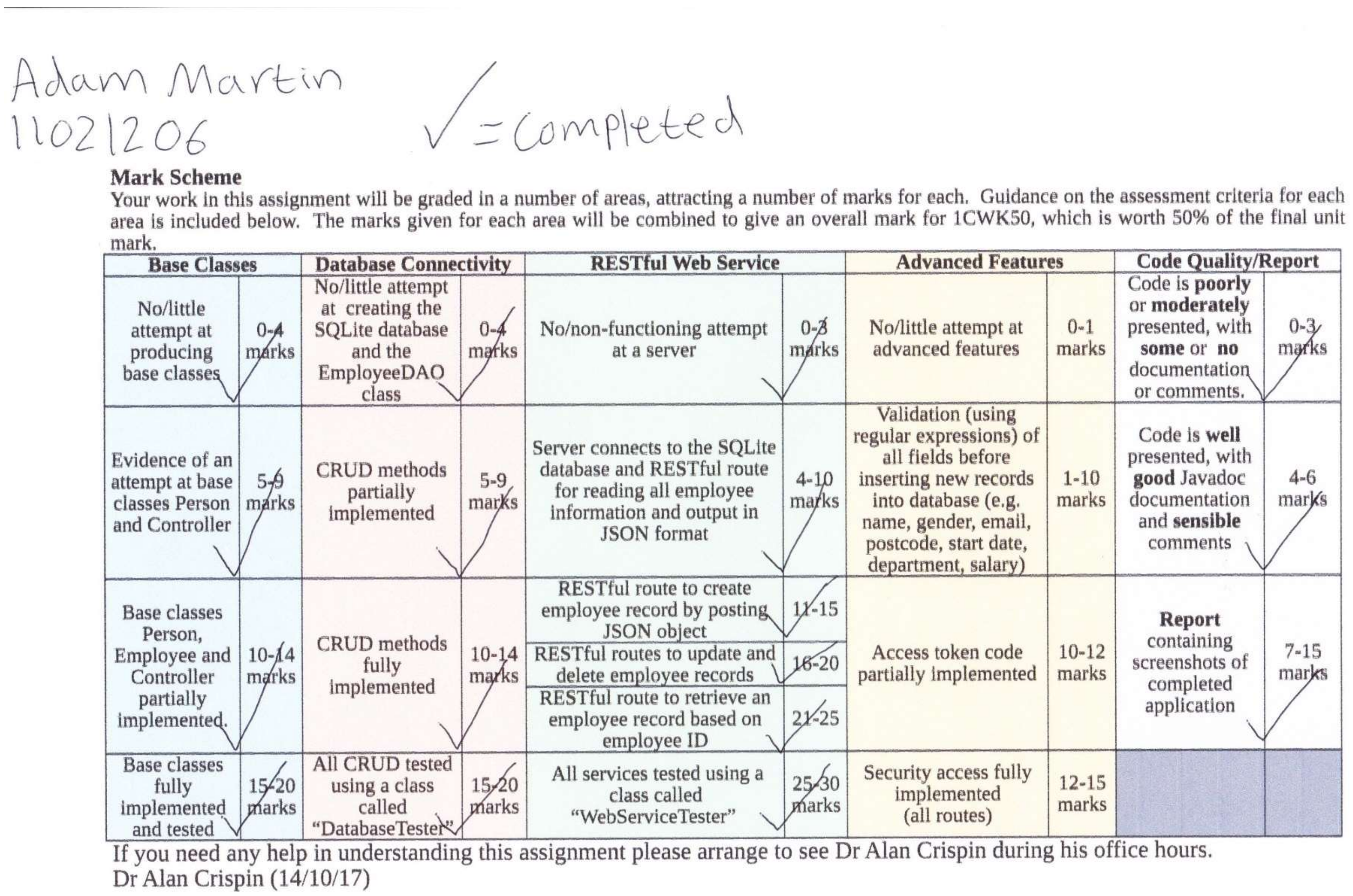
Name: Adam Martin

Student Number: 11021206

Module: Object Oriented and Web Programming

Assignment: Report - Java Employee RESTful Web Service Coursework

Annotated Marking Grid:



Adam Martin
11021206

✓ = completed

Mark Scheme

Your work in this assignment will be graded in a number of areas, attracting a number of marks for each. Guidance on the assessment criteria for each area is included below. The marks given for each area will be combined to give an overall mark for 1CWK50, which is worth 50% of the final unit mark.

Base Classes		Database Connectivity		RESTful Web Service		Advanced Features		Code Quality/Report	
No/little attempt at producing base classes ✓	0-4 marks ✓	No/little attempt at creating the SQLite database and the EmployeeDAO class ✓	0-4 marks ✓	No/non-functioning attempt at a server ✓	0-3 marks ✓	No/little attempt at advanced features	0-1 marks	Code is **poorly** or **moderately** presented, with **some** or **no** documentation or comments. ✓	0-3 marks ✓
Evidence of an attempt at base classes Person and Controller ✓	5-9 marks ✓	CRUD methods partially implemented ✓	5-9 marks ✓	Server connects to the SQLite database and RESTful route for reading all employee information and output in JSON format ✓	4-10 marks ✓	Validation (using regular expressions) of all fields before inserting new records into database (e.g. name, gender, email, postcode, start date, department, salary)	1-10 marks	Code is **well** presented, with **good** Javadoc documentation and **sensible** comments ✓	4-6 marks ✓
Base classes Person, Employee and Controller partially implemented. ✓	10-14 marks ✓	CRUD methods fully implemented ✓	10-14 marks ✓	RESTful route to create employee record by posting JSON object ✓	11-15 ✓	Access token code partially implemented	10-12 marks	**Report** containing screenshots of completed application ✓	7-15 marks ✓
				RESTful routes to update and delete employee records ✓	16-20 ✓				
				RESTful route to retrieve an employee record based on employee ID ✓	21-25 ✓				
Base classes fully implemented and tested ✓	15-20 marks ✓	All CRUD tested using a class called "DatabaseTester" ✓	15-20 marks ✓	All services tested using a class called "WebServiceTester" ✓	25-30 marks ✓	Security access fully implemented (all routes)	12-15 marks		

If you need any help in understanding this assignment please arrange to see Dr Alan Crispin during his office hours.
Dr Alan Crispin (14/10/17)

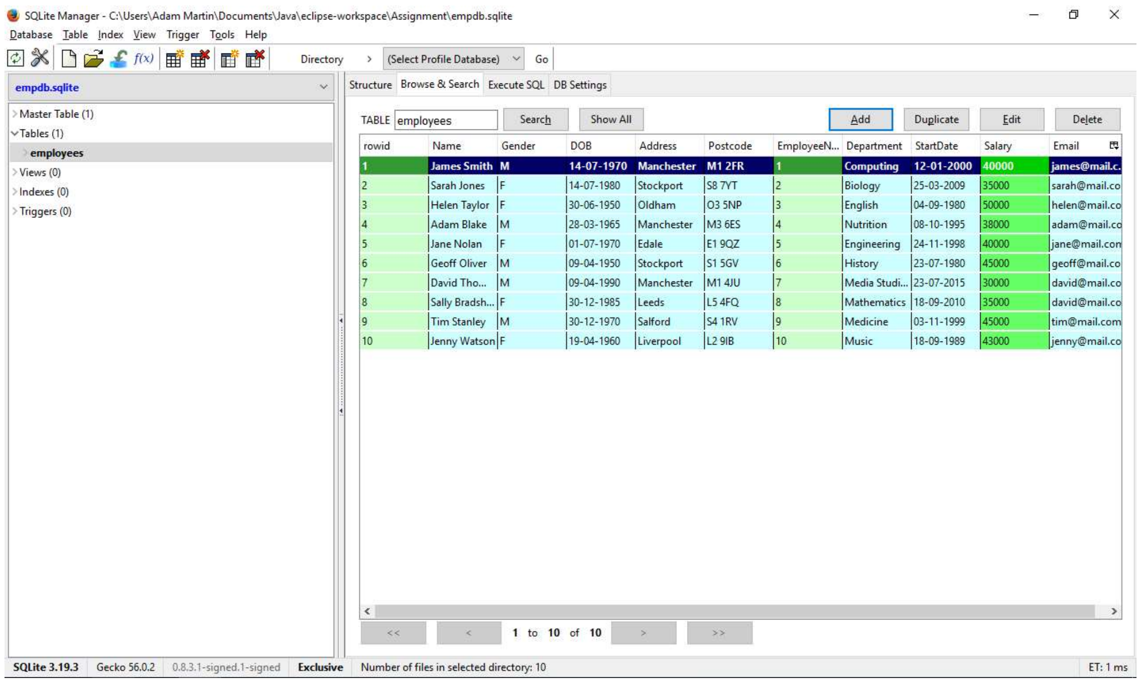
Report – Java Employee RESTful Web Service Coursework

Introduction:

This report will use screenshots and accompanied descriptions to display and describe the functionality of my Java Employee RESTful Web Service project, and the steps taken to implement this functionality. The aim of the project was to develop a Java RESTful web service, which sent requests to a back-end SQLite database of employees.

Functionality, and Steps Taken to Achieve Functionality:

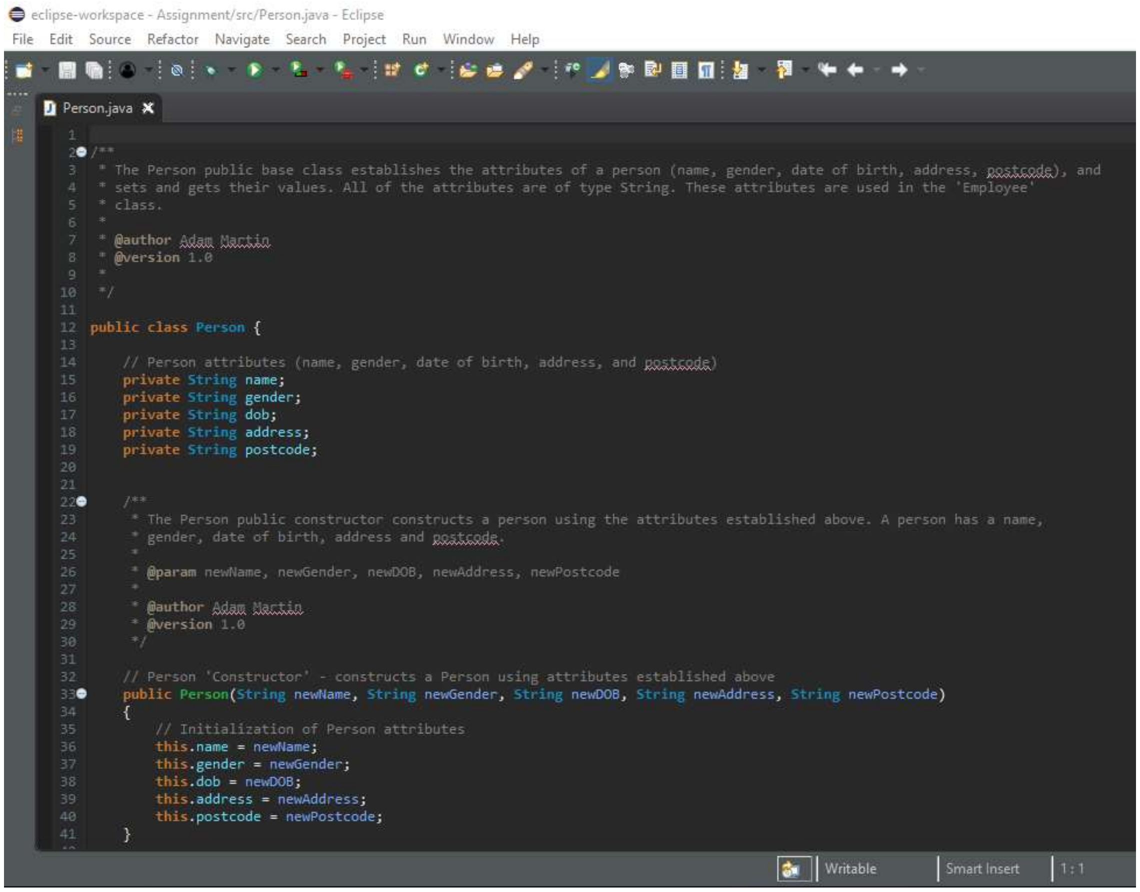
Step 1 – Creation of the 'Employees' SQLite Database:



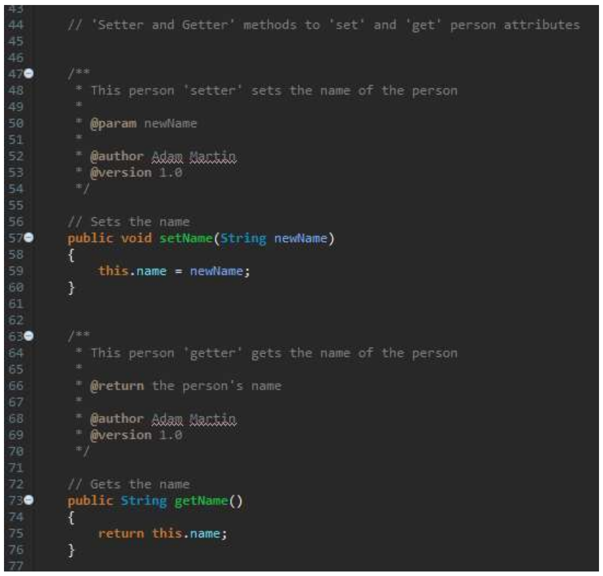
rowid	Name	Gender	DOB	Address	Postcode	EmployeeN...	Department	StartDate	Salary	Email
1	James Smith	M	14-07-1970	Manchester	M1 2FR	1	Computing	12-01-2000	40000	james@mail.c.
2	Sarah Jones	F	14-07-1980	Stockport	S8 7YT	2	Biology	25-03-2009	35000	sarah@mail.co
3	Helen Taylor	F	30-06-1950	Oldham	O3 5NP	3	English	04-09-1980	50000	helen@mail.co
4	Adam Blake	M	28-03-1965	Manchester	M3 6ES	4	Nutrition	08-10-1995	38000	adam@mail.co
5	Jane Nolan	F	01-07-1970	Edale	E1 9QZ	5	Engineering	24-11-1998	40000	jane@mail.con
6	Geoff Oliver	M	09-04-1950	Stockport	S1 5GV	6	History	23-07-1980	45000	geoff@mail.co
7	David Tho...	M	09-04-1990	Manchester	M1 4JU	7	Media Studi...	23-07-2015	30000	david@mail.co
8	Sally Bradsh...	F	30-12-1985	Leeds	L5 4FQ	8	Mathematics	18-09-2010	35000	david@mail.co
9	Tim Stanley	M	30-12-1970	Salford	S4 1RV	9	Medicine	03-11-1999	45000	tim@mail.com
10	Jenny Watson	F	19-04-1960	Liverpool	L2 9IB	10	Music	18-09-1989	43000	jenny@mail.co

The first step was to create the database, using the Firefox 'SQLite Manager' tool. The database contains a single 'Employees' table, populated with 10 fictitious records. In this scenario, the fictitious people are employees of a local university.

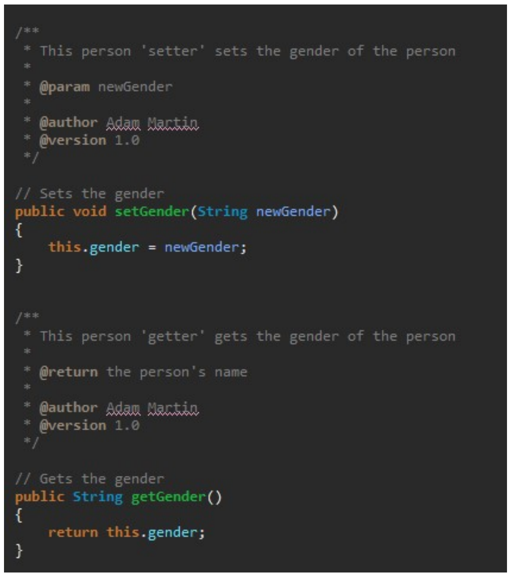
Step 2 – 'Person' Class:


eclipse-workspace - Assignment/src/Person.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Person.java
Writable
Smart Insert
1 : 1

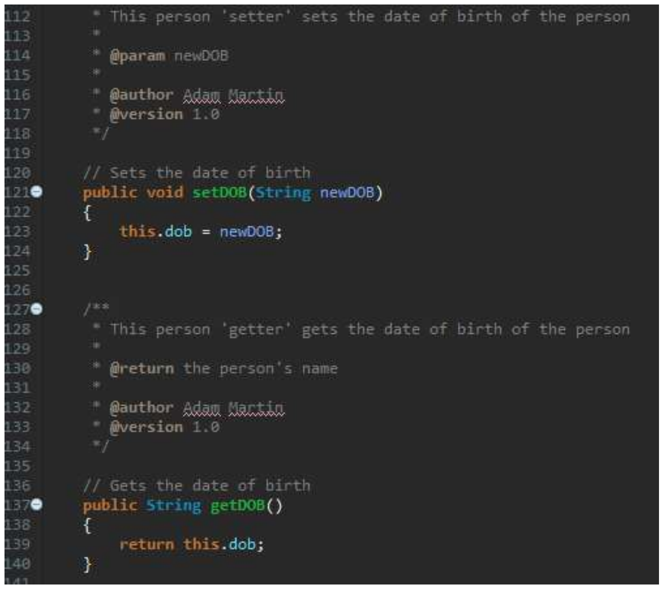
```java

/**
 * The Person public base class establishes the attributes of a person (name, gender, date of birth, address, postcode), and
 * sets and gets their values. All of the attributes are of type String. These attributes are used in the 'Employee'
 * class.
 *
 * @author Adam Martin
 * @version 1.0
 *
 */

public class Person {

    // Person attributes (name, gender, date of birth, address, and postcode)
    private String name;
    private String gender;
    private String dob;
    private String address;
    private String postcode;


    /**
     * The Person public constructor constructs a person using the attributes established above. A person has a name,
     * gender, date of birth, address and postcode.
     *
     * @param newName, newGender, newDOB, newAddress, newPostcode
     *
     * @author Adam Martin
     * @version 1.0
     */

    // Person 'Constructor' - constructs a Person using attributes established above
    public Person(String newName, String newGender, String newDOB, String newAddress, String newPostcode)
    {
        // Initialization of Person attributes
        this.name = newName;
        this.gender = newGender;
        this.dob = newDOB;
        this.address = newAddress;
        this.postcode = newPostcode;
    }
```



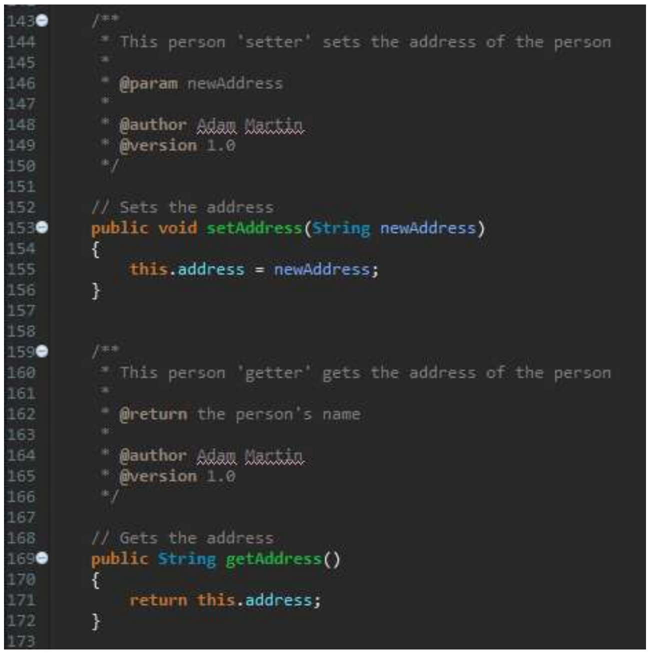
```java
    // 'Setter and Getter' methods to 'set' and 'get' person attributes


    /**
     * This person 'setter' sets the name of the person
     *
     * @param newName
     *
     * @author Adam Martin
     * @version 1.0
     */

    // Sets the name
    public void setName(String newName)
    {
        this.name = newName;
    }


    /**
     * This person 'getter' gets the name of the person
     *
     * @return the person's name
     *
     * @author Adam Martin
     * @version 1.0
     */

    // Gets the name
    public String getName()
    {
        return this.name;
    }
```



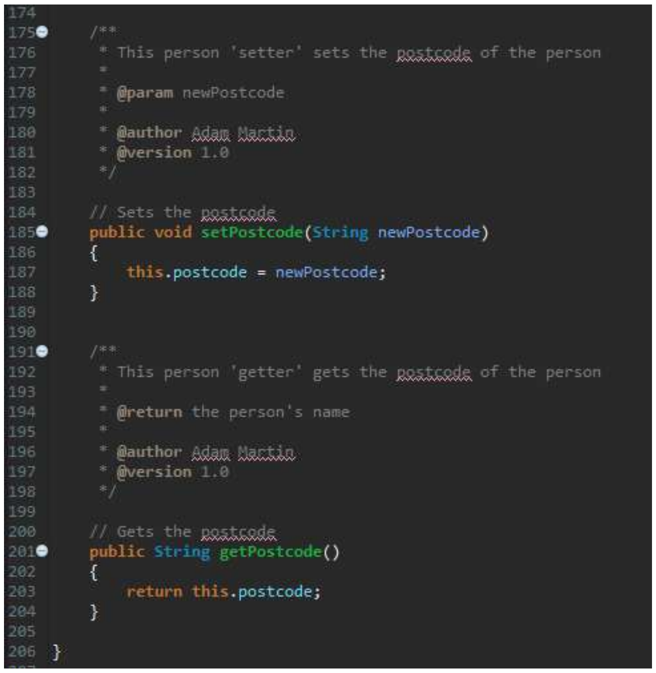
```java
/**
 * This person 'setter' sets the gender of the person
 *
 * @param newGender
 *
 * @author Adam Martin
 * @version 1.0
 */

// Sets the gender
public void setGender(String newGender)
{
    this.gender = newGender;
}

/**
 * This person 'getter' gets the gender of the person
 *
 * @return the person's name
 *
 * @author Adam Martin
 * @version 1.0
 */

// Gets the gender
public String getGender()
{
    return this.gender;
}
```



```
     * This person 'setter' sets the date of birth of the person
     *
     * @param newDOB
     *
     * @author Adam Martin
     * @version 1.0
     */

    // Sets the date of birth
    public void setDOB(String newDOB)
    {
        this.dob = newDOB;
    }


    /**
     * This person 'getter' gets the date of birth of the person
     *
     * @return the person's name
     *
     * @author Adam Martin
     * @version 1.0
     */

    // Gets the date of birth
    public String getDOB()
    {
        return this.dob;
    }
```



