BIKATHI MARTIN MUNYOLE
ADM 20/03380

ADVANCED JAVA PROGRAMMING ASSIGNMENT

FEBRUARY 2023

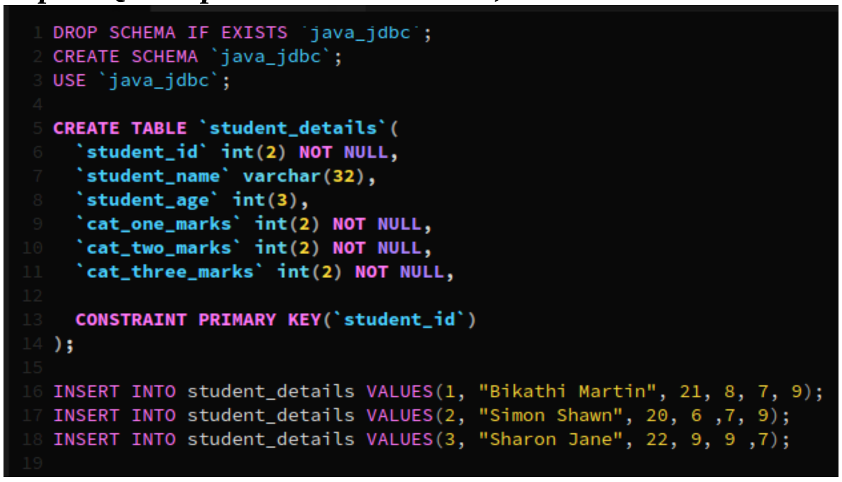
Using real world business applications, write programs to facilitate the following:

Java Database Connectivity

Java Database Connectivity or **JDBC** is a Java API(Application Programming Interface) used to connect with any table-like database, also called a **relational database**. It can be used to access databases like Oracle Database, MySQL or PostgreSQL. It is defined as part of the Java Standard Library and the APIs are provided by the ***java.sql.**** packages.

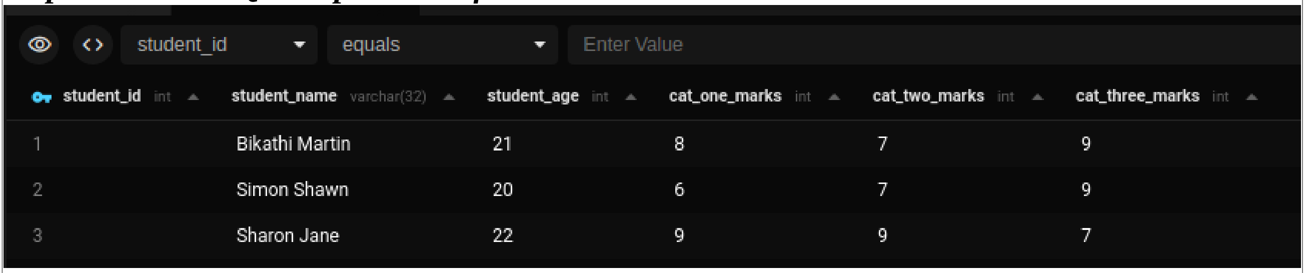
This simple example reads some **Student** records from a MySQL database and prints them on the console.

Step 1: Create a simple SQL script to create a database, a table and some records in the table



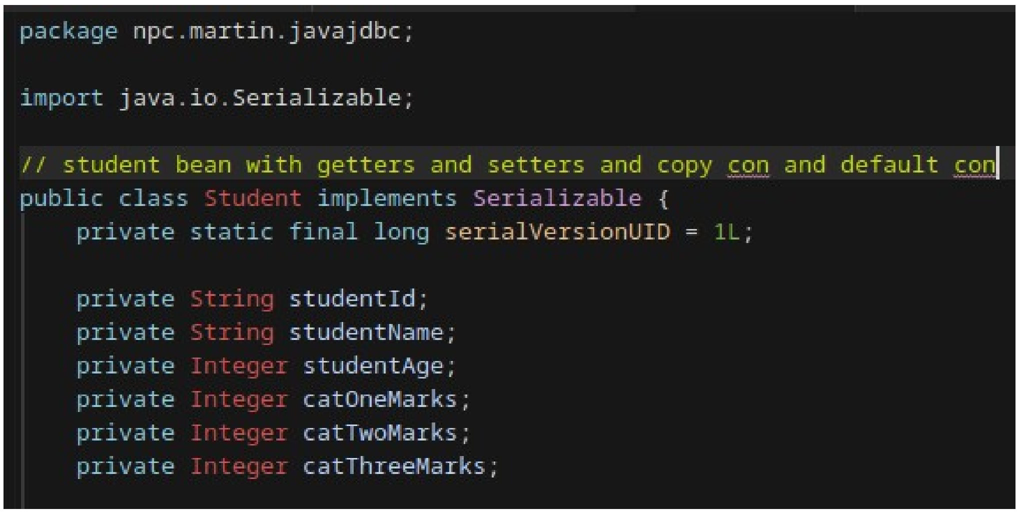
```sql
DROP SCHEMA IF EXISTS `java_jdbc`;
CREATE SCHEMA `java_jdbc`;
USE `java_jdbc`;

CREATE TABLE `student_details`(
  `student_id` int(2) NOT NULL,
  `student_name` varchar(32),
  `student_age` int(3),
  `cat_one_marks` int(2) NOT NULL,
  `cat_two_marks` int(2) NOT NULL,
  `cat_three_marks` int(2) NOT NULL,

  CONSTRAINT PRIMARY KEY(`student_id`)
);

INSERT INTO student_details VALUES(1, "Bikathi Martin", 21, 8, 7, 9);
INSERT INTO student_details VALUES(2, "Simon Shawn", 20, 6 ,7, 9);
INSERT INTO student_details VALUES(3, "Sharon Jane", 22, 9, 9 ,7);
```

