| **Name of Student:** Bhavesh Laku | | | |
| --- | --- | --- | --- |
| **Roll Number:** 28 | | **LAB Assignment Number:** 4 | |
| **Title of LAB Assignment:** Implementation of Remote Method Communication using JDBC and RMI. | | | |
| **DOP:** 02/09/2024 | | **DOS:** 04/09/2024 | |
| **CO Mapped:** | **PO Mapped:** | **Signature:** | **Marks:** |

Aim: Implementation of Remote Method Communication using JDBC and RMI.

Q1. Using MySQL create a Library database. Create table Book (Book\_id, Book\_name, Book\_author) and retrieve the Book information from Library database.

###### **Theory:**

JDBC or Java Database Connectivity is a Java API to connect and execute the query with the database. It is a specification from Sun microsystems that provides a standard abstraction(API or Protocol) for java applications to communicate with various databases. It provides the language with java database connectivity standards. It is used to write programs required to access databases. JDBC, along with the database driver, can access databases and spreadsheets. The enterprise data stored in a relational database(RDB) can be accessed with the help of JDBC APIs.

**Definition of JDBC(Java Database Connectivity)**

JDBC is an API(Application programming interface) used in java programming to interact with databases. The classes and interfaces of JDBC allow the application to send requests made by users to the specified database.

**Purpose of JDBC**

Enterprise applications created using the JAVA EE technology need to interact with databases to store application-specific information. So, interacting with a database requires efficient database connectivity, which can be achieved by using the ODBC(Open database connectivity) driver.

This driver is used with JDBC to interact or communicate with various kinds of databases such as Oracle, MS Access, Mysql, and SQL server database.

**JDBC Drivers**

JDBC drivers are client-side adapters (installed on the client machine, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand. There are 4 types of JDBC drivers:

Type-1 driver or JDBC-ODBC bridge driver Type-2 driver or Native-API driver

Type-3 driver or Network Protocol driver Type-4 driver or Thin driver

**Remote Method Invocation (RMI)** is an API that allows an object to invoke a method on an object that exists in another address space, which could be on the same machine or on a remote machine. Through RMI, an object running in a JVM present on a computer (Client-side) can invoke methods on an object present in another JVM (Server-side). RMI creates a public remote server object that enables client and server-side communications through simple method calls on the server object.

**Stub Object:** The stub object on the client machine builds an information block and sends this information to the server.

**Skeleton:**

The skeleton object passes the request from the stub object to the remote object. It performs the following tasks

● It calls the desired method on the real object present on the server.

● It forwards the parameters received from the stub object to the method.

##### 

##### **Code:**

**Book.java**

public class Book implements java.io.Serializable { private int book\_id;

private String book\_name, book\_author;

public int getBook\_id() { return book\_id;

}

public void setBook\_id(int book\_id) { this.book\_id = book\_id;

}

public String getBook\_name() { return book\_name;

}

public void setBook\_name(String book\_name) { this.book\_name = book\_name;

}

public String getBook\_author() { return book\_author;

}

public void setBook\_author(String book\_author) { this.book\_author = book\_author;

}}

##### **BooksInterface.java**

import java.rmi.Remote;

import java.rmi.RemoteException; import java.util.\*;

public interface BooksInterface extends Remote{ public List<Book> getAllBooks() throws Exception;

public Book getBookById(int id) throws Exception;

**ImplExample.java** import java.sql.\*; import java.util.\*;

public class ImplExample implements BooksInterface{ public List<Book> getAllBooks() throws Exception {

List<Book> list = new ArrayList<Book>();

String JDBC\_DRIVER = "com.mysql.jdbc.Driver"; String DB\_URL = "jdbc:mysql://localhost:3306/Library"; Connection conn = null;

Statement stmt = null;

System.out.println("Connecting to a selected database..."); conn = DriverManager.getConnection(DB\_URL, "root", ""); System.out.println("Connected database successfully..."); System.out.println("Creating statement...");

stmt = conn.createStatement();

String sql = "SELECT \* FROM books"; ResultSet rs = stmt.executeQuery(sql); while (rs.next()) {

int id = rs.getInt("book\_id");

String name = rs.getString("book\_name"); String author = rs.getString("book\_author"); Book book = new Book(); book.setBook\_id(id); book.setBook\_name(name); book.setBook\_author(author); list.add(book);

}

rs.close(); return list;

}

public Book getBookById(int id) throws Exception { Book book = null;

String JDBC\_DRIVER = "com.mysql.jdbc.Driver"; String DB\_URL = "jdbc:mysql://localhost:3306/Library"; Connection conn = null;

Statement stmt = null;

System.out.println("Connecting to a selected database..."); conn = DriverManager.getConnection(DB\_URL, "root", ""); System.out.println("Connected database successfully...");

System.out.println("Creating statement...");

String query = "Select \* from books where book\_id=?"; PreparedStatement myStmt = conn.prepareStatement(query); myStmt.setInt(1, id);

ResultSet rs = myStmt.executeQuery(); while (rs.next()) {

int id1 = rs.getInt("book\_id");

