookId, String newBookTitle, String newBookISBN, int newAuthorId) throws SQLException {  
 String sql = "UPDATE books SET title = ?, isbn = ?, author\_id = ? WHERE id = ?";  
 try (PreparedStatement statement = connection.prepareStatement(sql)) {  
 statement.setString(1, newBookTitle);  
 statement.setString(2, newBookISBN);  
 statement.setInt(3, newAuthorId);  
 statement.setInt(4, bookId);  
 statement.executeUpdate();  
 }  
 }  
  
 private int countBooks() throws SQLException {  
 String sql = "SELECT COUNT(\*) FROM books";  
 try (PreparedStatement statement = connection.prepareStatement(sql)) {  
 ResultSet resultSet = statement.executeQuery();  
 resultSet.next();  
 return resultSet.getInt(1);  
 }  
 }  
  
 private String getAllBooksWithAuthors() throws SQLException {  
 StringBuilder result = new StringBuilder();  
 String sql = "SELECT books.id, books.title, books.isbn, authors.name " +  
 "FROM books INNER JOIN authors ON books.author\_id = authors.id";  
 try (PreparedStatement statement = connection.prepareStatement(sql)) {  
 ResultSet resultSet = statement.executeQuery();  
 while (resultSet.next()) {  
 result.append("Book ID: ").append(resultSet.getInt(1)).append(", ");  
 result.append("Title: ").append(resultSet.getString(2)).append(", ");  
 result.append("ISBN: ").append(resultSet.getString(3)).append(", ");  
 result.append("Author: ").append(resultSet.getString(4)).append("\n");  
 }  
 }  
 return result.toString();  
 }  
  
 private String getBooksByAuthor(int authorId) throws SQLException {  
 StringBuilder result = new StringBuilder();  
 String sql = "SELECT id, title, isbn FROM books WHERE author\_id = ?";  
 try (PreparedStatement statement = connection.prepareStatement(sql)) {  
 statement.setInt(1, authorId);  
 ResultSet resultSet = statement.executeQuery();  
 while (resultSet.next()) {  
 result.append("Book ID: ").append(resultSet.getInt(1)).append(", ");  
 result.append("Title: ").append(resultSet.getString(2)).append(", ");  
 result.append("ISBN: ").append(resultSet.getString(3)).append("\n");  
 }  
 }  
 return result.toString();  
 }  
  
 private String getAllAuthors() throws SQLException {  
 StringBuilder result = new StringBuilder();  
 String sql = "SELECT id, name FROM authors";  
 try (PreparedStatement statement = connection.prepareStatement(sql)) {  
 ResultSet resultSet = statement.executeQuery();  
 while (resultSet.next()) {  
 result.append("Author ID: ").append(resultSet.getInt(1)).append(", ");  
 result.append("Name: ").append(resultSet.getString(2)).append("\n");  
 }  
 }  
 return result.toString();  
 }  
}

package org.labs;  
  
import java.io.\*;  
import java.net.\*;  
import java.sql.\*;  
  
public class ClientHandler extends Thread {  
 private Socket clientSocket;  
 private PrintWriter writer;  
 private BufferedReader reader;  
  
 private Connection connection;  
  
 public ClientHandler(Socket socket, Connection connection) {  
 this.clientSocket = socket;  
 this.connection = connection;  
 try {  
 writer = new PrintWriter(socket.getOutputStream(), true);  
 reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 @Override  
 public void run() {  
 try {  
 String request = reader.readLine();  
  
 String response = processRequest(request);  
 writer.println(response);  
  
 reader.close();  
 writer.close();  
 clientSocket.close();  
  
 } catch (IOException | SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private String processRequest(String request) throws SQLException {  
 String[] parts = request.split(",");  
 String operation = parts[0];  
  
 switch (operation) {  
 case "ADD\_AUTHOR":  
 String authorName = parts[1];  
 addAuthor(authorName);  
 return "Author added successfully.";  
  
 case "REMOVE\_AUTHOR":  
 int authorIdToRemove = Integer.*parseInt*(parts[1]);  
 removeAuthor(authorIdToRemove);  
 return "Author removed successfully.";  
  
 case "ADD\_BOOK":  
 String bookTitle = parts[1];  
 String bookISBN = parts[2];  
 int authorIdForBook = Integer.*parseInt*(parts[3]);  
 addBook(bookTitle, bookISBN, authorIdForBook);  
 return "Book added successfully.";  
  