Step 2: Run the SQL script and confirm the table has been created



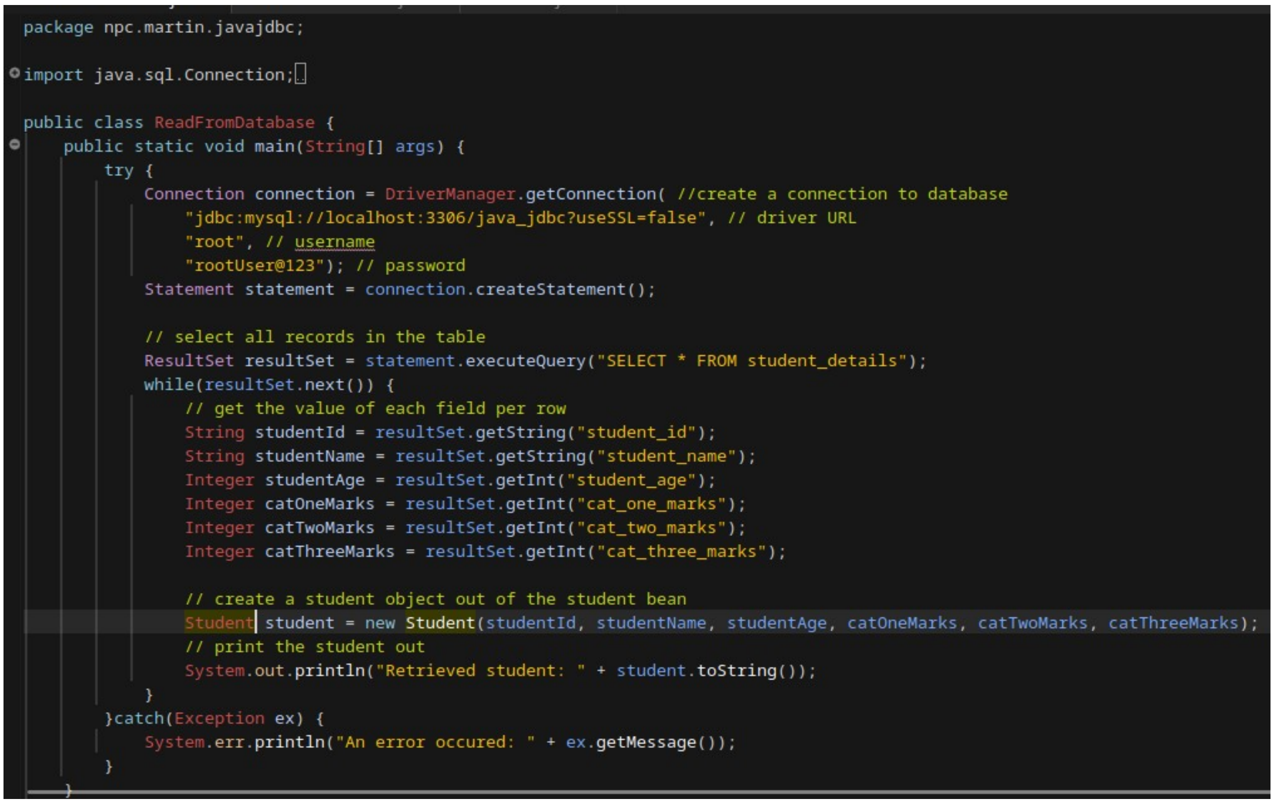
student_id int	student_name varchar(32)	student_age int	cat_one_marks int	cat_two_marks int	cat_three_marks int
1	Bikathi Martin	21	8	7	9
2	Simon Shawn	20	6	7	9
3	Sharon Jane	22	9	9	7

Step 3: Create a Java POJO(Plain Old Java Object) Bean that resembles the 'student_details' table we created in Step 2



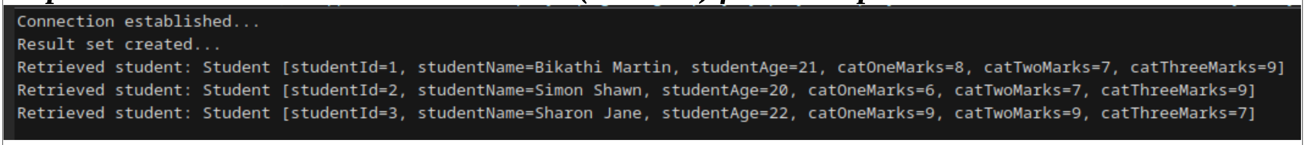
```java
package npc.martin.javajdbc;

import java.io.Serializable;

// student bean with getters and setters and copy con and default con
public class Student implements Serializable {
    private static final long serialVersionUID = 1L;

    private String studentId;
    private String studentName;
    private Integer studentAge;
    private Integer catOneMarks;
    private Integer catTwoMarks;
    private Integer catThreeMarks;
```