String name = rs.getString("book\_name"); String author = rs.getString("book\_author"); book = new Book();

book.setBook\_id(id1); book.setBook\_name(name); book.setBook\_author(author);

}

rs.close(); return book;

}}

##### **Client.java**

import java.rmi.registry.LocateRegistry; import java.rmi.registry.Registry; import java.util.\*;

public class Client { Client() {}

public static void main(String[] args) throws Exception{ try {

Registry registry = LocateRegistry.getRegistry(null); BooksInterface stub = (BooksInterface) registry.lookup("books"); Scanner sc = new Scanner(System.in);

while (true) {

System.out.print("Enter choice:\n1. Get All Books\n2. Get Book By Id\n"); int option = sc.nextInt();

if (option == 1) {

List<Book> list = (List<Book>) stub.getAllBooks(); for (Book s : list) {

System.out.println("ID: " + s.getBook\_id()); System.out.println("name: " + s.getBook\_name()); System.out.println("branch: " + s.getBook\_author());

}

}

if (option == 2) { System.out.print("Enter book id: "); int id = sc.nextInt();

Book book = (Book) stub.getBookById(id); System.out.println("Id: " + book.getBook\_id() + "\nName: "

+ book.getBook\_name() + "\nAuthor: " + book.getBook\_author() + "\n");

}

}

}catch (Exception e) {

System.err.println("Client exception: " + e.toString()); e.printStackTrace();

}

}

}

##### **Server.java**

import java.rmi.registry.Registry; import java.rmi.registry.LocateRegistry; import java.rmi.RemoteException;

import java.rmi.server.UnicastRemoteObject; public class Server extends ImplExample{

public Server() {}

public static void main(String[] args) { try {

ImplExample obj = new ImplExample();

BooksInterface stub = (BooksInterface) UnicastRemoteObject.exportObject(obj, 0); Registry registry = LocateRegistry.getRegistry();

registry.bind("books", stub); System.err.println("Server ready");

}

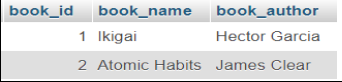
catch (Exception e) {

System.err.println("Server exception: " + e.toString()); e.printStackTrace();

}

}

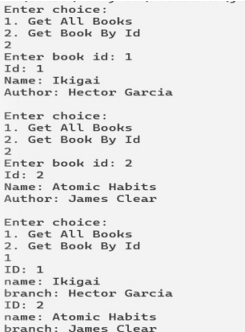
}



**Output:**

|  |  |
| --- | --- |
|  |  |

**Client :**

****

**Conclusions:**

We have successfully fetched data from Database using JDBC & remote object communication in this practical successfully.

Q2. Using MySQL create Elecrtic\_Bill database. Create table Bill (consumer\_name, bill\_due\_date, bill\_amount) and retrieve the Bill information from the Elecrtic\_Bill database using Remote Object Communication concept.

**Theory:**

JDBC or Java Database Connectivity is a Java API to connect and execute the query with the database. It is a specification from Sun microsystems that provides a standard abstraction(API or Protocol) for java applications to communicate with various databases. It provides the language with java database connectivity standards. It is used to write programs required to access databases. JDBC, along with the database driver, can access databases and spreadsheets. The enterprise data stored in a relational database(RDB) can be accessed with the help of JDBC APIs.

**Definition of JDBC(Java Database Connectivity)**

JDBC is an API(Application programming interface) used in java programming to interact with databases. The classes and interfaces of JDBC allow the application to send requests made by users to the specified database.

**Purpose of JDBC**

Enterprise applications created using the JAVA EE technology need to interact with databases to store application-specific information. So, interacting with a database requires efficient database connectivity, which can be achieved by using the ODBC(Open database connectivity) driver.

This driver is used with JDBC to interact or communicate with various kinds of databases such as Oracle, MS Access, Mysql, and SQL server databases.

**JDBC Drivers**

JDBC drivers are client-side adapters (installed on the client machine, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand. There are 4 types of JDBC drivers:

*1. Type-1 driver or JDBC-ODBC bridge driver*

*2. Type-2 driver or Native-API driver*

*3. Type-3 driver or Network Protocol driver*

*4. Type-4 driver or Thin driver*

**Remote Method Invocation (RMI)** is an API that allows an object to invoke a method on an object that exists in another address space, which could be on the same machine or on a remote machine. Through RMI, an object running in a JVM present on a computer (Client-side) can invoke methods on an object present in another JVM (Server-side). RMI creates a public remote server object that enables client and server-side communications through simple method calls on the server object.

**Stub Object:** The stub object on the client machine builds an information block and sends this information to the server.

**Skeleton:**

The skeleton object passes the request from the stub object to the remote object. It performs the following tasks

● It calls the desired method on the real object present on the server.

● It forwards the parameters received from the stub object to the method.