 case "REMOVE\_BOOK":  
 int bookIdToRemove = Integer.*parseInt*(parts[1]);  
 removeBook(bookIdToRemove);  
 return "Book removed successfully.";  
  
 case "EDIT\_BOOK":  
 int bookIdToEdit = Integer.*parseInt*(parts[1]);  
 String newBookTitle = parts[2];  
 String newBookISBN = parts[3];  
 int newAuthorIdForBook = Integer.*parseInt*(parts[4]);  
 editBook(bookIdToEdit, newBookTitle, newBookISBN, newAuthorIdForBook);  
 return "Book edited successfully.";  
  
 case "COUNT\_BOOKS":  
 int totalBooks = countBooks();  
 return "Total number of books: " + totalBooks;  
  
 case "GET\_ALL\_BOOKS\_WITH\_AUTHORS":  
 String booksWithAuthors = getAllBooksWithAuthors();  
 return booksWithAuthors;  
  
 case "GET\_BOOKS\_BY\_AUTHOR":  
 int authorId = Integer.*parseInt*(parts[1]);  
 String booksByAuthor = getBooksByAuthor(authorId);  
 return booksByAuthor;  
  
 case "GET\_ALL\_AUTHORS":  
 String allAuthors = getAllAuthors();  
 return allAuthors;  
  
 default:  
 return "Invalid operation.";  
 }  
 }  
  
 private void addAuthor(String authorName) throws SQLException {  
 String sql = "INSERT INTO authors (name) VALUES (?)";  
 try (PreparedStatement statement = connection.prepareStatement(sql)) {  
 statement.setString(1, authorName);  
 statement.executeUpdate();  
 }  
 }  
  
 private void removeAuthor(int authorId) throws SQLException {  
 String sql = "DELETE FROM authors WHERE id = ?";  
 try (PreparedStatement statement = connection.prepareStatement(sql)) {  
 statement.setInt(1, authorId);  
 statement.executeUpdate();  
 }  
 }  
  
 private void addBook(String bookTitle, String bookISBN, int authorId) throws SQLException {  
 String sql = "INSERT INTO books (title, isbn, author\_id) VALUES (?, ?, ?)";  
 try (PreparedStatement statement = connection.prepareStatement(sql)) {  
 statement.setString(1, bookTitle);  
 statement.setString(2, bookISBN);  
 statement.setInt(3, authorId);  
 statement.executeUpdate();  
 }  
 }  
  
 private void removeBook(int bookId) throws SQLException {  
 String sql = "DELETE FROM books WHERE id = ?";  
 try (PreparedStatement statement = connection.prepareStatement(sql)) {  
 statement.setInt(1, bookId);  
 statement.executeUpdate();  
 }  
 }  
  
 private void editBook(int bookId, String newBookTitle, String newBookISBN, int newAuthorId) throws SQLException {  
 String sql = "UPDATE books SET title = ?, isbn = ?, author\_id = ? WHERE id = ?";  
 try (PreparedStatement statement = connection.prepareStatement(sql)) {  
 statement.setString(1, newBookTitle);  
 statement.setString(2, newBookISBN);  
 statement.setInt(3, newAuthorId);  
 statement.setInt(4, bookId);  
 statement.executeUpdate();  
 }  
 }  
  
 private int countBooks() throws SQLException {  
 String sql = "SELECT COUNT(\*) FROM books";  
 try (PreparedStatement statement = connection.prepareStatement(sql)) {  
 ResultSet resultSet = statement.executeQuery();  
 resultSet.next();  
 return resultSet.getInt(1);  
 }  
 }  
  
 private String getAllBooksWithAuthors() throws SQLException {  
 StringBuilder result = new StringBuilder();  
 String sql = "SELECT books.id, books.title, books.isbn, authors.name " +  
 "FROM books INNER JOIN authors ON books.author\_id = authors.id";  
 try (PreparedStatement statement = connection.prepareStatement(sql)) {  
 ResultSet resultSet = statement.executeQuery();  
 while (resultSet.next()) {  
 result.append("Book ID: ").append(resultSet.getInt(1)).append(", ");  
 result.append("Title: ").append(resultSet.getString(2)).append(", ");  
 result.append("ISBN: ").append(resultSet.getString(3)).append(", ");  
 result.append("Author: ").append(resultSet.getString(4)).append("\n");  
 }  
 }  
 return result.toString();  
 }  
  
