

Name of Student: Pushkar Sane		
Roll Number: 45		Lab Assignment Number: 4
Title of Lab Assignment: Implementation of Remote Method Communication using JDBC and RMI.		
DOP: 09-09-2024		DOS: 12-09-2024
CO Mapped:	PO Mapped:	Signature:

Practical No. 4

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

Description: Make use of JDBC and RMI for accessing multiple data access objects.

Theory:

- **Java Database Connectivity (JDBC)**

Java Database Connectivity (JDBC): JDBC is an API in Java that enables applications to interact with relational databases. It allows Java programs to execute SQL queries, retrieve results, and manage database connections. JDBC acts as a bridge between the Java application and the database, providing a standard method for connecting to and operating on databases like MySQL, Oracle, and others.

- **Steps to Connect Java JDBC**

To connect Java to a database using JDBC, the following steps are typically followed:

1. Load the JDBC driver class using `Class.forName`.
2. Establish a connection to the database using `DriverManager.getConnection`.
3. Create a `Statement` or `PreparedStatement` object to execute SQL queries.
4. Execute the query and process the results using `ResultSet`.
5. Finally, close the `ResultSet`, `Statement`, and `Connection` objects to release resources.

- **RMI (Remote Method Invocation)**

RMI is a Java API that allows objects residing in different Java Virtual Machines (JVMs) to communicate with each other. It enables the invocation of methods on a remote object as if it were a local object. RMI abstracts the complexities of remote communication, making it easy to build distributed applications where methods can be invoked across a network.

- **Stub**

In RMI, a stub is a client-side proxy object that represents the remote object. When a client invokes a method on the stub, the stub handles the communication with the remote server, sending the method call over the network, and receiving the response, making the remote method invocation appear seamless to the client.

- **Skeleton**

A skeleton was a server-side component that acted as a gateway between the RMI runtime and the actual remote object implementation. It received incoming requests from the stub, deserialized the parameters, invoked the corresponding method on the remote object, and sent back the results. In later versions of Java, the skeleton functionality was incorporated into the RMI runtime, making the skeleton class unnecessary.

Add the following code in pom.xml for both questions:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>DSCC4</groupId>
  <artifactId>DSCC4</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <dependencies>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.33</version>
    </dependency>
  </dependencies>
</project>
```

1. Using MySQL, create a Library database. Create table Book (Book_id, Book_name, Book_author) and retrieve the Book information from the Library database.

Code:

SQL Code:

```
CREATE DATABASE Library;
```

```
USE Library;
```

```
CREATE TABLE Book (
```

```
  Book_id INT AUTO_INCREMENT PRIMARY KEY,
```

```
  Book_name VARCHAR(100) NOT NULL,
```

```
Book_author VARCHAR(100)
);
INSERT INTO Book (Book_name, Book_author)
VALUES
('To Kill a Mockingbird', 'Harper Lee'),
('1984', 'George Orwell'),
('The Great Gatsby', 'F. Scott Fitzgerald');
```

Book.java

```
import java.io.Serializable;
public class Book implements Serializable {
    private int id;
    private String name;
    private String author;
    public Book(int id, String name, String author) {
        this.id = id;
        this.name = name;
        this.author = author;
    }
    // Getters and toString method
    public int getId() {
        return id;
    }
    public String getName() {
        return name;
    }
    public String getAuthor() {
        return author;
    }
    @Override
    public String toString() {
        return "Book{" +
            "id=" + id +
            ", name=" + name + "\"" +
            ", author=" + author + "\"" +
            '}';
    }
}
```

```
    }  
}
```

LibraryClient.java

```
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:1099/LibraryService");  
            List<Book> books = libraryService.getBooks();  
            for (Book book : books) {  
                System.out.println(book);  
            }  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