Step 4: Create JDBC code to access the Student records from the table and print them out to the console



```java
package npc.martin.javajdbc;

import java.sql.Connection;

public class ReadFromDatabase {
    public static void main(String[] args) {
        try {
            Connection connection = DriverManager.getConnection( //create a connection to database
                "jdbc:mysql://localhost:3306/java_jdbc?useSSL=false", // driver URL
                "root", // username
                "rootUser@123"); // password
            Statement statement = connection.createStatement();

            // select all records in the table
            ResultSet resultSet = statement.executeQuery("SELECT * FROM student_details");
            while(resultSet.next()) {
                // get the value of each field per row
                String studentId = resultSet.getString("student_id");
                String studentName = resultSet.getString("student_name");
                Integer studentAge = resultSet.getInt("student_age");
                Integer catOneMarks = resultSet.getInt("cat_one_marks");
                Integer catTwoMarks = resultSet.getInt("cat_two_marks");
                Integer catThreeMarks = resultSet.getInt("cat_three_marks");

                // create a student object out of the student bean
                Student student = new Student(studentId, studentName, studentAge, catOneMarks, catTwoMarks, catThreeMarks);
                // print the student out
                System.out.println("Retrieved student: " + student.toString());
            }
        }catch(Exception ex) {
            System.err.println("An error occured: " + ex.getMessage());
        }
    }
```

Step 5: Run the code and check the console(terminal) for the output



```
Connection established...
Result set created...
Retrieved student: Student [studentId=1, studentName=Bikathi Martin, studentAge=21, catOneMarks=8, catTwoMarks=7, catThreeMarks=9]
Retrieved student: Student [studentId=2, studentName=Simon Shawn, studentAge=20, catOneMarks=6, catTwoMarks=7, catThreeMarks=9]
Retrieved student: Student [studentId=3, studentName=Sharon Jane, studentAge=22, catOneMarks=9, catTwoMarks=9, catThreeMarks=7]
```

<u>Java Beans</u>

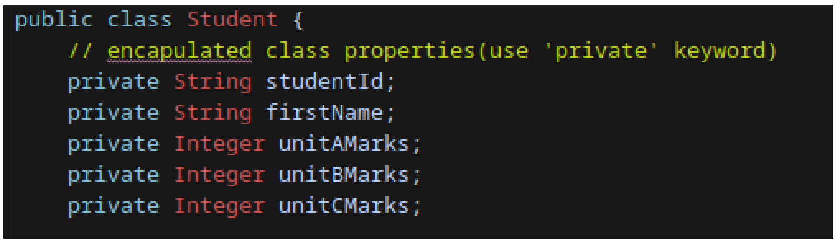
Java beans are Java classes that meet the following conditions:

- They have a no-argument constructor
- It implements the serializable interface
- It provides methods to get and set the properties of the class, known as **getters** and **setters**

<u>Example:</u>

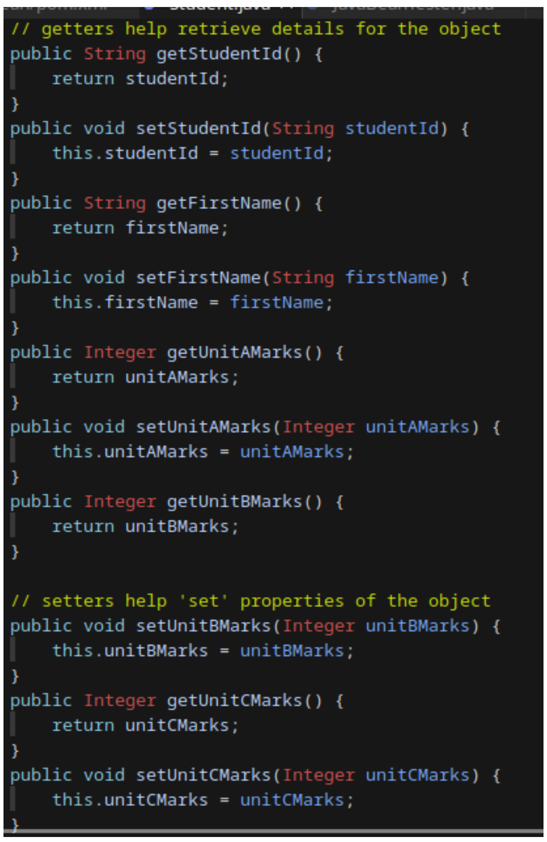
In the simple example below, we create a **Student** bean that we can award some marks and print them out to the console.