 private String getBooksByAuthor(int authorId) throws SQLException {  
 StringBuilder result = new StringBuilder();  
 String sql = "SELECT id, title, isbn FROM books WHERE author\_id = ?";  
 try (PreparedStatement statement = connection.prepareStatement(sql)) {  
 statement.setInt(1, authorId);  
 ResultSet resultSet = statement.executeQuery();  
 while (resultSet.next()) {  
 result.append("Book ID: ").append(resultSet.getInt(1)).append(", ");  
 result.append("Title: ").append(resultSet.getString(2)).append(", ");  
 result.append("ISBN: ").append(resultSet.getString(3)).append("\n");  
 }  
 }  
 return result.toString();  
 }  
  
 private String getAllAuthors() throws SQLException {  
 StringBuilder result = new StringBuilder();  
 String sql = "SELECT id, name FROM authors";  
 try (PreparedStatement statement = connection.prepareStatement(sql)) {  
 ResultSet resultSet = statement.executeQuery();  
 while (resultSet.next()) {  
 result.append("Author ID: ").append(resultSet.getInt(1)).append(", ");  
 result.append("Name: ").append(resultSet.getString(2)).append("\n");  
 }  
 }  
 return result.toString();  
 }  
}

package org.labs;  
  
import java.io.\*;  
import java.net.Socket;  
import java.net.SocketTimeoutException;  
  
public class LibraryClient {  
  
 private static final String *SERVER\_HOST* = "localhost";  
 private static final int *SERVER\_PORT* = 12345;  
  
 public static void main(String[] args) {  
 try (Socket socket = new Socket(*SERVER\_HOST*, *SERVER\_PORT*);  
 BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 PrintWriter writer = new PrintWriter(socket.getOutputStream(), true)) {  
  
 socket.setSoTimeout(10000);  
 *addAuthor*(writer, reader, "John Doe");  
 *addBook*(writer, reader, "Sample Book", "1234567890", 1);  
 *editBook*(writer, reader, 2, "Updated Book", "0987654321", 1);  
 int totalBooks = *countBooks*(writer, reader);  
 *removeBook*(writer, reader, 1);  
 *removeAuthor*(writer, reader, 1);  
  
 System.*out*.println("Total number of books: " + totalBooks);  
 String allBooksWithAuthors = *getAllBooksWithAuthors*(writer, reader);  
 System.*out*.println("All books with authors:\n" + allBooksWithAuthors);  
 String booksByAuthor = *getBooksByAuthor*(writer, reader, 2);  
 System.*out*.println("Books by author:\n" + booksByAuthor);  
 String allAuthors = *getAllAuthors*(writer, reader);  
 System.*out*.println("All authors:\n" + allAuthors);  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 }  
  
 private static void addAuthor(PrintWriter writer, BufferedReader reader, String authorName) throws InterruptedException {  
 writer.println("ADD\_AUTHOR," + authorName);  
 Thread.*sleep*(1000);  
 String response = *receiveResponse*(reader);  
 System.*out*.println(response);  
 Thread.*sleep*(1000);  
 }  
  
 private static void removeAuthor(PrintWriter writer, BufferedReader reader, int authorId) throws InterruptedException {  
 writer.println("REMOVE\_AUTHOR," + authorId);  
 Thread.*sleep*(1000);  
 String response = *receiveResponse*(reader);  
 System.*out*.println(response);  
 }  
  
 private static void addBook(PrintWriter writer, BufferedReader reader, String bookTitle, String bookISBN, int authorId) throws InterruptedException {  
 writer.println("ADD\_BOOK," + bookTitle + "," + bookISBN + "," + authorId);  
 Thread.*sleep*(1000);  
 String response = *receiveResponse*(reader);  
 System.*out*.println(response);  
 }  
  
 private static void removeBook(PrintWriter writer, BufferedReader reader, int bookId) throws InterruptedException {  
 writer.println("REMOVE\_BOOK," + bookId);  
 Thread.*sleep*(1000);  
 String response = *receiveResponse*(reader);  
 System.*out*.println(response);  
 }  
  