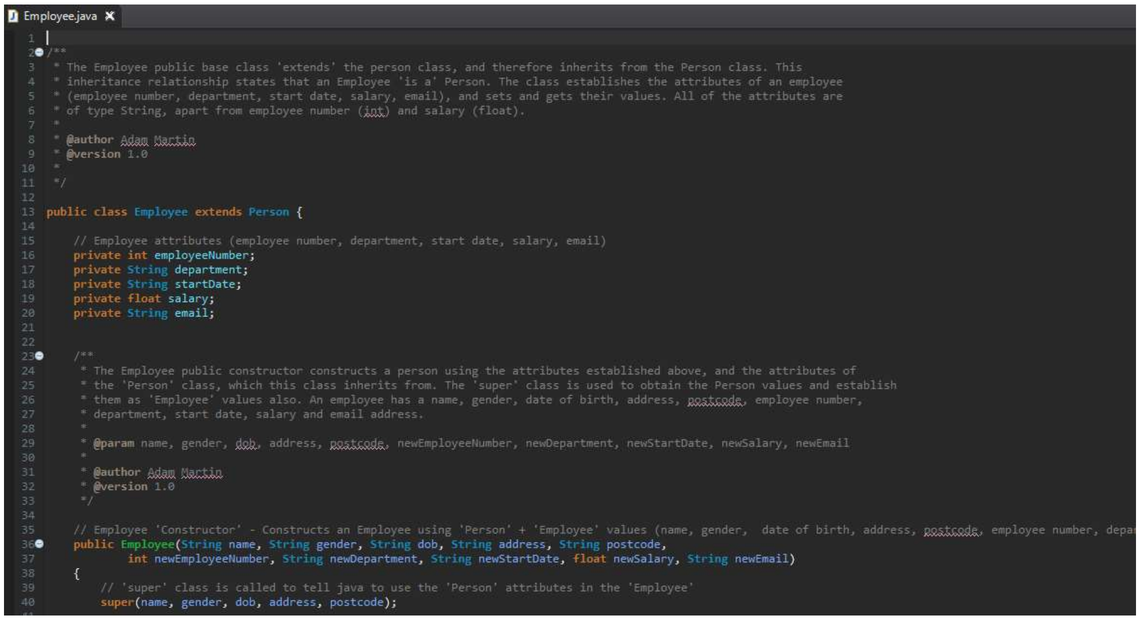
```
    /**
     * This person 'setter' sets the address of the person
     *
     * @param newAddress
     *
     * @author Adam Martin
     * @version 1.0
     */

    // Sets the address
    public void setAddress(String newAddress)
    {
        this.address = newAddress;
    }


    /**
     * This person 'getter' gets the address of the person
     *
     * @return the person's name
     *
     * @author Adam Martin
     * @version 1.0
     */

    // Gets the address
    public String getAddress()
    {
        return this.address;
    }

```



```

    /**
     * This person 'setter' sets the postcode of the person
     *
     * @param newPostcode
     *
     * @author Adam Martin
     * @version 1.0
     */

    // Sets the postcode
    public void setPostcode(String newPostcode)
    {
        this.postcode = newPostcode;
    }


    /**
     * This person 'getter' gets the postcode of the person
     *
     * @return the person's name
     *
     * @author Adam Martin
     * @version 1.0
     */

    // Gets the postcode
    public String getPostcode()
    {
        return this.postcode;
    }

}
```

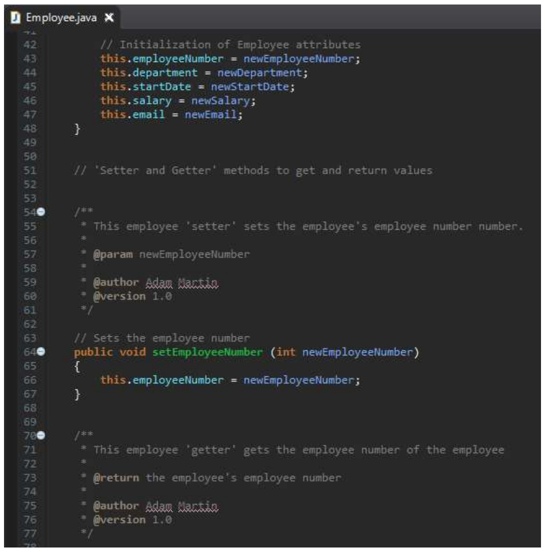
The person base class sets the attributes of a person (name, gender, date of birth, address, postcode), which are all of type string. The class also implements 'setter' and 'getter' methods to set and get the person's values. This class underpins the 'Employee' class, which inherits its values.

Step 3 – 'Employee' Class:



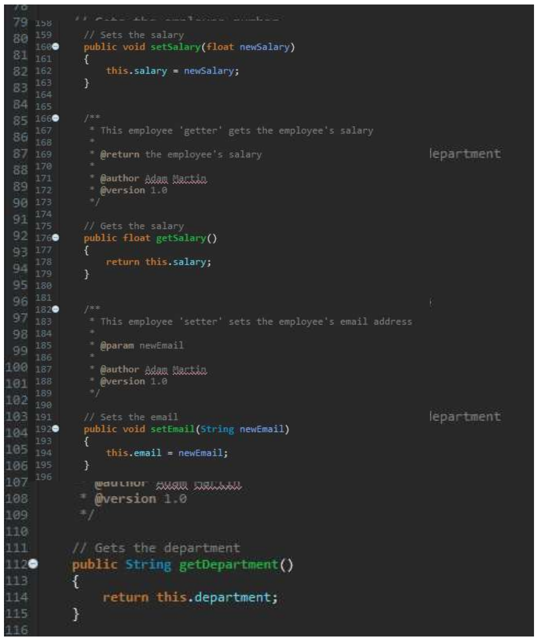
Employee.java

```java

/**
 * The Employee public base class 'extends' the person class, and therefore inherits from the Person class. This
 * inheritance relationship states that an Employee 'is a' Person. The class establishes the attributes of an employee
 * (employee number, department, start date, salary, email), and sets and gets their values. All of the attributes are
 * of type String, apart from employee number (int) and salary (float).
 *
 * @author Adam Martin
 * @version 1.0
 *
 */

public class Employee extends Person {

    // Employee attributes (employee number, department, start date, salary, email)
    private int employeeNumber;
    private String department;
    private String startDate;
    private float salary;
    private String email;


    /**
     * The Employee public constructor constructs a person using the attributes established above, and the attributes of
     * the 'Person' class, which this class inherits from. The 'super' class is used to obtain the Person values and establish
     * them as 'Employee' values also. An employee has a name, gender, date of birth, address, postcode, employee number,
     * department, start date, salary and email address.
     *
     * @param name, gender, dob, address, postcode, newEmployeeNumber, newDepartment, newStartDate, newSalary, newEmail
     *
     * @author Adam Martin
     * @version 1.0
     */

    // Employee 'Constructor' - Constructs an Employee using 'Person' + 'Employee' values (name, gender,  date of birth, address, postcode, employee number, depa
    public Employee(String name, String gender, String dob, String address, String postcode,
            int newEmployeeNumber, String newDepartment, String newStartDate, float newSalary, String newEmail)
    {
        // 'super' class is called to tell java to use the 'Person' attributes in the 'Employee'
        super(name, gender, dob, address, postcode);
```

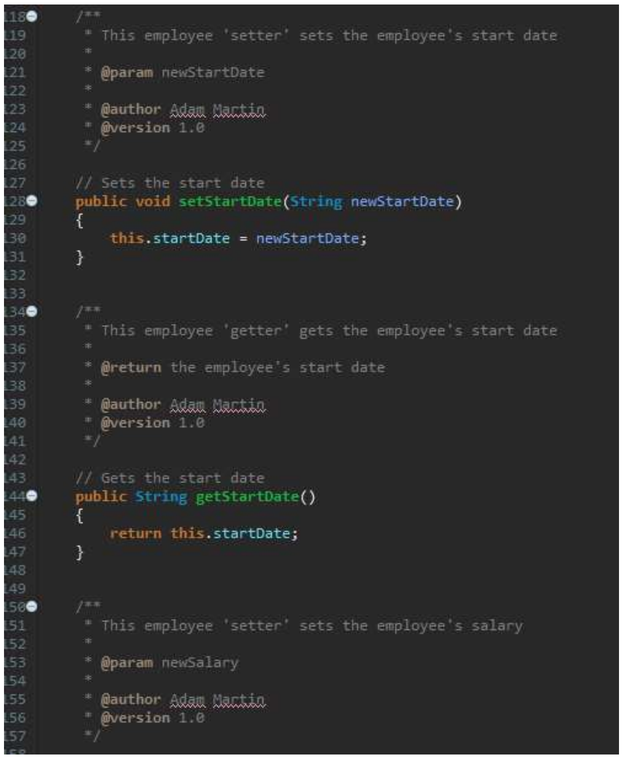


Employee.java

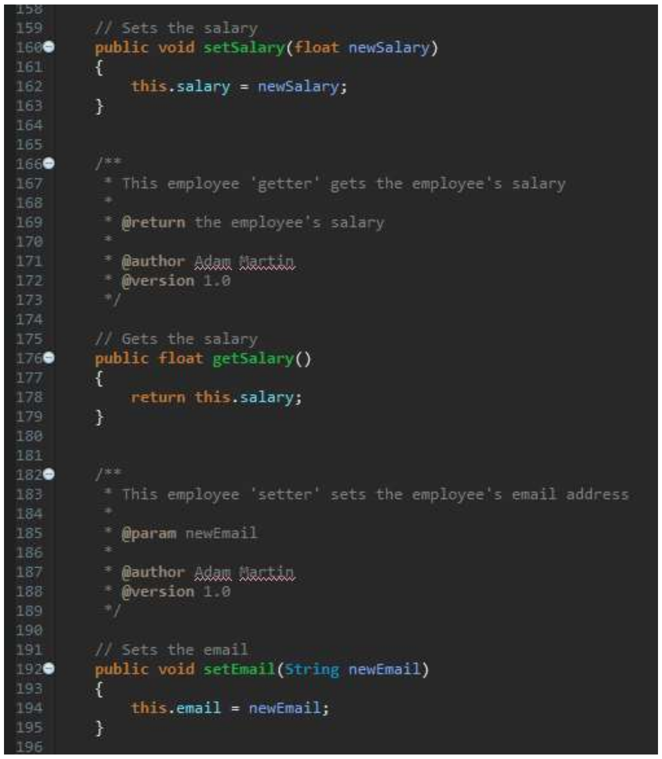
```java
        // Initialization of Employee attributes
        this.employeeNumber = newEmployeeNumber;
        this.department = newDepartment;
        this.startDate = newStartDate;
        this.salary = newSalary;
        this.email = newEmail;
    }


    // 'Setter and Getter' methods to get and return values


    /**
     * This employee 'setter' sets the employee's employee number number.
     *
     * @param newEmployeeNumber
     *
     * @author Adam Martin
     * @version 1.0
     */

    // Sets the employee number
    public void setEmployeeNumber (int newEmployeeNumber)
    {
        this.employeeNumber = newEmployeeNumber;
    }


    /**
     * This employee 'getter' gets the employee number of the employee
     *
     * @return the employee's employee number
     *
     * @author Adam Martin
     * @version 1.0
     */
```