Step 1: Create the Student Bean



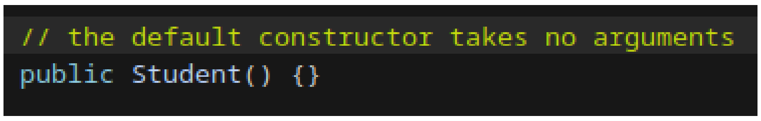
```java
public class Student {
    // encapulated class properties(use 'private' keyword)
    private String studentId;
    private String firstName;
    private Integer unitAMarks;
    private Integer unitBMarks;
    private Integer unitCMarks;
```

Step 2: Create getters and setters for the bean



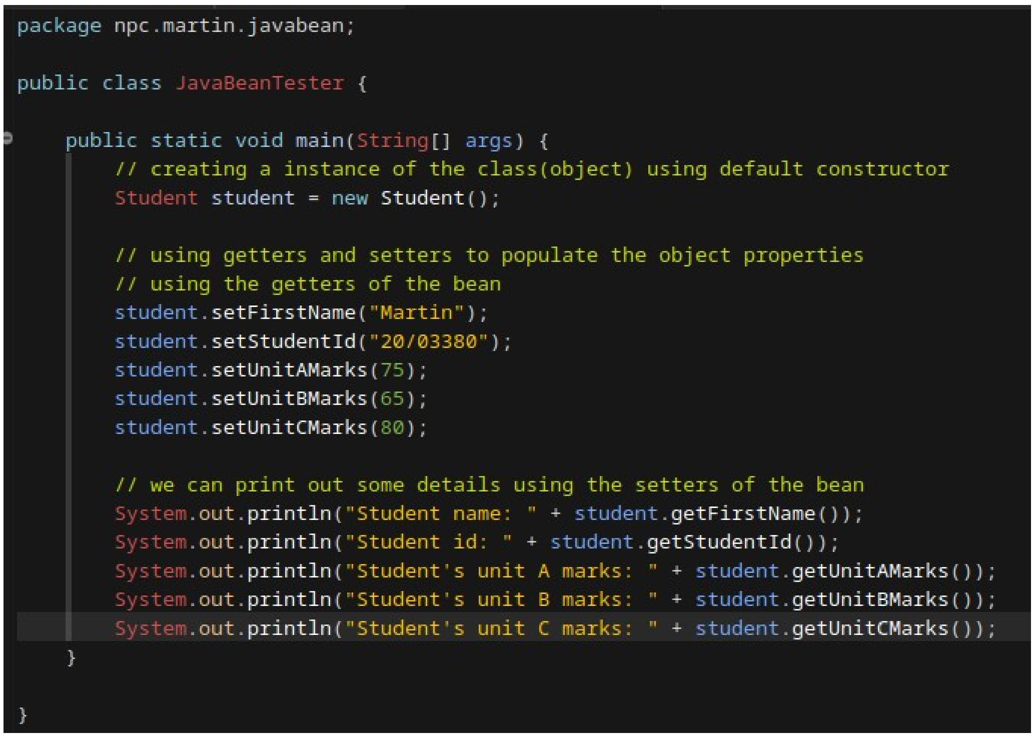
```java
// getters help retrieve details for the object
public String getStudentId() {
    return studentId;
}
public void setStudentId(String studentId) {
    this.studentId = studentId;
}
public String getFirstName() {
    return firstName;
}
public void setFirstName(String firstName) {
    this.firstName = firstName;
}
public Integer getUnitAMarks() {
    return unitAMarks;
}
public void setUnitAMarks(Integer unitAMarks) {
    this.unitAMarks = unitAMarks;
}
public Integer getUnitBMarks() {
    return unitBMarks;
}

// setters help 'set' properties of the object
public void setUnitBMarks(Integer unitBMarks) {
    this.unitBMarks = unitBMarks;
}
public Integer getUnitCMarks() {
    return unitCMarks;
}
public void setUnitCMarks(Integer unitCMarks) {
    this.unitCMarks = unitCMarks;
}
```

Step 3: Add the default constructor



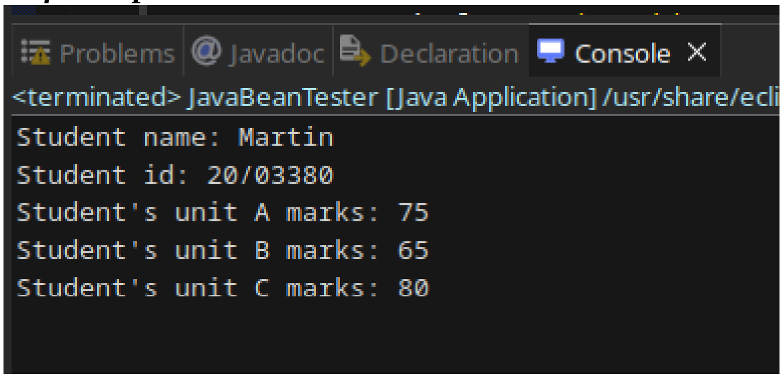
```java
// the default constructor takes no arguments
public Student() {}
```

Step 4: Create an instance of the Student class using the default constructor and use the getters and setters on it



```java
package npc.martin.javabean;

public class JavaBeanTester {

    public static void main(String[] args) {
        // creating a instance of the class(object) using default constructor
        Student student = new Student();

        // using getters and setters to populate the object properties
        // using the getters of the bean
        student.setFirstName("Martin");
        student.setStudentId("20/03380");
        student.setUnitAMarks(75);
        student.setUnitBMarks(65);
        student.setUnitCMarks(80);

        // we can print out some details using the setters of the bean
        System.out.println("Student name: " + student.getFirstName());
        System.out.println("Student id: " + student.getStudentId());
        System.out.println("Student's unit A marks: " + student.getUnitAMarks());
        System.out.println("Student's unit B marks: " + student.getUnitBMarks());
        System.out.println("Student's unit C marks: " + student.getUnitCMarks());
    }

}
```

Step 5: Check console for output:

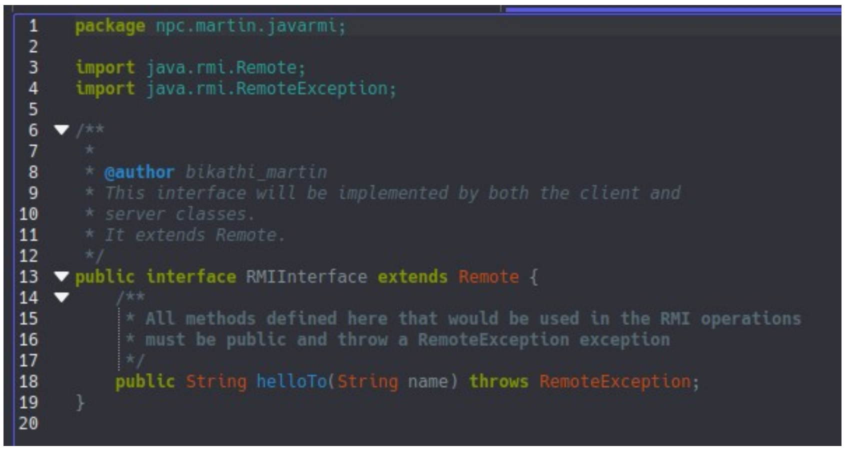


```
<terminated> JavaBeanTester [Java Application] /usr/share/ecli
Student name: Martin
Student id: 20/03380
Student's unit A marks: 75
Student's unit B marks: 65
Student's unit C marks: 80
```

<u>Remote Method Invocation</u>

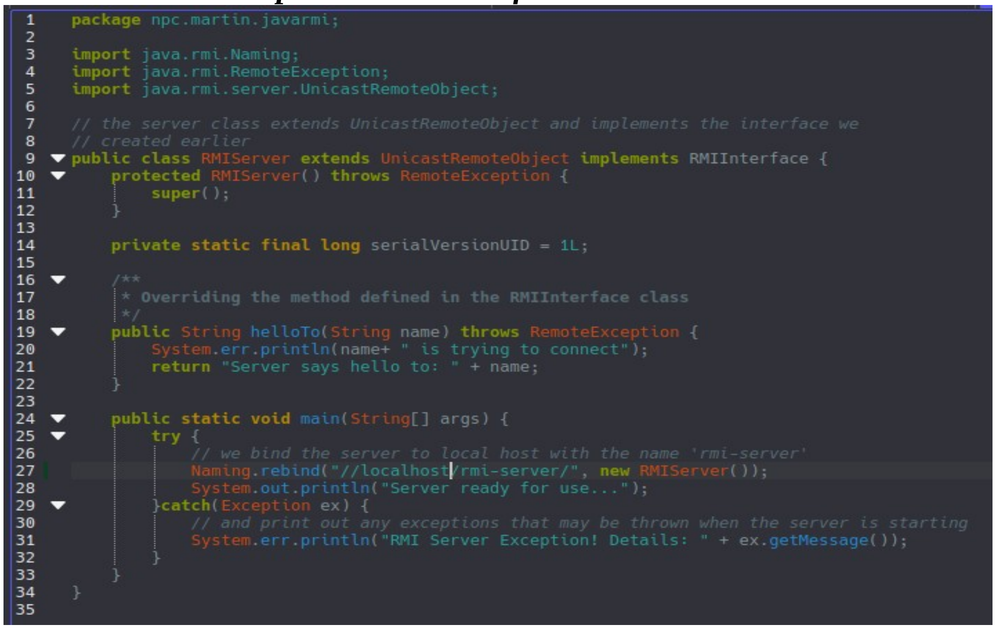
Java RMI(**Remote Method Invocation**) is a technology that allows the methods of one Object Oriented Programming language make calls and receive responses from methods of another class running on a different machine. It is the OOP equivalent of RPCs(**Remote Procedure Calls**) that is used by functional programming languages like C.

Step 1: Create an interface to house all the methods exposed in the RMI communication