 private static void editBook(PrintWriter writer, BufferedReader reader, int bookId, String newBookTitle, String newBookISBN, int newAuthorId) throws InterruptedException {  
 writer.println("EDIT\_BOOK," + bookId + "," + newBookTitle + "," + newBookISBN + "," + newAuthorId);  
 Thread.*sleep*(1000);  
 String response = *receiveResponse*(reader);  
 System.*out*.println(response);  
 }  
  
 private static int countBooks(PrintWriter writer, BufferedReader reader) throws InterruptedException {  
 writer.println("COUNT\_BOOKS");  
 Thread.*sleep*(1000);  
 String response = *receiveResponse*(reader);  
  
 if (response.startsWith("Error")) {  
 System.*out*.println(response);  
 return -1;  
 }  
  
 return Integer.*parseInt*(response.split(" ")[4]);  
 }  
  
 private static String getAllBooksWithAuthors(PrintWriter writer, BufferedReader reader) throws InterruptedException {  
 writer.println("GET\_ALL\_BOOKS\_WITH\_AUTHORS");  
 Thread.*sleep*(1000);  
 return *receiveResponse*(reader);  
 }  
  
 private static String getBooksByAuthor(PrintWriter writer, BufferedReader reader, int authorId) throws InterruptedException {  
 writer.println("GET\_BOOKS\_BY\_AUTHOR," + authorId);  
 Thread.*sleep*(1000);  
 return *receiveResponse*(reader);  
 }  
  
 private static String getAllAuthors(PrintWriter writer, BufferedReader reader) throws InterruptedException {  
 writer.println("GET\_ALL\_AUTHORS");  
 Thread.*sleep*(1000);  
 return *receiveResponse*(reader);  
 }  
  
 private static String receiveResponse(BufferedReader reader) {  
 try {  
 return reader.readLine();  
 } catch (SocketTimeoutException e) {  
 System.*out*.println("Час очікування з'єднання чи відповіді вийшов.");  
 e.printStackTrace();  
 return "Error receiving response.";  
 }catch (IOException e) {  
 e.printStackTrace();  
 return "Error receiving response.";  
 }  
 }  
}

Програма реалізує сервер-клієнтську архітектуру для управління бібліотекою.

Клас **LibraryServer**:

* **Поле PORT (int):**
  + Порт, на якому працює сервер.
* **Поле connection (Connection):**
  + З'єднання з базою даних.
* **Метод main(String[] args):**
  + Основний метод сервера.
  + Створює **ServerSocket** на вказаному порту.
  + Очікує на підключення клієнтів та запускає для кожного нового клієнта **ClientHandler**.

Клас **ClientHandler**:

* **Поле clientSocket (Socket):**
  + Сокет для взаємодії з клієнтом.
* **Поле writer (PrintWriter):**
  + Об'єкт для виведення даних до клієнта.
* **Поле reader (BufferedReader):**
  + Об'єкт для читання даних від клієнта.
* **Поле connection (Connection):**
  + З'єднання з базою даних.
* **Метод run():**
  + Перевизначений метод для обробки запитів клієнта.
  + Отримує запит, обробляє його та відправляє відповідь клієнту.
* **Метод processRequest(String request): String:**
  + Обробляє отриманий запит від клієнта та повертає відповідь.

Клас **LibraryClient**:

* **Поле SERVER\_HOST (String):**
  + Адреса сервера.
* **Поле SERVER\_PORT (int):**
  + Порт сервера.
* **Метод main(String[] args):**
  + Основний метод клієнта.
  + Встановлює з'єднання з сервером та взаємодіє з ним.
* **Методи для взаємодії з сервером:**
  + **addAuthor**, **removeAuthor**, **addBook**, **removeBook**, **editBook**, **countBooks**, **getAllBooksWithAuthors**, **getBooksByAuthor**, **getAllAuthors**.
* **Метод receiveResponse(BufferedReader reader): String:**
  + Отримує відповідь від сервера через **BufferedReader**.