##### **Code:**

**Bill.java**

import java.util.Date;

public class Bill implements java.io.Serializable { private int bill\_id,amount;

private String consumer\_name; private Date bill\_due\_date; public int getBill\_id() {

return bill\_id;

}

public void setBill\_id(int bill\_id) { this.bill\_id = bill\_id;

}

public int getAmount() { return amount;

}

public void setAmount(int amount) { this.amount = amount;

}

public String getConsumer\_name() { return consumer\_name;

}

public void setConsumer\_name(String consumer\_name) { this.consumer\_name = consumer\_name;

}

public Date getBill\_due\_date() { return bill\_due\_date;

}

public void setBill\_due\_date(Date bill\_due\_date) { this.bill\_due\_date = bill\_due\_date;

}

}

##### **BillInterface.java**

import java.rmi.Remote;

import java.rmi.RemoteException; import java.util.\*;

public interface BillInterface extends Remote { public Bill getBillById(int id) throws Exception; }

**ImplExample.java** import java.sql.\*; import java.sql.Date; import java.util.\*;

public class ImplExample implements BillInterface { public Bill getBillById(int id) throws Exception { String JDBC\_DRIVER = "com.mysql.jdbc.Driver";

String DB\_URL = "jdbc:mysql://localhost:3306/Electric\_Bill"; **//name of database**

Connection conn = null; Statement stmt = null;

System.out.println("Connecting to a selected database..."); conn = DriverManager.getConnection(DB\_URL, "root", ""); System.out.println("Connected database successfully..."); System.out.println("Creating statement...");

String query = "Select \* from bill where bill\_id=?"; PreparedStatement myStmt = conn.prepareStatement(query); myStmt.setInt(1, id);

ResultSet rs = myStmt.executeQuery(); Bill bill = new Bill();

while(rs.next()) {

int id1 = rs.getInt("bill\_id");

String name = rs.getString("consumer\_name"); int amount = rs.getInt("bill\_amount"); System.out.println(id1+' '+name+' '+amount); Date date = rs.getDate("bill\_due\_date"); System.out.println(date); System.out.println("why1");

bill.setBill\_id(id); bill.setConsumer\_name(name); System.out.println("why1"); bill.setBill\_due\_date(date); System.out.println("why2"); bill.setAmount(amount);

}

System.out.println(bill); rs.close();

return bill;

}

}

##### **Client.java**

import java.rmi.registry.LocateRegistry; import java.rmi.registry.Registry; import java.util.\*;

public class Client { Client() {}

public static void main(String[] args)throws Exception { try {

Registry registry = LocateRegistry.getRegistry(null); BillInterface stub = (BillInterface) registry.lookup("bill"); Scanner sc = new Scanner(System.in); System.out.print("Enter bill id:\n");

int id = sc.nextInt();

Bill bill = (Bill)stub.getBillById(id);

System.out.println("Bill id: " + bill.getBill\_id()+"\nConsumer Name: " + bill.getConsumer\_name()+"\nBill Amount: " + bill.getAmount()+"\n" +"Due date: "+bill.getBill\_due\_date()+"\n");

}

catch (Exception e) {

System.err.println("Client exception: " + e.toString()); e.printStackTrace();}}}

##### **Server.java**

import java.rmi.registry.Registry; import java.rmi.registry.LocateRegistry; import java.rmi.RemoteException;

import java.rmi.server.UnicastRemoteObject; public class Server extends ImplExample { public Server() {}

public static void main(String args[]) { try {

ImplExample obj = new ImplExample();

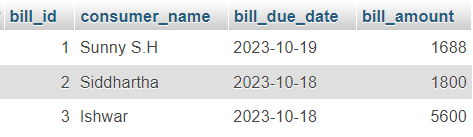
BillInterface stub = (BillInterface) UnicastRemoteObject.exportObject(obj, 0); Registry registry = LocateRegistry.getRegistry();

registry.bind("bill", stub); System.err.println("Server ready");

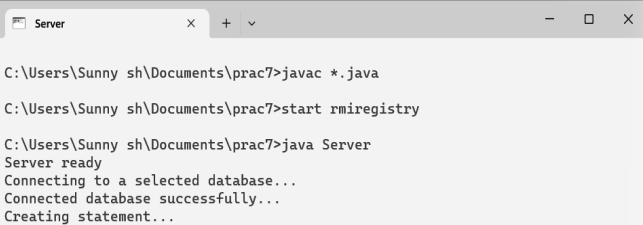
} catch (Exception e) {

System.err.println("Server exception: " + e.toString()); e.printStackTrace();

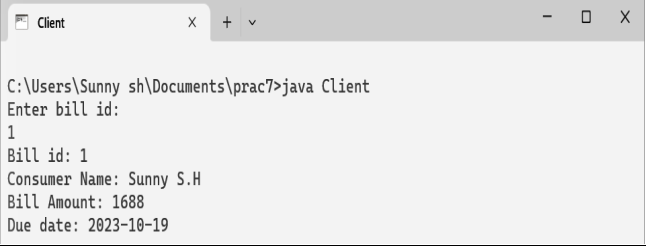
}}}



**Output Server :**

****

**Client :**

****

**Conclusions:**

We have successfully fetched electric bill records from a database using JDBC & remote Object Communication in this practice successfully.