

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();

}

}

}

```

book_id	book_name	book_author
1	Ikigai	Hector Garcia
2	Atomic Habits	James Clear

Output:

```

Server ready
Connecting to a selected database...
Connected database successfully...
Creating statement...
Connecting to a selected database...
Connected database successfully...
Creating statement...
Connecting to a selected database...
Connected database successfully...
Creating statement...

```

Client :

```
Enter choice:
1. Get All Books
2. Get Book By Id
2
Enter book id: 1
Id: 1
Name: Ikigai
Author: Hector Garcia

Enter choice:
1. Get All Books
2. Get Book By Id
2
Enter book id: 2
Id: 2
Name: Atomic Habits
Author: James Clear

Enter choice:
1. Get All Books
2. Get Book By Id
1
ID: 1
name: Ikigai
branch: Hector Garcia
ID: 2
name: Atomic Habits
branch: James Clear
```

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

```

va 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");

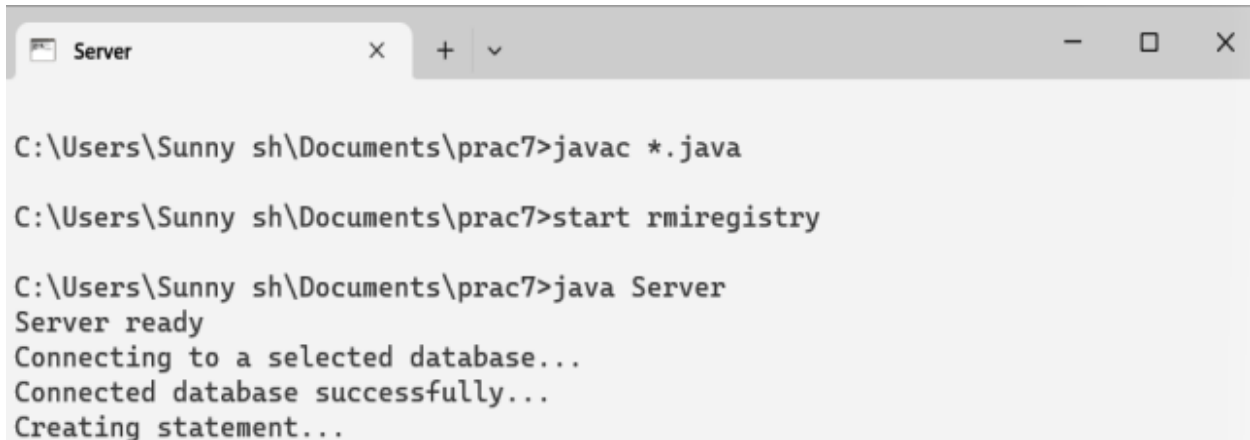
} catch (Exception e) {

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

}}}
```

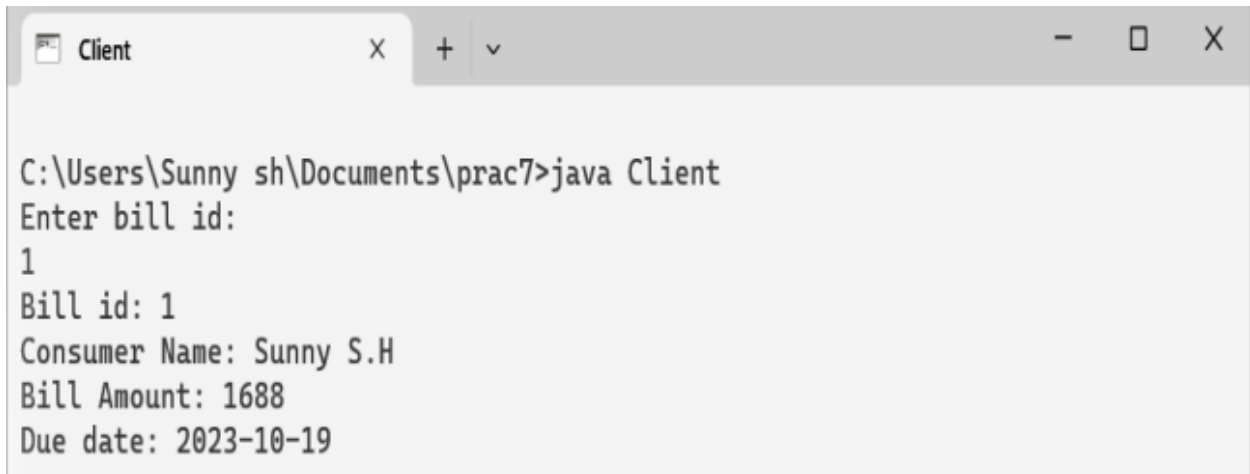
bill_id	consumer_name	bill_due_date	bill_amount
1	Sunny S.H	2023-10-19	1688
2	Siddhartha	2023-10-18	1800
3	Ishwar	2023-10-18	5600

Output Server :



```
C:\Users\Sunny sh\Documents\prac7>javac *.java  
C:\Users\Sunny sh\Documents\prac7>start rmiregistry  
C:\Users\Sunny sh\Documents\prac7>java Server  
Server ready  
Connecting to a selected database...  
Connected database successfully...  
Creating statement...
```

Client :



```
C:\Users\Sunny sh\Documents\prac7>java Client  
Enter bill id:  
1  
Bill id: 1  
Consumer Name: Sunny S.H  
Bill Amount: 1688  
Due date: 2023-10-19
```

Conclusions:

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