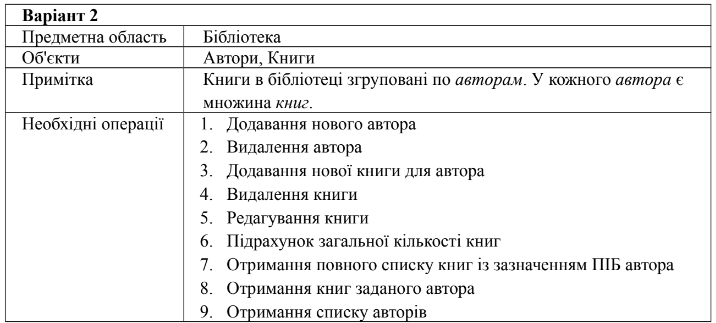
**Протокол взаємодії клієнта і сервера:**

1. **ADD\_AUTHOR:**
   * Клієнт надсилає запит "ADD\_AUTHOR,authorName" для додавання нового автора.
   * Сервер обробляє запит, додає нового автора до бази даних та надсилає відповідь "Author added successfully."
2. **REMOVE\_AUTHOR:**
   * Клієнт надсилає запит "REMOVE\_AUTHOR,authorId" для видалення автора за ідентифікатором.
   * Сервер обробляє запит, видаляє автора з бази даних та надсилає відповідь "Author removed successfully."
3. **ADD\_BOOK:**
   * Клієнт надсилає запит "ADD\_BOOK,bookTitle,bookISBN,authorId" для додавання нової книги.
   * Сервер обробляє запит, додає нову книгу до бази даних та надсилає відповідь "Book added successfully."
4. **REMOVE\_BOOK:**
   * Клієнт надсилає запит "REMOVE\_BOOK,bookId" для видалення книги за ідентифікатором.
   * Сервер обробляє запит, видаляє книгу з бази даних та надсилає відповідь "Book removed successfully."
5. **EDIT\_BOOK:**
   * Клієнт надсилає запит "EDIT\_BOOK,bookId,newBookTitle,newBookISBN,newAuthorId" для редагування інформації про книгу.
   * Сервер обробляє запит, оновлює дані про книгу в базі даних та надсилає відповідь "Book edited successfully."
6. **COUNT\_BOOKS:**
   * Клієнт надсилає запит "COUNT\_BOOKS" для отримання загальної кількості книг у бібліотеці.
   * Сервер обробляє запит, визначає кількість та надсилає відповідь "Total number of books: {totalBooks}"
7. **GET\_ALL\_BOOKS\_WITH\_AUTHORS:**
   * Клієнт надсилає запит "GET\_ALL\_BOOKS\_WITH\_AUTHORS" для отримання списку всіх книг з авторами.
   * Сервер обробляє запит, формує відповідь та надсилає список книг.
8. **GET\_BOOKS\_BY\_AUTHOR:**
   * Клієнт надсилає запит "GET\_BOOKS\_BY\_AUTHOR,authorId" для отримання списку книг конкретного автора.
   * Сервер обробляє запит, формує відповідь та надсилає список книг.
9. **GET\_ALL\_AUTHORS:**
   * Клієнт надсилає запит "GET\_ALL\_AUTHORS" для отримання списку всіх авторів.
   * Сервер обробляє запит, формує відповідь та надсилає список авторів.
10. **Invalid operation:**
    * Клієнт надсилає запит, який не відповідає жодній із вищеперелічених операцій.
    * Сервер надсилає відповідь "Invalid operation."

**«Клієнт-серверна взаємодія на основі технології віддаленого виклику процедур»**

**Мета:** в інформаційній системі, розробленій в рамках лабораторної роботи No3, організувати взаємодію клієнта і сервера з використанням механізму віддаленого виклику процедур / методів (RPC / RMI).

**Варіант завдання:**



**Лістинг розробленої програми**

package org.labs;  
  
import java.rmi.Remote;  
import java.rmi.RemoteException;  
import java.util.List;  
  
public interface LibraryService extends Remote {  
 void addAuthor(String name) throws RemoteException;  
 void removeAuthor(int authorId) throws RemoteException;  
 void addBook(String title, String isbn, int authorId) throws RemoteException;  
 void removeBook(int bookId) throws RemoteException;  
 void editBook(int bookId, String title, String isbn, int authorId) throws RemoteException;  
 int countBooks() throws RemoteException;  
 List<String> getAllBooksWithAuthors() throws RemoteException;  
 List<String> getBooksByAuthor(int authorId) throws RemoteException;  
 List<String> getAllAuthors() throws RemoteException;  
}

package org.labs;  
  
import org.labs.LibraryService;  
  
import java.rmi.RemoteException;  
import java.rmi.server.UnicastRemoteObject;  
import java.sql.\*;  
import java.util.ArrayList;  
import java.util.List;  
  