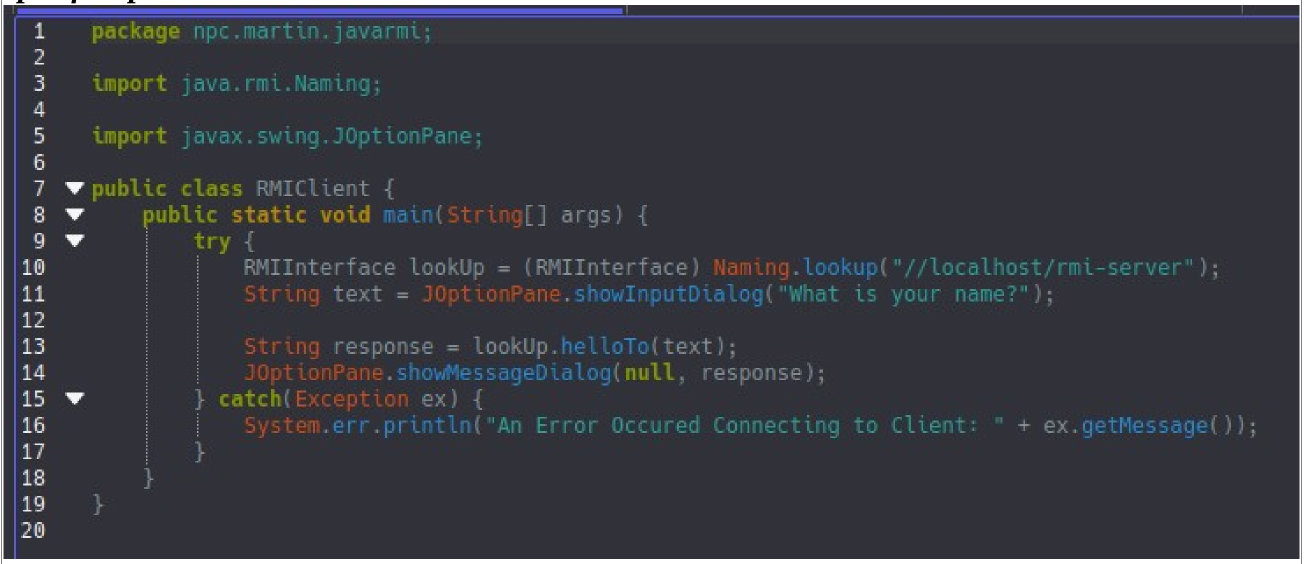
```java
package npc.martin.javarmi;

import java.rmi.Remote;
import java.rmi.RemoteException;

/**
 *
 * @author bikathi_martin
 * This interface will be implemented by both the client and
 * server classes.
 * It extends Remote.
 */
public interface RMIInterface extends Remote {
    /**
     * All methods defined here that would be used in the RMI operations
     * must be public and throw a RemoteException exception
     */
    public String helloTo(String name) throws RemoteException;
}

```

Step 2: Create the server that implements the interface



```java
package npc.martin.javarmi;

import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

// the server class extends UnicastRemoteObject and implements the interface we
// created earlier
public class RMIServer extends UnicastRemoteObject implements RMIInterface {
    protected RMIServer() throws RemoteException {
        super();
    }

    private static final long serialVersionUID = 1L;

    /**
     * Overriding the method defined in the RMIInterface class
     */
    public String helloTo(String name) throws RemoteException {
        System.err.println(name+ " is trying to connect");
        return "Server says hello to: " + name;
    }

    public static void main(String[] args) {
        try {
            // we bind the server to local host with the name 'rmi-server'
            Naming.rebind("//localhost/rmi-server/", new RMIServer());
            System.out.println("Server ready for use...");
        }catch(Exception ex) {
            // and print out any exceptions that may be thrown when the server is starting
            System.err.println("RMI Server Exception! Details: " + ex.getMessage());
        }
    }
}

```

Step 3: Create a client that also implements the interface but connects to the server on the specified port



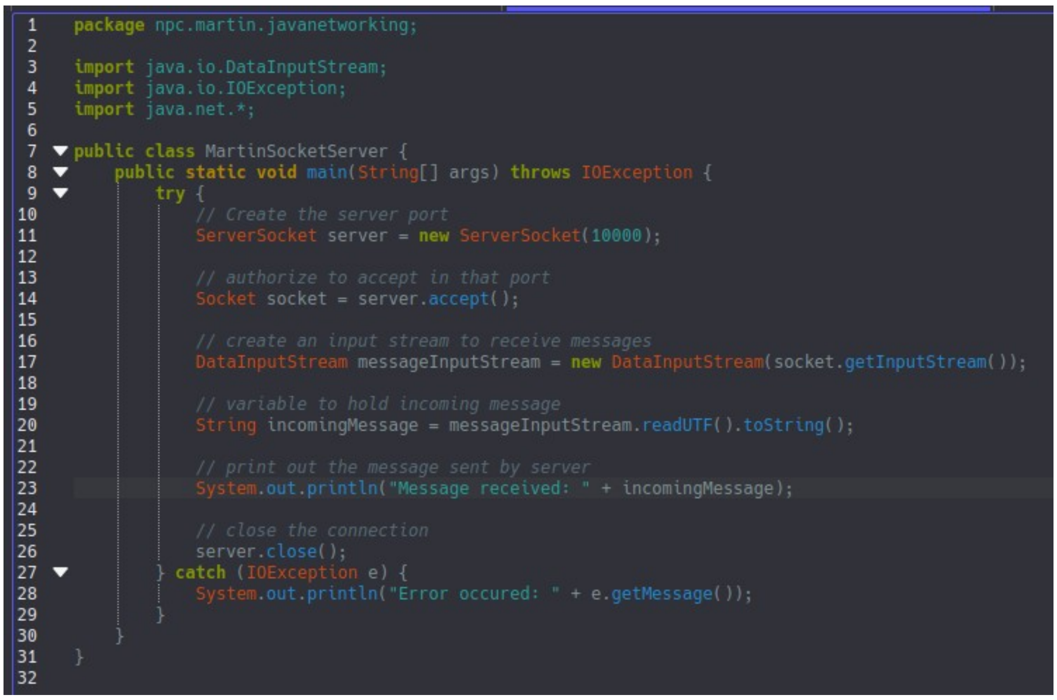
```java
package npc.martin.javarmi;

import java.rmi.Naming;

import javax.swing.JOptionPane;

public class RMIClient {
    public static void main(String[] args) {
        try {
            RMIInterface lookUp = (RMIInterface) Naming.lookup("//localhost/rmi-server");
            String text = JOptionPane.showInputDialog("What is your name?");

            String response = lookUp.helloTo(text);
            JOptionPane.showMessageDialog(null, response);
        } catch(Exception ex) {
            System.err.println("An Error Occured Connecting to Client: " + ex.getMessage());
        }
    }
}

```

Network Programming

Network programming allows two Java programs to communicate via a network, even if they are on different machines. This can be in the form of HTTP packets, WebSockets, TCP connections or other protocols supported by the language.