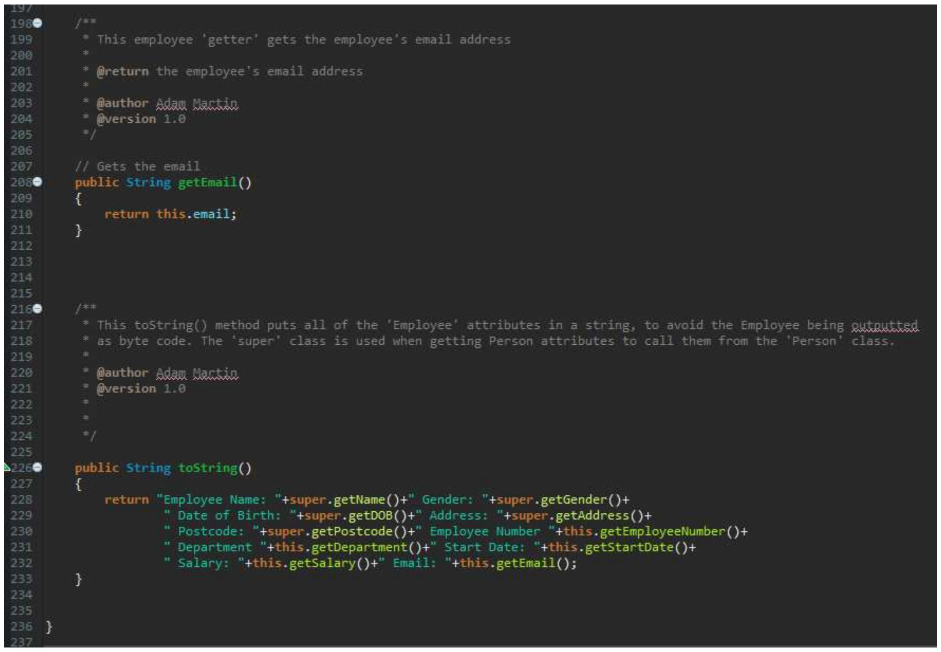
```java
    // Sets the salary
    public void setSalary(float newSalary)
    {
        this.salary = newSalary;
    }


    /**
     * This employee 'getter' gets the employee's salary
     *
     * @return the employee's salary
     *
     * @author Adam Martin
     * @version 1.0
     */

    // Gets the salary
    public float getSalary()
    {
        return this.salary;
    }


    /**
     * This employee 'setter' sets the employee's email address
     *
     * @param newEmail
     *
     * @author Adam Martin
     * @version 1.0
     */

    // Sets the email
    public void setEmail(String newEmail)
    {
        this.email = newEmail;
    }

     * @version 1.0
     */

    // Gets the department
    public String getDepartment()
    {
        return this.department;
    }
```



```java
    /**
     * This employee 'setter' sets the employee's start date
     *
     * @param newStartDate
     *
     * @author Adam Martin
     * @version 1.0
     */

    // Sets the start date
    public void setStartDate(String newStartDate)
    {
        this.startDate = newStartDate;
    }


    /**
     * This employee 'getter' gets the employee's start date
     *
     * @return the employee's start date
     *
     * @author Adam Martin
     * @version 1.0
     */

    // Gets the start date
    public String getStartDate()
    {
        return this.startDate;
    }


    /**
     * This employee 'setter' sets the employee's salary
     *
     * @param newSalary
     *
     * @author Adam Martin
     * @version 1.0
     */
```



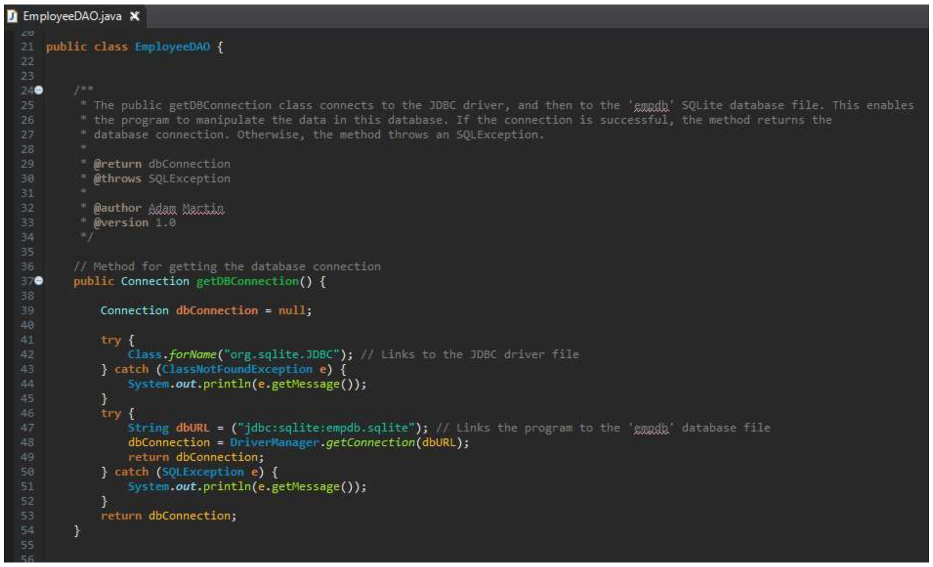
```java
    // Sets the salary
    public void setSalary(float newSalary)
    {
        this.salary = newSalary;
    }


    /**
     * This employee 'getter' gets the employee's salary
     *
     * @return the employee's salary
     *
     * @author Adam Martin
     * @version 1.0
     */

    // Gets the salary
    public float getSalary()
    {
        return this.salary;
    }


    /**
     * This employee 'setter' sets the employee's email address
     *
     * @param newEmail
     *
     * @author Adam Martin
     * @version 1.0
     */

    // Sets the email
    public void setEmail(String newEmail)
    {
        this.email = newEmail;
    }

```



```java
    /**
     * This employee 'getter' gets the employee's email address
     *
     * @return the employee's email address
     *
     * @author Adam Martin
     * @version 1.0
     */

    // Gets the email
    public String getEmail()
    {
        return this.email;
    }




    /**
     * This toString() method puts all of the 'Employee' attributes in a string, to avoid the Employee being outputted
     * as byte code. The 'super' class is used when getting Person attributes to call them from the 'Person' class.
     *
     * @author Adam Martin
     * @version 1.0
     *
     *
     */

    public String toString()
    {
        return "Employee Name: "+super.getName()+" Gender: "+super.getGender()+
                " Date of Birth: "+super.getDOB()+" Address: "+super.getAddress()+
                " Postcode: "+super.getPostcode()+" Employee Number "+this.getEmployeeNumber()+
                " Department "+this.getDepartment()+" Start Date: "+this.getStartDate()+
                " Salary: "+this.getSalary()+" Email: "+this.getEmail();
    }


}
```

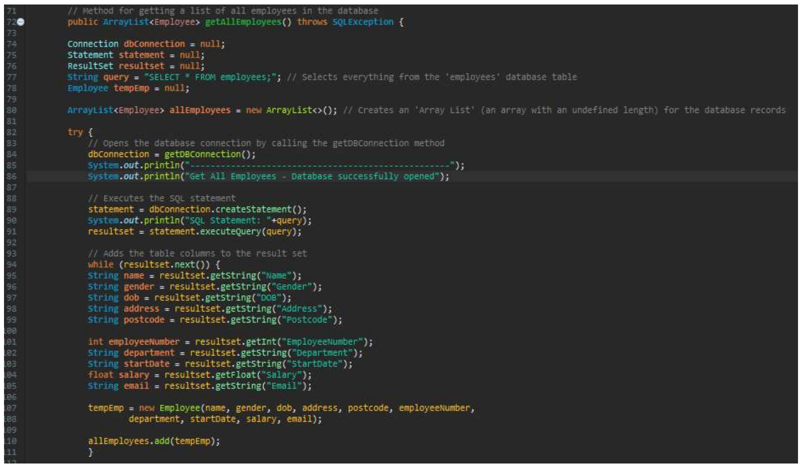
The employee base class 'extends' the person class, to inherit its attributes. The class contains setter and getter methods for employee-specific attributes (employee number, department, start date, salary email). These attributes are placed in a toString method, in order to print them to the console in a readable text format. This successful implementation of person and employee base classes allows to program to manipulate database records.

Step 3 – ‘EmployeeDAO’ Class:

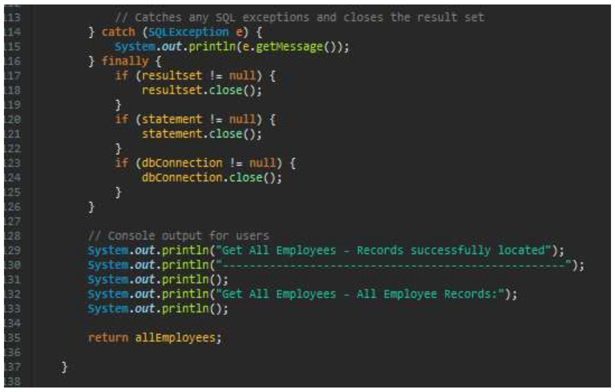


EmployeeDAO.java

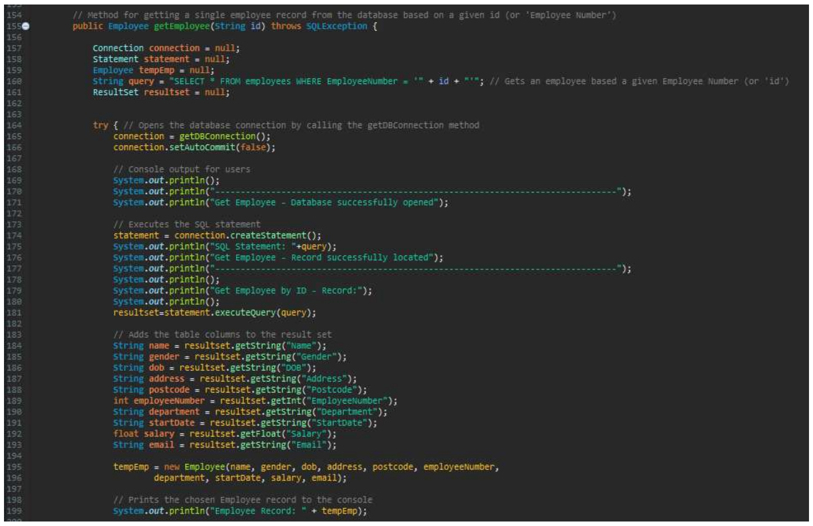
```java
public class EmployeeDAO {


    /**
     * The public getDBConnection class connects to the JDBC driver, and then to the 'empdb' SQLite database file. This enables
     * the program to manipulate the data in this database. If the connection is successful, the method returns the
     * database connection. Otherwise, the method throws an SQLException.
     *
     * @return dbConnection
     * @throws SQLException
     *
     * @author Adam Martin
     * @version 1.0
     */

    // Method for getting the database connection
    public Connection getDBConnection() {

        Connection dbConnection = null;

        try {
            Class.forName("org.sqlite.JDBC"); // Links to the JDBC driver file
        } catch (ClassNotFoundException e) {
            System.out.println(e.getMessage());
        }
        try {
            String dbURL = ("jdbc:sqlite:empdb.sqlite"); // Links the program to the 'empdb' database file
            dbConnection = DriverManager.getConnection(dbURL);
            return dbConnection;
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
        return dbConnection;
    }
```