public class LibraryServiceImpl extends UnicastRemoteObject implements LibraryService {  
 private Connection connection;  
  
 public LibraryServiceImpl() throws RemoteException {  
 super();  
 initializeDatabase();  
 }  
  
 private void initializeDatabase() {  
 try {  
 String jdbcUrl = "jdbc:mysql://localhost/library\_db";  
 String jdbcUser = "root";  
 String jdbcPassword = "12345";  
 connection = DriverManager.getConnection(jdbcUrl, jdbcUser, jdbcPassword);  
  
 createTablesIfNotExists();  
 } catch (SQLException e) {  
 throw new RuntimeException("Failed to initialize the database connection.", e);  
 }  
 }  
  
 private void createTablesIfNotExists() throws SQLException {  
 try (Statement statement = connection.createStatement()) {  
 statement.executeUpdate("CREATE DATABASE IF NOT EXISTS library\_db");  
 statement.executeUpdate("USE library\_db");  
  
 statement.executeUpdate("CREATE TABLE IF NOT EXISTS authors (" +  
 "id INT AUTO\_INCREMENT PRIMARY KEY," +  
 "name VARCHAR(255) NOT NULL)");  
  
 statement.executeUpdate("CREATE TABLE IF NOT EXISTS books (" +  
 "id INT AUTO\_INCREMENT PRIMARY KEY," +  
 "title VARCHAR(255) NOT NULL," +  
 "isbn VARCHAR(20) NOT NULL," +  
 "author\_id INT," +  
 "FOREIGN KEY (author\_id) REFERENCES authors(id) ON DELETE CASCADE)");  
 }  
 }  
  
 @Override  
 public void addAuthor(String name) throws RemoteException {  
 try (PreparedStatement preparedStatement = connection.prepareStatement("INSERT INTO authors (name) VALUES (?)", Statement.RETURN\_GENERATED\_KEYS)) {  
 preparedStatement.setString(1, name);  
 preparedStatement.executeUpdate();  
  
 ResultSet generatedKeys = preparedStatement.getGeneratedKeys();  
 if (generatedKeys.next()) {  
 int authorId = generatedKeys.getInt(1);  
 System.out.println("Author added successfully. Author ID: " + authorId);  
 } else {  
 System.out.println("Failed to retrieve the generated author ID.");  
 }  
 } catch (SQLException e) {  
 System.out.println("Error adding author: " + e.getMessage());  
 }  
 }  
  
 @Override  
 public void removeAuthor(int authorId) throws RemoteException {  
 try (PreparedStatement preparedStatement = connection.prepareStatement("DELETE FROM authors WHERE id = ?")) {  
 preparedStatement.setInt(1, authorId);  
 int rowsAffected = preparedStatement.executeUpdate();  
  
 if (rowsAffected > 0) {  
 System.out.println("Author removed successfully.");  
 } else {  
 System.out.println("Author not found.");  
 }  
 } catch (SQLException e) {  
 System.out.println("Error removing author: " + e.getMessage());  
 }  
 }  
  
 @Override  
 public void addBook(String title, String isbn, int authorId) throws RemoteException {  
 try (PreparedStatement preparedStatement = connection.prepareStatement("INSERT INTO books (title, isbn, author\_id) VALUES (?, ?, ?)", Statement.RETURN\_GENERATED\_KEYS)) {  
 preparedStatement.setString(1, title);  
 preparedStatement.setString(2, isbn);  
 preparedStatement.setInt(3, authorId);  
 preparedStatement.executeUpdate();  
  
 ResultSet generatedKeys = preparedStatement.getGeneratedKeys();  
 if (generatedKeys.next()) {  
 int bookId = generatedKeys.getInt(1);  
 System.out.println("Book added successfully. Book ID: " + bookId);  
 } else {  
 System.out.println("Failed to retrieve the generated book ID.");  
 }  
 } catch (SQLException e) {  
 System.*out*.println("Error adding book: " + e.getMessage());  
 }  
 }  
  