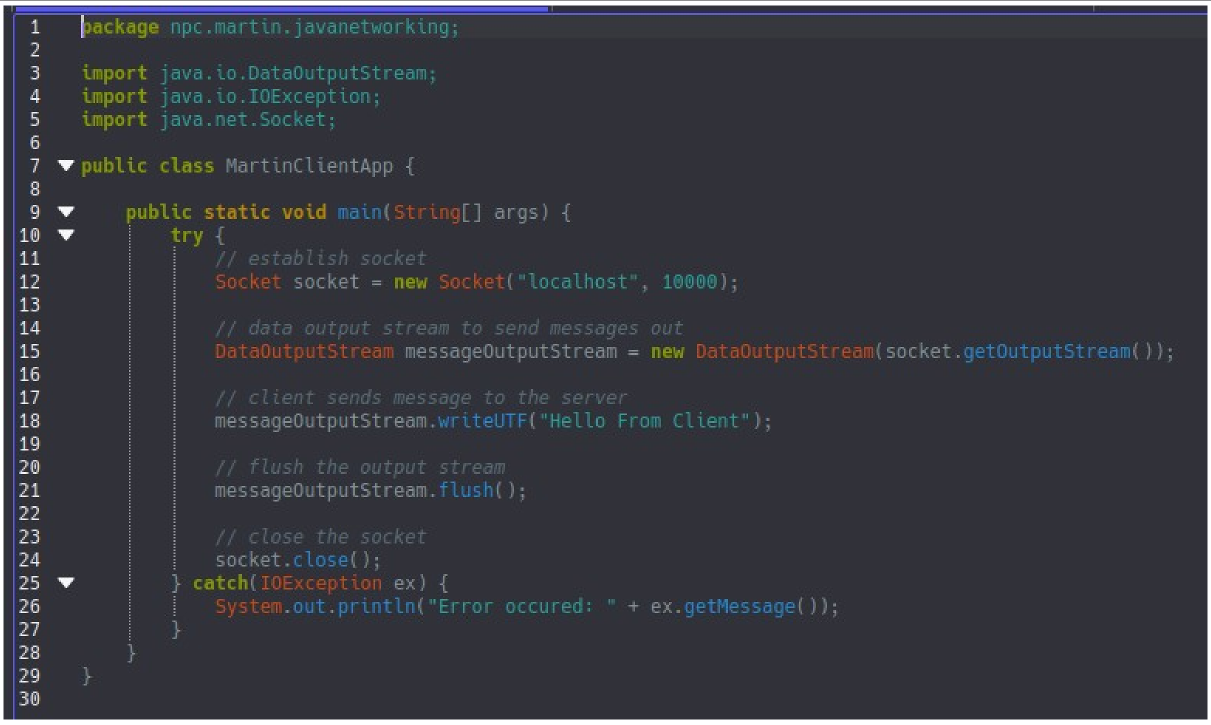
In this example, we have created a simple socket program that runs on the terminal. The server is started and waits for a client program to connect and then displays the message that the client sends to it in the terminal.

Step 1: Server code that establishes the socket connection and a port for clients to connect to



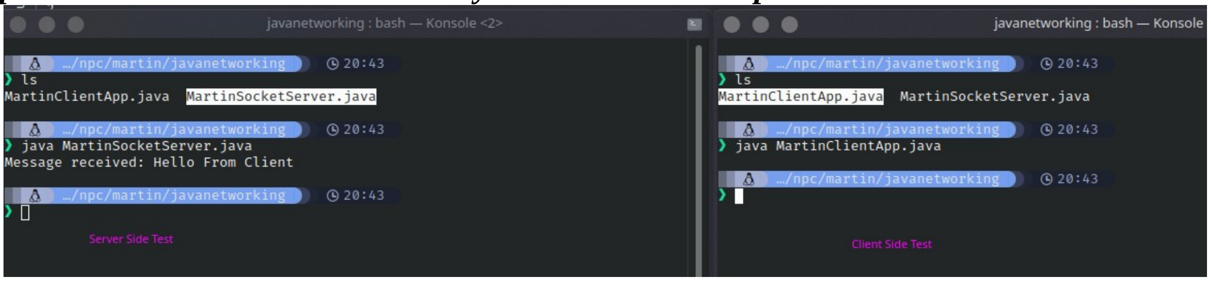
```java
package npc.martin.javanetworking;

import java.io.DataInputStream;
import java.io.IOException;
import java.net.*;

public class MartinSocketServer {
    public static void main(String[] args) throws IOException {
        try {
            // Create the server port
            ServerSocket server = new ServerSocket(10000);

            // authorize to accept in that port
            Socket socket = server.accept();

            // create an input stream to receive messages
            DataInputStream messageInputStream = new DataInputStream(socket.getInputStream());

            // variable to hold incoming message
            String incomingMessage = messageInputStream.readUTF().toString();

            // print out the message sent by server
            System.out.println("Message received: " + incomingMessage);

            // close the connection
            server.close();
        } catch (IOException e) {
            System.out.println("Error occured: " + e.getMessage());
        }
    }
}