LibraryServer.java

```
import java.rmi.Naming;  
import java.rmi.registry.LocateRegistry;  
public class LibraryServer {  
    public static void main(String[] args) {  
        try {  
            LocateRegistry.createRegistry(1099); // Default RMI port  
            LibraryService libraryService = new LibraryServiceImpl();  
            Naming.rebind("rmi://localhost:1099/LibraryService", libraryService);  
            System.out.println("Library RMI Server is running...");  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

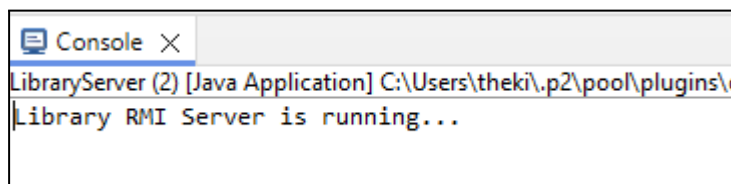
LibraryService.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;
public interface LibraryService extends Remote {
    List<Book> getBooks() throws RemoteException;
}
```

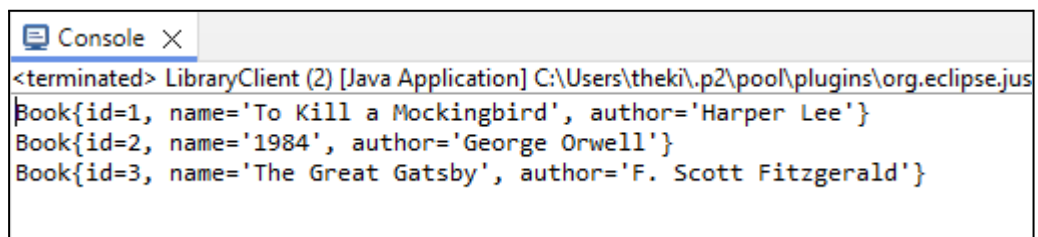
LibraryServiceImpl.java

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
public class LibraryServiceImpl extends UnicastRemoteObject implements
LibraryService {
    protected LibraryServiceImpl() throws RemoteException {
        super();
    }
    @Override
    public List<Book> getBooks() throws RemoteException {
        List<Book> books = new ArrayList<>();
        try {
            // Connect to the MySQL database
            Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/Library", "root", "");
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM Book");
            // Retrieve book information
            while (rs.next()) {
                int id = rs.getInt("Book_id");
                String name = rs.getString("Book_name");
                String author = rs.getString("Book_author");
```

```
        books.add(new Book(id, name, author));
    }
    conn.close();
} catch (Exception e) {
    e.printStackTrace();
}
return books;
}
}
```

Output:

Console X
LibraryServer (2) [Java Application] C:\Users\theki\.p2\pool\plugins\...
Library RMI Server is running...



Console X
<terminated> LibraryClient (2) [Java Application] C:\Users\theki\.p2\pool\plugins\org.eclipse.j...
Book{id=1, name='To Kill a Mockingbird', author='Harper Lee'}
Book{id=2, name='1984', author='George Orwell'}
Book{id=3, name='The Great Gatsby', author='F. Scott Fitzgerald'}

2. Using MySQL, create an Electric_Bill database. Create table Bill. (bill_id, consumer_name, bill_due_date, bill_amount) and retrieve the Bill information from the Electric_Bill database.

Code:**SQL Code:**

```
CREATE DATABASE Electric_Bill;
USE Electric_Bill;
CREATE TABLE Bill (
    bill_id INT AUTO_INCREMENT PRIMARY KEY,
    consumer_name VARCHAR(100) NOT NULL,
    bill_due_date DATE NOT NULL,
    bill_amount DECIMAL(10, 2) NOT NULL
);
INSERT INTO Bill (consumer_name, bill_due_date, bill_amount)
VALUES
('John Doe', '2024-09-30', 150.75),
```

```
('Jane Smith', '2024-10-15', 250.5),  
('Alice Johnson', '2024-10-20', 325.4);
```

Bill.java

```
import java.io.Serializable;  
  
public class Bill implements Serializable {  
    private int billId;  
    private String consumerName;  
    private String billDueDate;  
    private double billAmount;  
    public Bill(int billId, String consumerName, String billDueDate, double  
billAmount) {  
        this.billId = billId;  
        this.consumerName = consumerName;  
        this.billDueDate = billDueDate;  
        this.billAmount = billAmount;  
    }  
    // Getters and toString method  
    public int getBillId() {  
        return billId;  
    }  
    public String getConsumerName() {  
        return consumerName;  
    }  
    public String getBillDueDate() {  
        return billDueDate;  
    }  
    public double getBillAmount() {  
        return billAmount;  
    }  
    @Override  
    public String toString() {  
        return "Bill{" + "billId=" + billId + ", consumerName=" +  
consumerName + "\" + ", billDueDate=" + billDueDate  
        + "\" + ", billAmount=" + billAmount + "}";  
    }  
}
```