 @Override  
 public void removeBook(int bookId) throws RemoteException {  
 try (PreparedStatement preparedStatement = connection.prepareStatement("DELETE FROM books WHERE id = ?")) {  
 preparedStatement.setInt(1, bookId);  
 int rowsAffected = preparedStatement.executeUpdate();  
  
 if (rowsAffected > 0) {  
 System.*out*.println("Book removed successfully.");  
 } else {  
 System.*out*.println("Book not found.");  
 }  
 } catch (SQLException e) {  
 System.*out*.println("Error removing book: " + e.getMessage());  
 }  
 }  
  
 @Override  
 public void editBook(int bookId, String title, String isbn, int authorId) throws RemoteException {  
 try (PreparedStatement preparedStatement = connection.prepareStatement("UPDATE books SET title = ?, isbn = ?, author\_id = ? WHERE id = ?")) {  
 preparedStatement.setString(1, title);  
 preparedStatement.setString(2, isbn);  
 preparedStatement.setInt(3, authorId);  
 preparedStatement.setInt(4, bookId);  
  
 int rowsAffected = preparedStatement.executeUpdate();  
 if (rowsAffected > 0) {  
 System.out.println("Book edited successfully.");  
 } else {  
 System.out.println("Book not found.");  
 }  
 } catch (SQLException e) {  
 System.out.println("Error editing book: " + e.getMessage());  
 }  
 }  
  
 @Override  
 public int countBooks() throws RemoteException {  
 try (Statement statement = connection.createStatement();  
 ResultSet resultSet = statement.executeQuery("SELECT COUNT(\*) FROM books")) {  
 if (resultSet.next()) {  
 return resultSet.getInt(1);  
 }  
 } catch (SQLException e) {  
 System.out.println("Error counting books: " + e.getMessage());  
 }  
 return -1;  
 }  
  
 @Override  
 public List<String> getAllBooksWithAuthors() throws RemoteException {  
 List<String> result = new ArrayList<>();  
 try (Statement statement = connection.createStatement();  
 ResultSet resultSet = statement.executeQuery("SELECT books.id, books.title, books.isbn, authors.name " +  
 "FROM books INNER JOIN authors ON books.author\_id = authors.id")) {  
 while (resultSet.next()) {  
 int bookId = resultSet.getInt("id");  
 String bookTitle = resultSet.getString("title");  
 String bookIsbn = resultSet.getString("isbn");  
 String authorName = resultSet.getString("name");  
 result.add("Book ID: " + bookId + ", Title: " + bookTitle + ", ISBN: " + bookIsbn + ", Author: " + authorName);  
 }  
 } catch (SQLException e) {  
 System.out.println("Error retrieving books with authors: " + e.getMessage());  
 }  
 return result;  
 }  
  
 @Override  
 public List<String> getBooksByAuthor(int authorId) throws RemoteException {  
 List<String> result = new ArrayList<>();  
 try (PreparedStatement preparedStatement = connection.prepareStatement("SELECT id, title, isbn FROM books WHERE author\_id = ?");  
 ResultSet resultSet = preparedStatement.executeQuery()) {  
 preparedStatement.setInt(1, authorId);  
 while (resultSet.next()) {  
 int bookId = resultSet.getInt("id");  
 String bookTitle = resultSet.getString("title");  
 String bookIsbn = resultSet.getString("isbn");  
 result.add("Book ID: " + bookId + ", Title: " + bookTitle + ", ISBN: " + bookIsbn);  
 }  
 } catch (SQLException e) {  
 System.out.println("Error retrieving books by author: " + e.getMessage());  
 }  
 return result;  
 }  
  
 @Override  
 public List<String> getAllAuthors() throws RemoteException {  
 List<String> result = new ArrayList<>();  
 try (Statement statement = connection.createStatement();  
 ResultSet resultSet = statement.executeQuery("SELECT id, name FROM authors")) {  
 while (resultSet.next()) {  
 int authorId = resultSet.getInt("id");  
 String authorName = resultSet.getString("name");  
 result.add("Author ID: " + authorId + ", Name: " + authorName);  
 }  
 } catch (SQLException e) {  
 System.out.println("Error retrieving authors: " + e.getMessage());  
 }  
 return result;  
 }  
  
 public static void main(String[] args) {  
 try {  
 LibraryService libraryService = new LibraryServiceImpl();  
 java.rmi.registry.LocateRegistry.createRegistry(8080);  
 java.rmi.Naming.rebind("LibraryService", libraryService);  
 System.out.println("Server is ready.");  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

package org.labs;  
  
import java.rmi.Naming;  
import java.util.List;  
  
public class LibraryClient {  
 public static void main(String[] args) {  
 try {  
 LibraryService libraryService = (LibraryService) Naming.*lookup*("rmi://localhost/LibraryService");  
  
  
 libraryService.addAuthor("John Doe");  
 libraryService.addBook("Sample Book", "1234567890", 1);  
 libraryService.editBook(2, "Updated Book", "0987654321", 1);  
 int totalBooks = libraryService.countBooks();  
 libraryService.removeBook(1);  
 libraryService.removeAuthor(1);  
  