```

Step 2: Create client code that connects to the socket port opened by the server



```java
package npc.martin.javanetworking;

import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;

public class MartinClientApp {

    public static void main(String[] args) {
        try {
            // establish socket
            Socket socket = new Socket("localhost", 10000);

            // data output stream to send messages out
            DataOutputStream messageOutputStream = new DataOutputStream(socket.getOutputStream());

            // client sends message to the server
            messageOutputStream.writeUTF("Hello From Client");

            // flush the output stream
            messageOutputStream.flush();

            // close the socket
            socket.close();
        } catch(IOException ex) {
            System.out.println("Error occured: " + ex.getMessage());
        }
    }
}

```

Step 3: Run the server and client side by side and see the output



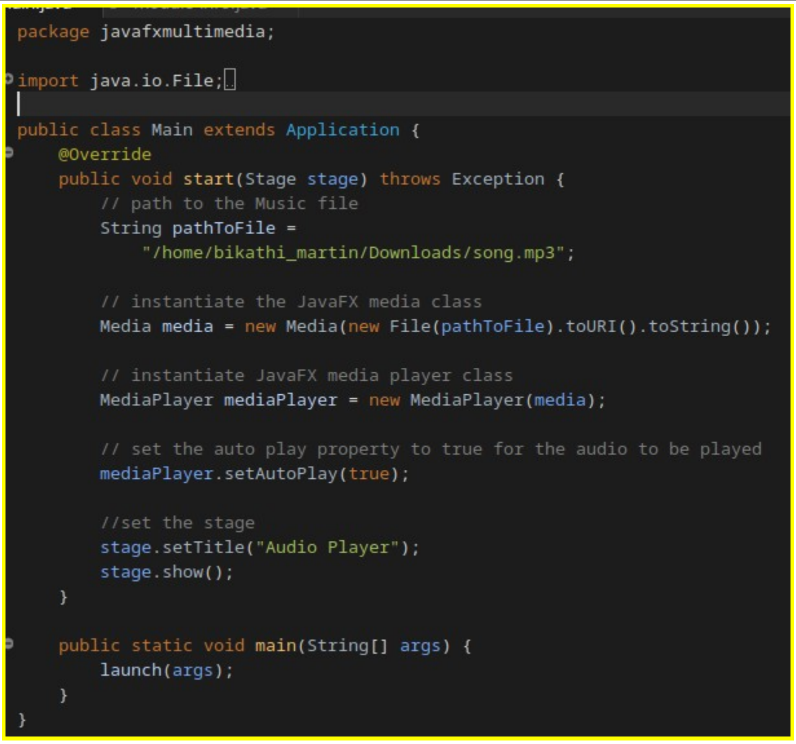
```
javanetworking : bash — Konsole <2>

 …/npc/martin/javanetworking   20:43
❯ ls
MartinClientApp.java  MartinSocketServer.java

 …/npc/martin/javanetworking   20:43
❯ java MartinSocketServer.java
Message received: Hello From Client

 …/npc/martin/javanetworking   20:43
❯
```

```
javanetworking : bash — Konsole

 …/npc/martin/javanetworking   20:43
❯ ls
MartinClientApp.java  MartinSocketServer.java

 …/npc/martin/javanetworking   20:43
❯ java MartinClientApp.java

 …/npc/martin/javanetworking   20:43
❯
```

<u>Multimedia Programming</u>

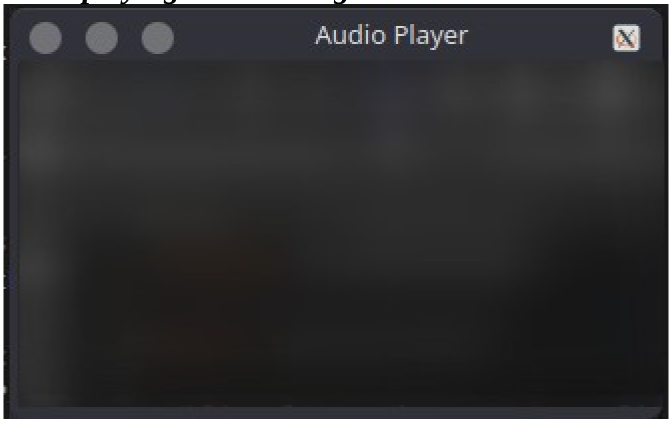
The multimedia library allows a java application to play audio or video media. The easiest way to get an audio playing is by using the JavaFX media classes as in the simple example below:

Step 1: Create the code



```java
package javafxmultimedia;

import java.io.File;

public class Main extends Application {
    @Override
    public void start(Stage stage) throws Exception {
        // path to the Music file
        String pathToFile =
            "/home/bikathi_martin/Downloads/song.mp3";

        // instantiate the JavaFX media class
        Media media = new Media(new File(pathToFile).toURI().toString());

        // instantiate JavaFX media player class
        MediaPlayer mediaPlayer = new MediaPlayer(media);

        // set the auto play property to true for the audio to be played
        mediaPlayer.setAutoPlay(true);

        //set the stage
        stage.setTitle("Audio Player");
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Step 2: Play the media by running the code above. You should see a window similar to this one with the media you linked to playing in the background.


Audio Player