```
}
```

ElectricBillClient.java

```
import java.rmi.Naming;
import java.util.List;
public class ElectricBillClient {
    public static void main(String[] args) {
        try {
            ElectricBillService billService = (ElectricBillService) Naming
                .lookup("rmi://localhost:1099/ElectricBillService");
            List<Bill> bills = billService.getBills();
            for (Bill bill : bills) {
                System.out.println(bill);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

ElectricBillServer.java

```
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
public class ElectricBillServer {
    public static void main(String[] args) {
        try {
            LocateRegistry.createRegistry(1099); // Default RMI port
            ElectricBillService billService = new ElectricBillServiceImpl();
            Naming.rebind("rmi://localhost:1099/ElectricBillService",
billService);

            System.out.println("Electric Bill RMI Server is running...");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

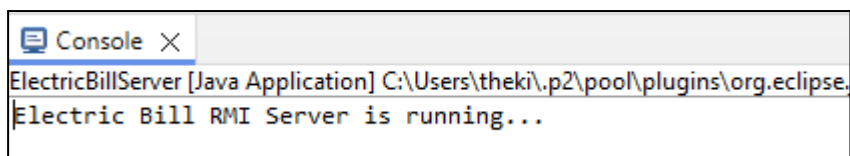
ElectricBillService.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;
public interface ElectricBillService extends Remote {
    List<Bill> getBills() throws RemoteException;
}
```

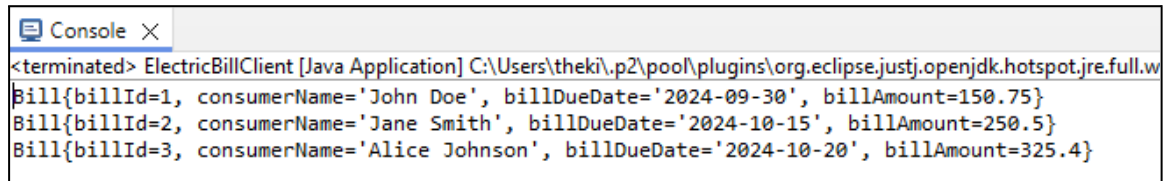
ElectricBillServiceImpl.java

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
public class ElectricBillServiceImpl extends UnicastRemoteObject implements
ElectricBillService {
    protected ElectricBillServiceImpl() throws RemoteException {
        super();
    }
    @Override
    public List<Bill> getBills() throws RemoteException {
        List<Bill> bills = new ArrayList<>();
        try {
            // Connect to the MySQL database using XAMPP's settings
            Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/Electric_Bill", "root", "");
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM Bill");
            // Retrieve bill information
            while (rs.next()) {
                int id = rs.getInt("bill_id");
                String consumerName =
rs.getString("consumer_name");
```

```
String dueDate = rs.getString("bill_due_date");
double amount = rs.getDouble("bill_amount");
bills.add(new Bill(id, consumerName, dueDate,
amount));
    }
    conn.close();
} catch (Exception e) {
    e.printStackTrace();
}
return bills;
}
}
```

Output:

Console X
ElectricBillServer [Java Application] C:\Users\theki\.p2\pool\plugins\org.eclipse.
Electric Bill RMI Server is running...



Console X
<terminated> ElectricBillClient [Java Application] C:\Users\theki\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.w
Bill{billId=1, consumerName='John Doe', billDueDate='2024-09-30', billAmount=150.75}
Bill{billId=2, consumerName='Jane Smith', billDueDate='2024-10-15', billAmount=250.5}
Bill{billId=3, consumerName='Alice Johnson', billDueDate='2024-10-20', billAmount=325.4}

Conclusion:

Successfully demonstrated the Implementation of Remote Method Communication using JDBC and RMI.