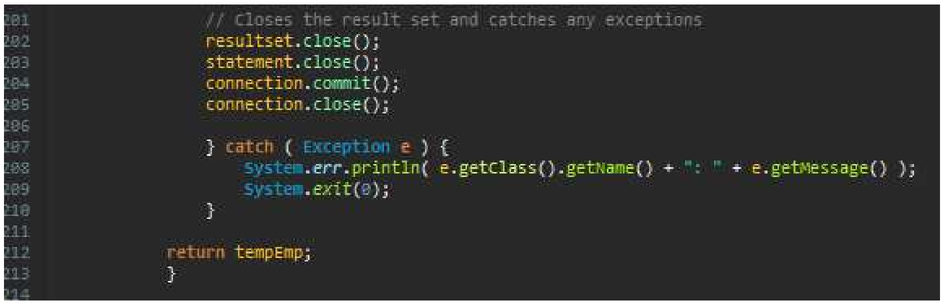
```java
    // Method for getting a list of all employees in the database
    public ArrayList<Employee> getAllEmployees() throws SQLException {

    Connection dbConnection = null;
    Statement statement = null;
    ResultSet resultset = null;
    String query = "SELECT * FROM employees;"; // Selects everything from the 'employees' database table
    Employee tempEmp = null;

    ArrayList<Employee> allEmployees = new ArrayList<>(); // Creates an 'Array List' (an array with an undefined length) for the database records

    try {
        // Opens the database connection by calling the getDBConnection method
        dbConnection = getDBConnection();
        System.out.println("-----------------------------------------------------");
        System.out.println("Get All Employees - Database successfully opened");

        // Executes the SQL statement
        statement = dbConnection.createStatement();
        System.out.println("SQL Statement: "+query);
        resultset = statement.executeQuery(query);

        // Adds the table columns to the result set
        while (resultset.next()) {
        String name = resultset.getString("Name");
        String gender = resultset.getString("Gender");
        String dob = resultset.getString("DOB");
        String address = resultset.getString("Address");
        String postcode = resultset.getString("Postcode");

        int employeeNumber = resultset.getInt("EmployeeNumber");
        String department = resultset.getString("Department");
        String startDate = resultset.getString("StartDate");
        float salary = resultset.getFloat("Salary");
        String email = resultset.getString("Email");

        tempEmp = new Employee(name, gender, dob, address, postcode, employeeNumber,
                department, startDate, salary, email);

        allEmployees.add(tempEmp);
        }
```



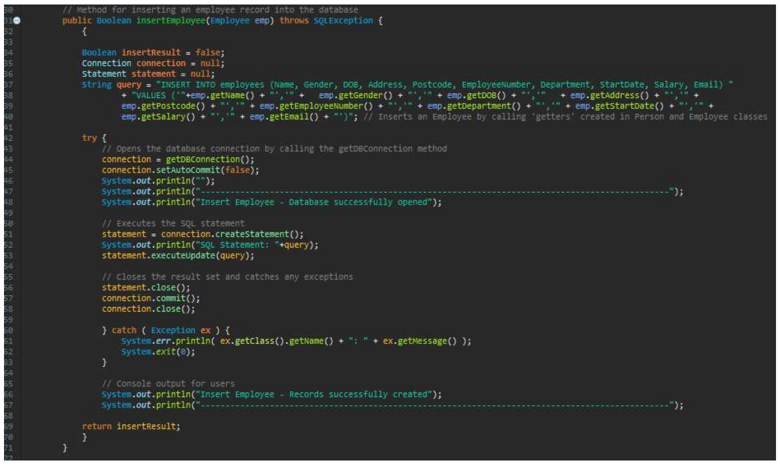
```java
            // Catches any SQL exceptions and closes the result set
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        } finally {
            if (resultset != null) {
                resultset.close();
            }
            if (statement != null) {
                statement.close();
            }
            if (dbConnection != null) {
                dbConnection.close();
            }
        }

        // Console output for users
        System.out.println("Get All Employees - Records successfully located");
        System.out.println("-----------------------------------------------------");
        System.out.println();
        System.out.println("Get All Employees - All Employee Records:");
        System.out.println();

        return allEmployees;

    }
```



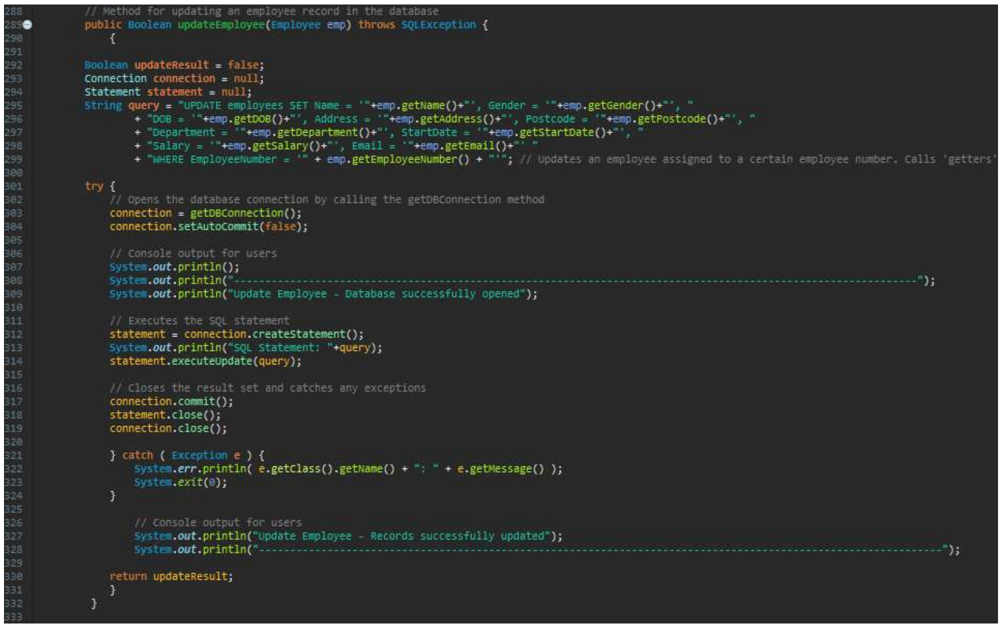
```java
    // Method for getting a single employee record from the database based on a given id (or 'Employee Number')
    public Employee getEmployee(String id) throws SQLException {

        Connection connection = null;
        Statement statement = null;
        Employee tempEmp = null;
        String query = "SELECT * FROM employees WHERE EmployeeNumber = '" + id + "'"; // Gets an employee based a given Employee Number (or 'id')
        ResultSet resultset = null;


        try { // Opens the database connection by calling the getDBConnection method
            connection = getDBConnection();
            connection.setAutoCommit(false);

            // Console output for users
            System.out.println();
            System.out.println("--------------------------------------------------------------------------------");
            System.out.println("Get Employee - Database successfully opened");

            // Executes the SQL statement
            statement = connection.createStatement();
            System.out.println("SQL Statement: "+query);
            System.out.println("Get Employee - Record successfully located");
            System.out.println("--------------------------------------------------------------------------------");
            System.out.println();
            System.out.println("Get Employee by ID - Record:");
            System.out.println();
            resultset=statement.executeQuery(query);

            // Adds the table columns to the result set
            String name = resultset.getString("Name");
            String gender = resultset.getString("Gender");
            String dob = resultset.getString("DOB");
            String address = resultset.getString("Address");
            String postcode = resultset.getString("Postcode");
            int employeeNumber = resultset.getInt("EmployeeNumber");
            String department = resultset.getString("Department");
            String startDate = resultset.getString("StartDate");
            float salary = resultset.getFloat("Salary");
            String email = resultset.getString("Email");

            tempEmp = new Employee(name, gender, dob, address, postcode, employeeNumber,
                    department, startDate, salary, email);

            // Prints the chosen Employee record to the console
            System.out.println("Employee Record: " + tempEmp);
```



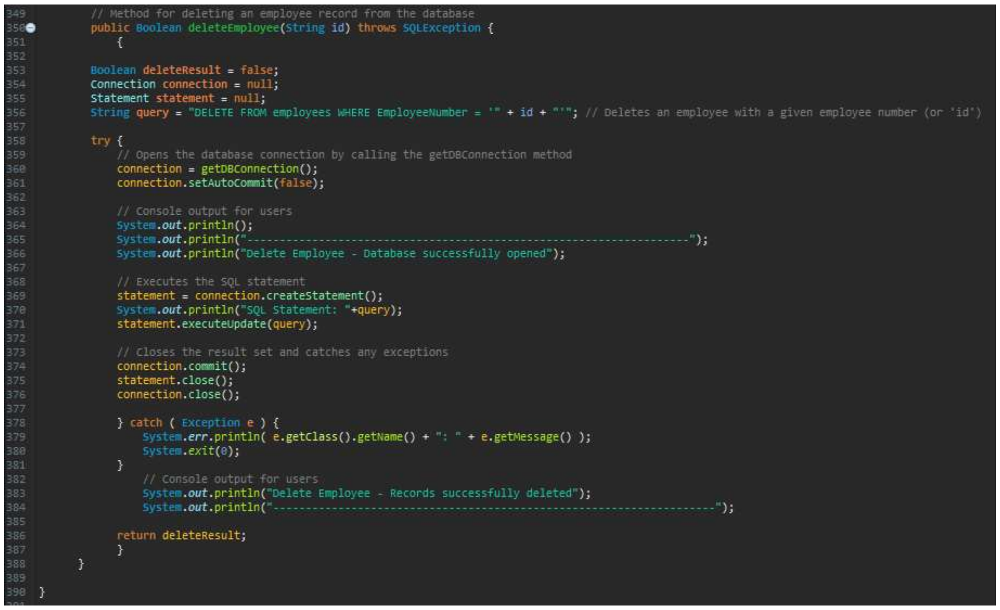
```java
            // Closes the result set and catches any exceptions
            resultset.close();
            statement.close();
            connection.commit();
            connection.close();

        } catch ( Exception e ) {
            System.err.println( e.getClass().getName() + ": " + e.getMessage() );
            System.exit(0);
        }

    return tempEmp;
    }
```



```java
    // Method for inserting an employee record into the database
    public Boolean insertEmployee(Employee emp) throws SQLException {
        {

        Boolean insertResult = false;
        Connection connection = null;
        Statement statement = null;
        String query = "INSERT INTO employees (Name, Gender, DOB, Address, Postcode, EmployeeNumber, Department, StartDate, Salary, Email) "
                + "VALUES ('"+emp.getName() + "','" +   emp.getGender() + "','" + emp.getDOB() + "','"   + emp.getAddress() + "','" +
                emp.getPostcode() + "','" + emp.getEmployeeNumber() + "','" + emp.getDepartment() + "','" + emp.getStartDate() + "','" +
                emp.getSalary() + "','" + emp.getEmail() + "')"; // Inserts an Employee by calling 'getters' created in Person and Employee classes

        try {
            // Opens the database connection by calling the getDBConnection method
            connection = getDBConnection();
            connection.setAutoCommit(false);
            System.out.println("");
            System.out.println("----------------------------------------------------------------------------------------");
            System.out.println("Insert Employee - Database successfully opened");

            // Executes the SQL statement
            statement = connection.createStatement();
            System.out.println("SQL Statement: "+query);
            statement.executeUpdate(query);

            // Closes the result set and catches any exceptions
            statement.close();
            connection.commit();
            connection.close();

            } catch ( Exception ex ) {
                System.err.println( ex.getClass().getName() + ": " + ex.getMessage() );
                System.exit(0);
            }

            // Console output for users
            System.out.println("Insert Employee - Records successfully created");
            System.out.println("----------------------------------------------------------------------------------------");

        return insertResult;
        }
    }
```