  
 System.*out*.println("Total number of books: " + totalBooks);  
 List<String> allBooksWithAuthors = libraryService.getAllBooksWithAuthors();  
 System.*out*.println("All books with authors:\n" + allBooksWithAuthors);  
 List<String> booksByAuthor = libraryService.getBooksByAuthor(1);  
 System.*out*.println("Books by author:\n" + booksByAuthor);  
 List<String> allAuthors = libraryService.getAllAuthors();  
 System.*out*.println("All authors:\n" + allAuthors);  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

Програма реалізує використання RMI (Remote Method Invocation) для взаємодії між сервером та клієнтом у системі управління бібліотекою.

Клас **LibraryServiceImpl**:

* **Поле connection (Connection):**
  + З'єднання з базою даних.
* **Конструктор LibraryServiceImpl() throws RemoteException:**
  + Створює об'єкт **LibraryServiceImpl**.
  + Ініціалізує з'єднання з базою даних в методі **initializeDatabase()**.
* **Метод initializeDatabase():**
  + Ініціалізує з'єднання з базою даних за допомогою JDBC.
  + Створює таблиці "authors" та "books", якщо вони не існують.
* **Метод createTablesIfNotExists() throws SQLException:**
  + Створює базу даних "library\_db" та переводить робочу область на неї.
  + Створює таблицю "authors" з полями **id** та **name**.
  + Створює таблицю "books" з полями **id**, **title**, **isbn**, **author\_id** та зовнішнім ключем до таблиці "authors".
* **Реалізація методів інтерфейсу LibraryService:**
  + **addAuthor**, **removeAuthor**, **addBook**, **removeBook**, **editBook**, **countBooks**, **getAllBooksWithAuthors**, **getBooksByAuthor**, **getAllAuthors**.
  + Кожен з методів виконує відповідну операцію з базою даних та виводить результат на консоль.
* **Метод main(String[] args):**
  + Запускає RMI-сервер на порту 8080 та реєструє об'єкт **LibraryService** під ім'ям "LibraryService" для подальшого використання з клієнта.

Клас **LibraryClient**:

* **Метод main(String[] args):**
  + Здійснює підключення до RMI-сервера за допомогою імені "LibraryService".
  + Викликає ряд методів **LibraryService** для додавання та видалення авторів та книг, редагування та отримання інформації про книги та авторів.
  + Виводить результати виконаних операцій на консоль.

**Інтерфейс LibraryService:**

* Інтерфейс визначає методи для взаємодії з бібліотекою через RMI.
* Кожен метод описує певну операцію, таку як додавання або видалення автора, додавання книги тощо.
* Кожен метод може викидати виняток **RemoteException**, оголошений у **java.rmi.Remote**.

**Протокол взаємодії клієнта і сервера:**

* Клієнт взаємодіє з сервером за допомогою викликів методів інтерфейсу **LibraryService**.
* Сервер виконує запити клієнта, взаємодіючи з базою даних, та повертає результати викликів клієнту.
* Програма використовує RMI для взаємодії між об'єктами, розташованими на різних вузлах мережі.
* Для запуску RMI-сервера використовується порт 8080.

**Висновок**

Виконуючи дану лабораторну роботу було розроблено на основі програми, розробленої в рамках лабораторної роботи No7, серверну програму, що забезпечує виконання віддалених запитів з управління об'єктами. Відомості про об'єкти повинні зберігаються в базі даних. Було розроблено також клієнтську програму, що відправляє серверу запити на введення, редагування і отримання інформації про об'єкти. Взаємодія між клієнтом і сервером здійснюється по протоколу TCP / IP. В другій частині лабораторної роботи було в інформаційній системі, розробленій в рамках першої частини, організовано взаємодію клієнта і сервера з використанням механізму віддаленого виклику процедур / методів (RPC / RMI).