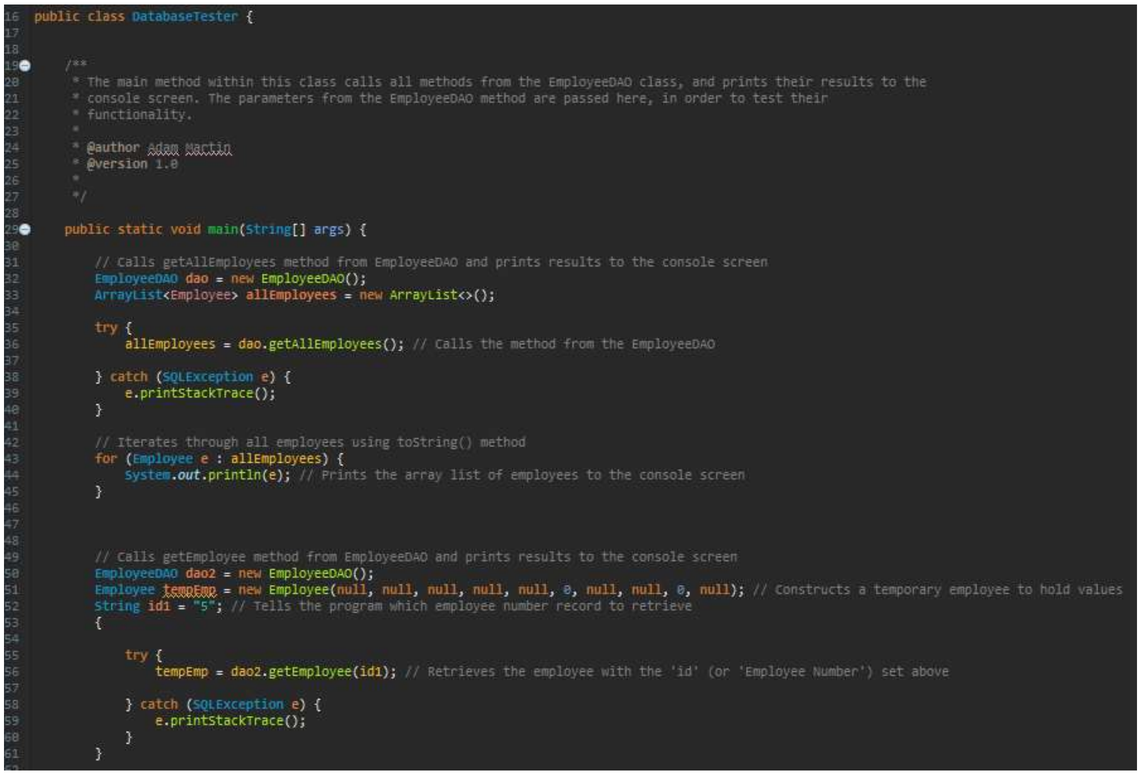
```java
    // Method for updating an employee record in the database
    public Boolean updateEmployee(Employee emp) throws SQLException {
        {

    Boolean updateResult = false;
    Connection connection = null;
    Statement statement = null;
    String query = "UPDATE employees SET Name = '"+emp.getName()+"', Gender = '"+emp.getGender()+"', "
            + "DOB = '"+emp.getDOB()+"', Address = '"+emp.getAddress()+"', Postcode = '"+emp.getPostcode()+"', "
            + "Department = '"+emp.getDepartment()+"', StartDate = '"+emp.getStartDate()+"', "
            + "Salary = '"+emp.getSalary()+"', Email = '"+emp.getEmail()+"' "
            + "WHERE EmployeeNumber = '" + emp.getEmployeeNumber() + "'"; // Updates an employee assigned to a certain employee number. Calls 'getters'

    try {
        // Opens the database connection by calling the getDBConnection method
        connection = getDBConnection();
        connection.setAutoCommit(false);

        // Console output for users
        System.out.println();
        System.out.println("--------------------------------------------------------------------------------------------------------------------");
        System.out.println("Update Employee - Database successfully opened");

        // Executes the SQL statement
        statement = connection.createStatement();
        System.out.println("SQL Statement: "+query);
        statement.executeUpdate(query);

        // Closes the result set and catches any exceptions
        connection.commit();
        statement.close();
        connection.close();

        } catch ( Exception e ) {
            System.err.println( e.getClass().getName() + ": " + e.getMessage() );
            System.exit(0);
        }

            // Console output for users
            System.out.println("Update Employee - Records successfully updated");
            System.out.println("--------------------------------------------------------------------------------------------------------------------");

        return updateResult;
        }
    }
```



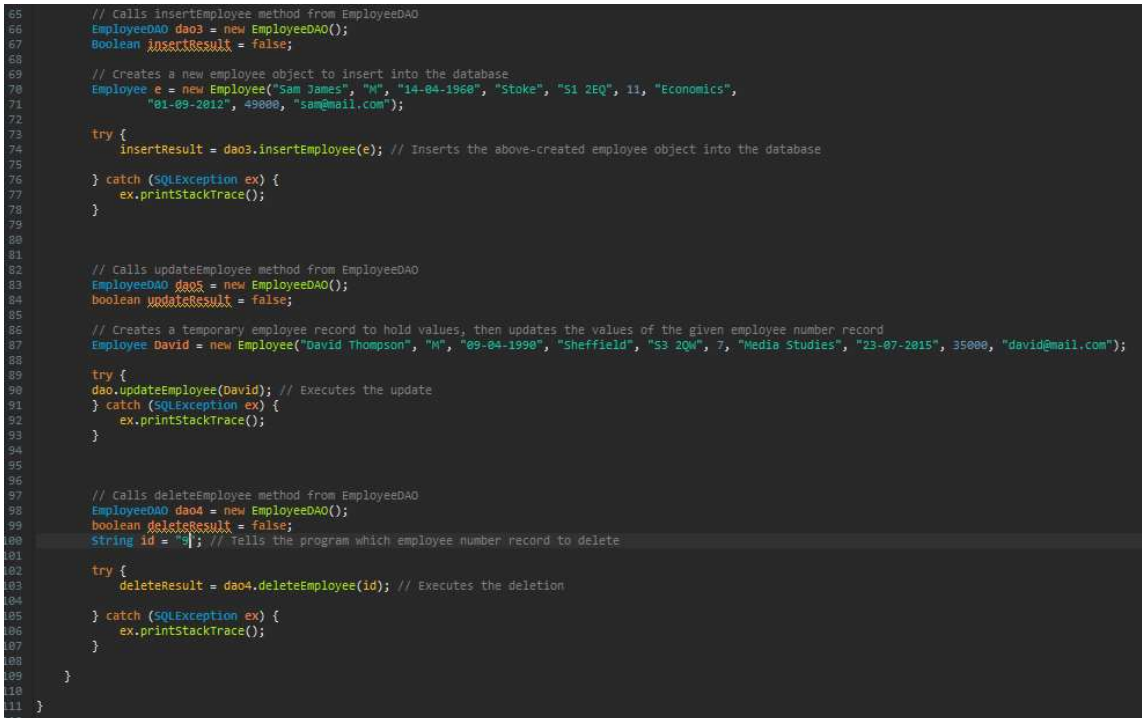
```java
    // Method for deleting an employee record from the database
    public Boolean deleteEmployee(String id) throws SQLException {
        {

    Boolean deleteResult = false;
    Connection connection = null;
    Statement statement = null;
    String query = "DELETE FROM employees WHERE EmployeeNumber = '" + id + "'"; // Deletes an employee with a given employee number (or 'id')

    try {
        // Opens the database connection by calling the getDBConnection method
        connection = getDBConnection();
        connection.setAutoCommit(false);

        // Console output for users
        System.out.println();
        System.out.println("-----------------------------------------------------------------------------");
        System.out.println("Delete Employee - Database successfully opened");

        // Executes the SQL statement
        statement = connection.createStatement();
        System.out.println("SQL Statement: "+query);
        statement.executeUpdate(query);

        // Closes the result set and catches any exceptions
        connection.commit();
        statement.close();
        connection.close();

        } catch ( Exception e ) {
            System.err.println( e.getClass().getName() + ": " + e.getMessage() );
            System.exit(0);
        }
            // Console output for users
            System.out.println("Delete Employee - Records successfully deleted");
            System.out.println("-----------------------------------------------------------------------------");

        return deleteResult;
        }
    }

}
```

The EmployeeDAO class contains a method for connecting to the database, and the CRUD methods for getting all employees, getting a single employee, inserting an employee, updating an employee and deleting an employee. The CRUD operations are actioned using a range of SQL statements. This class is crucial to the application, as its CRUD methods are called throughout the program.

Step 4 – 'DatabaseTester' Class:



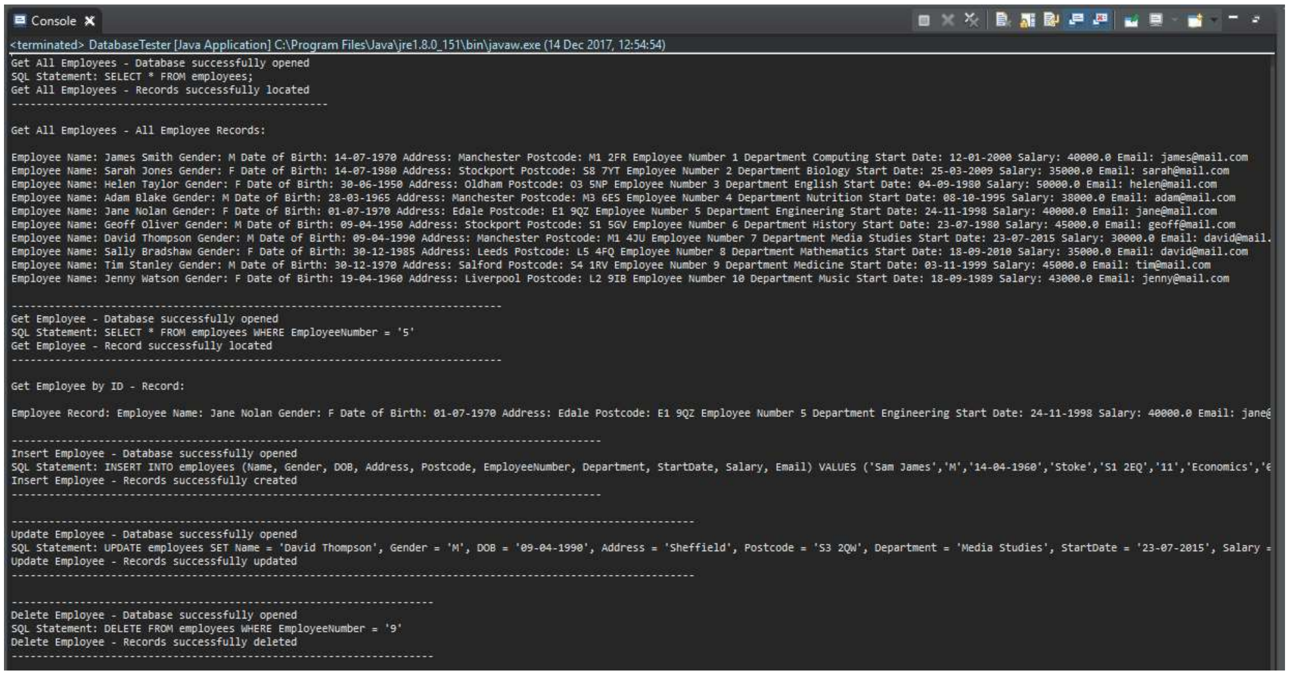
```java
public class DatabaseTester {


    /**
     * The main method within this class calls all methods from the EmployeeDAO class, and prints their results to the
     * console screen. The parameters from the EmployeeDAO method are passed here, in order to test their
     * functionality.
     *
     * @author Adam Martin
     * @version 1.0
     *
     */

    public static void main(String[] args) {

        // Calls getAllEmployees method from EmployeeDAO and prints results to the console screen
        EmployeeDAO dao = new EmployeeDAO();
        ArrayList<Employee> allEmployees = new ArrayList<>();

        try {
            allEmployees = dao.getAllEmployees(); // Calls the method from the EmployeeDAO

        } catch (SQLException e) {
            e.printStackTrace();
        }

        // Iterates through all employees using toString() method
        for (Employee e : allEmployees) {
            System.out.println(e); // Prints the array list of employees to the console screen
        }



        // Calls getEmployee method from EmployeeDAO and prints results to the console screen
        EmployeeDAO dao2 = new EmployeeDAO();
        Employee tempEmp = new Employee(null, null, null, null, null, 0, null, null, 0, null); // Constructs a temporary employee to hold values
        String id1 = "5"; // Tells the program which employee number record to retrieve
        {

            try {
                tempEmp = dao2.getEmployee(id1); // Retrieves the employee with the 'id' (or 'Employee Number') set above

            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
```



```java
        // Calls insertEmployee method from EmployeeDAO
        EmployeeDAO dao3 = new EmployeeDAO();
        Boolean insertResult = false;

        // Creates a new employee object to insert into the database
        Employee e = new Employee("Sam James", "M", "14-04-1960", "Stoke", "S1 2EQ", 11, "Economics",
                "01-09-2012", 49000, "sam@mail.com");

        try {
            insertResult = dao3.insertEmployee(e); // Inserts the above-created employee object into the database

        } catch (SQLException ex) {
            ex.printStackTrace();
        }



        // Calls updateEmployee method from EmployeeDAO
        EmployeeDAO dao5 = new EmployeeDAO();
        boolean updateResult = false;

        // Creates a temporary employee record to hold values, then updates the values of the given employee number record
        Employee David = new Employee("David Thompson", "M", "09-04-1990", "Sheffield", "S3 2QW", 7, "Media Studies", "23-07-2015", 35000, "david@mail.com");

        try {
        dao.updateEmployee(David); // Executes the update
        } catch (SQLException ex) {
            ex.printStackTrace();
        }



        // Calls deleteEmployee method from EmployeeDAO
        EmployeeDAO dao4 = new EmployeeDAO();
        boolean deleteResult = false;
        String id = "5"; // Tells the program which employee number record to delete

        try {
            deleteResult = dao4.deleteEmployee(id); // Executes the deletion

        } catch (SQLException ex) {
            ex.printStackTrace();
        }

    }

}
```

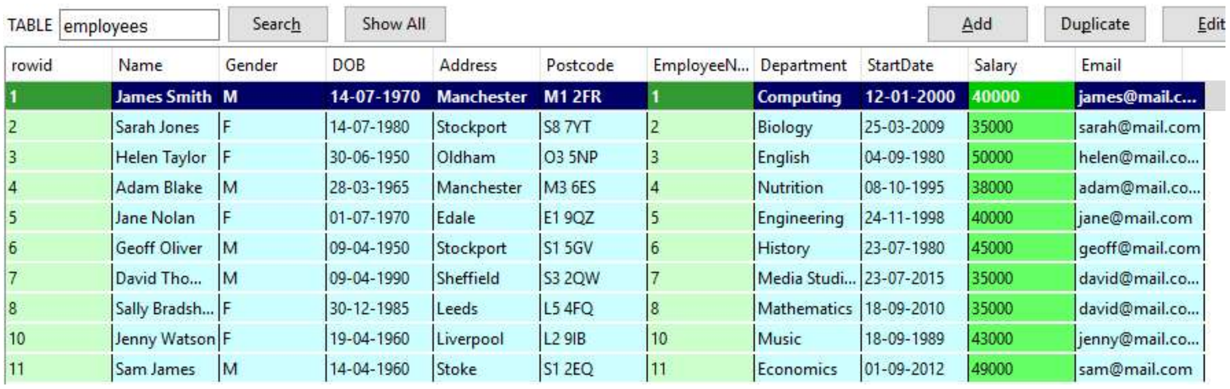
The DatabaseTester class calls the CRUD methods from the EmployeeDAO class, by passing in test data. The results of the test are printed to the console screen.

DatabaseTester – Console Output:



```
<terminated> DatabaseTester [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (14 Dec 2017, 12:54:54)
Get All Employees - Database successfully opened
SQL Statement: SELECT * FROM employees;
Get All Employees - Records successfully located
---------------------------------------------------

Get All Employees - All Employee Records:

Employee Name: James Smith Gender: M Date of Birth: 14-07-1970 Address: Manchester Postcode: M1 2FR Employee Number 1 Department Computing Start Date: 12-01-2000 Salary: 40000.0 Email: james@mail.com
Employee Name: Sarah Jones Gender: F Date of Birth: 14-07-1980 Address: Stockport Postcode: S8 7YT Employee Number 2 Department Biology Start Date: 25-03-2009 Salary: 35000.0 Email: sarah@mail.com
Employee Name: Helen Taylor Gender: F Date of Birth: 30-06-1950 Address: Oldham Postcode: O3 5NP Employee Number 3 Department English Start Date: 04-09-1980 Salary: 50000.0 Email: helen@mail.com
Employee Name: Adam Blake Gender: M Date of Birth: 28-03-1965 Address: Manchester Postcode: M3 6ES Employee Number 4 Department Nutrition Start Date: 08-10-1995 Salary: 38000.0 Email: adam@mail.com
Employee Name: Jane Nolan Gender: F Date of Birth: 01-07-1970 Address: Edale Postcode: E1 9QZ Employee Number 5 Department Engineering Start Date: 24-11-1998 Salary: 40000.0 Email: jane@mail.com
Employee Name: Geoff Oliver Gender: M Date of Birth: 09-04-1950 Address: Stockport Postcode: S1 5GV Employee Number 6 Department History Start Date: 23-07-1980 Salary: 45000.0 Email: geoff@mail.com
Employee Name: David Thompson Gender: M Date of Birth: 09-04-1990 Address: Manchester Postcode: M1 4JU Employee Number 7 Department Media Studies Start Date: 23-07-2015 Salary: 30000.0 Email: david@mail.
Employee Name: Sally Bradshaw Gender: F Date of Birth: 30-12-1985 Address: Leeds Postcode: L5 4FQ Employee Number 8 Department Mathematics Start Date: 18-09-2010 Salary: 35000.0 Email: david@mail.com
Employee Name: Tim Stanley Gender: M Date of Birth: 30-12-1970 Address: Salford Postcode: S4 1RV Employee Number 9 Department Medicine Start Date: 03-11-1999 Salary: 45000.0 Email: tim@mail.com
Employee Name: Jenny Watson Gender: F Date of Birth: 19-04-1960 Address: Liverpool Postcode: L2 9IB Employee Number 10 Department Music Start Date: 18-09-1989 Salary: 43000.0 Email: jenny@mail.com

----------------------------------------------------------------------------
Get Employee - Database successfully opened
SQL Statement: SELECT * FROM employees WHERE EmployeeNumber = '5'
Get Employee - Record successfully located
----------------------------------------------------------------------------

Get Employee by ID - Record:

Employee Record: Employee Name: Jane Nolan Gender: F Date of Birth: 01-07-1970 Address: Edale Postcode: E1 9QZ Employee Number 5 Department Engineering Start Date: 24-11-1998 Salary: 40000.0 Email: jane@

---------------------------------------------------------------------------------------------
Insert Employee - Database successfully opened
SQL Statement: INSERT INTO employees (Name, Gender, DOB, Address, Postcode, EmployeeNumber, Department, StartDate, Salary, Email) VALUES ('Sam James','M','14-04-1960','Stoke','S1 2EQ','11','Economics','
Insert Employee - Records successfully created
---------------------------------------------------------------------------------------------


------------------------------------------------------------------------------------------------------------
Update Employee - Database successfully opened
SQL Statement: UPDATE employees SET Name = 'David Thompson', Gender = 'M', DOB = '09-04-1990', Address = 'Sheffield', Postcode = 'S3 2QW', Department = 'Media Studies', StartDate = '23-07-2015', Salary =
Update Employee - Records successfully updated
------------------------------------------------------------------------------------------------------------


-------------------------------------------------------------------
Delete Employee - Database successfully opened
SQL Statement: DELETE FROM employees WHERE EmployeeNumber = '9'
Delete Employee - Records successfully deleted
-------------------------------------------------------------------
```

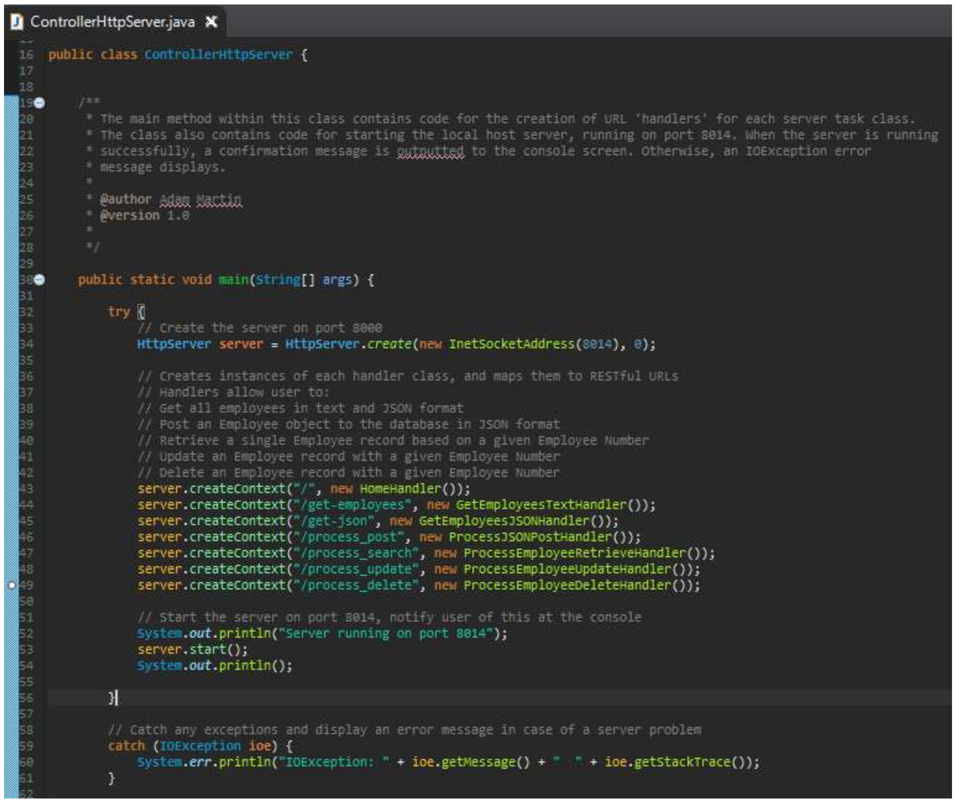
DatabaseTester – Updated 'Employees' Database Table:



rowid	Name	Gender	DOB	Address	Postcode	EmployeeN...	Department	StartDate	Salary	Email
1	James Smith	M	14-07-1970	Manchester	M1 2FR	1	Computing	12-01-2000	40000	james@mail.c...
2	Sarah Jones	F	14-07-1980	Stockport	S8 7YT	2	Biology	25-03-2009	35000	sarah@mail.com
3	Helen Taylor	F	30-06-1950	Oldham	O3 5NP	3	English	04-09-1980	50000	helen@mail.co...
4	Adam Blake	M	28-03-1965	Manchester	M3 6ES	4	Nutrition	08-10-1995	38000	adam@mail.co...
5	Jane Nolan	F	01-07-1970	Edale	E1 9QZ	5	Engineering	24-11-1998	40000	jane@mail.com
6	Geoff Oliver	M	09-04-1950	Stockport	S1 5GV	6	History	23-07-1980	45000	geoff@mail.com
7	David Tho...	M	09-04-1990	Sheffield	S3 2QW	7	Media Studi...	23-07-2015	35000	david@mail.co...
8	Sally Bradsh...	F	30-12-1985	Leeds	L5 4FQ	8	Mathematics	18-09-2010	35000	david@mail.co...
10	Jenny Watson	F	19-04-1960	Liverpool	L2 9IB	10	Music	18-09-1989	43000	jenny@mail.co...
11	Sam James	M	14-04-1960	Stoke	S1 2EQ	11	Economics	01-09-2012	49000	sam@mail.com

This screenshot depicts the 'employees' SQLite database table, after the DatabaseTester class was run in Eclipse. As you can see, all of the operations worked successfully. All employee records were printed to the console, and a single employee, with an employee number of '5' was also printed to the console. As was stated in the test data, a new employee, 'Sam James', with an employee number of '11' was inserted into the database. Also, the employee with an employee number of '7', David Thompson, had his address, postcode and salary information updated (all other attributes can also be updated). As you can see in the table, the employee with an employee number of '9' (Tim Stanley) is no longer present, as his record was successfully deleted from the database.

Step 5 – 'ControllerHttpServer' Class:

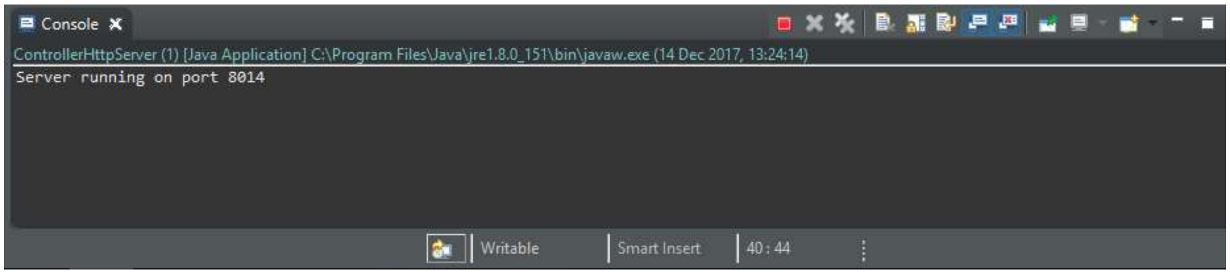


ControllerHttpServer.java

```java
public class ControllerHttpServer {


    /**
     * The main method within this class contains code for the creation of URL 'handlers' for each server task class.
     * The class also contains code for starting the local host server, running on port 8014. When the server is running
     * successfully, a confirmation message is outputted to the console screen. Otherwise, an IOException error
     * message displays.
     *
     * @author Adam Martin
     * @version 1.0
     *
     */

    public static void main(String[] args) {

        try {
            // Create the server on port 8000
            HttpServer server = HttpServer.create(new InetSocketAddress(8014), 0);

            // Creates instances of each handler class, and maps them to RESTful URLs
            // Handlers allow user to:
            // Get all employees in text and JSON format
            // Post an Employee object to the database in JSON format
            // Retrieve a single Employee record based on a given Employee Number
            // Update an Employee record with a given Employee Number
            // Delete an Employee record with a given Employee Number
            server.createContext("/", new HomeHandler());
            server.createContext("/get-employees", new GetEmployeesTextHandler());
            server.createContext("/get-json", new GetEmployeesJSONHandler());
            server.createContext("/process_post", new ProcessJSONPostHandler());
            server.createContext("/process_search", new ProcessEmployeeRetrieveHandler());
            server.createContext("/process_update", new ProcessEmployeeUpdateHandler());
            server.createContext("/process_delete", new ProcessEmployeeDeleteHandler());

            // Start the server on port 8014, notify user of this at the console
            System.out.println("Server running on port 8014");
            server.start();
            System.out.println();

        }

        // Catch any exceptions and display an error message in case of a server problem
        catch (IOException ioe) {
            System.err.println("IOException: " + ioe.getMessage() + "  " + ioe.getStackTrace());
        }
```

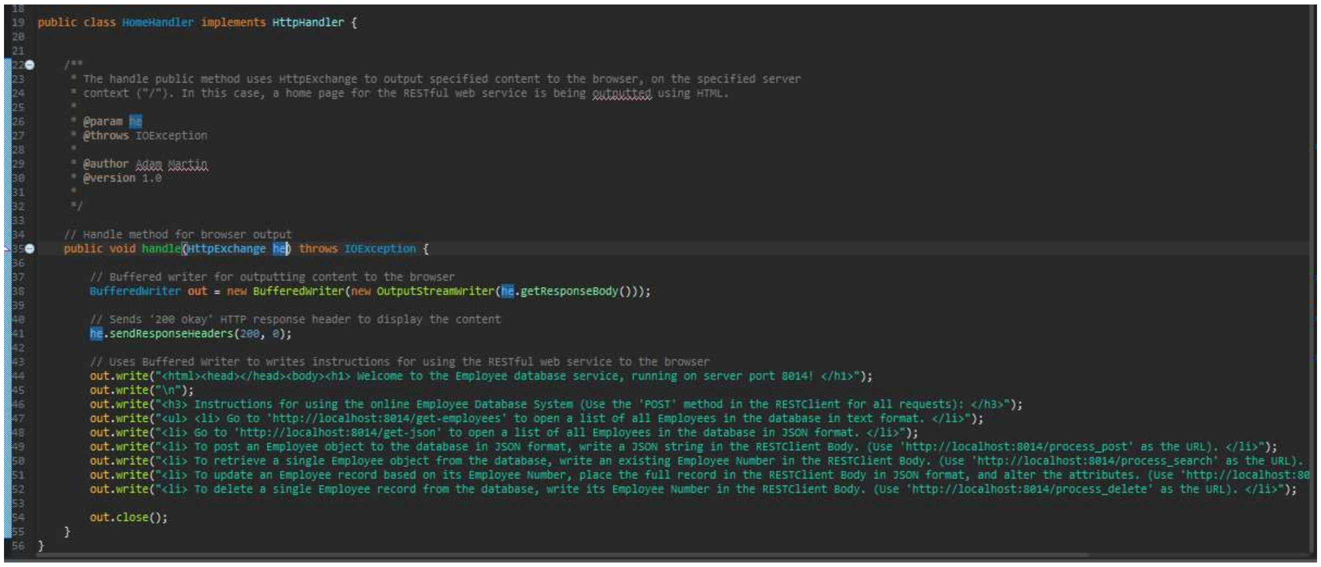
The ControllerHttpServer class was the first step of the 'server-side programming' element of the assignment. The first purpose of this class was to create a local host Http web server, running on port 8014. The server runs from this class, and displays console confirmation of this to the user. The second purpose of this class was to create server contexts for each function of the RESTful web service, by mapping the URLs to instances of each 'handler' class (handler classes are described below).

Console Output – ControllerHttpServer:



```
ControllerHttpServer (1) [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (14 Dec 2017, 13:24:14)
Server running on port 8014
```

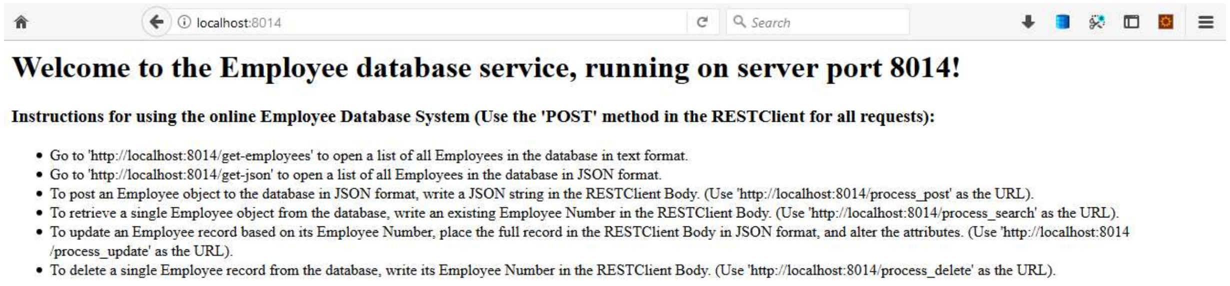
Step 6 – 'Home Handler' Class:



```java
public class HomeHandler implements HttpHandler {


    /**
     * The handle public method uses HttpExchange to output specified content to the browser, on the specified server
     * context ("/"). In this case, a home page for the RESTful web service is being outputted using HTML.
     *
     * @param he
     * @throws IOException
     *
     * @author Adam Martin
     * @version 1.0
     *
     */

    // Handle method for browser output
    public void handle(HttpExchange he) throws IOException {

        // Buffered writer for outputting content to the browser
        BufferedWriter out = new BufferedWriter(new OutputStreamWriter(he.getResponseBody()));

        // Sends '200 okay' HTTP response header to display the content
        he.sendResponseHeaders(200, 0);

        // Uses Buffered Writer to writes instructions for using the RESTful web service to the browser
        out.write("<html><head></head><body><h1> Welcome to the Employee database service, running on server port 8014! </h1>");
        out.write("\n");
        out.write("<h3> Instructions for using the online Employee Database System (Use the 'POST' method in the RESTClient for all requests): </h3>");
        out.write("<ul> <li> Go to 'http://localhost:8014/get-employees' to open a list of all Employees in the database in text format. </li>");
        out.write("<li> Go to 'http://localhost:8014/get-json' to open a list of all Employees in the database in JSON format. </li>");
        out.write("<li> To post an Employee object to the database in JSON format, write a JSON string in the RESTClient Body. (Use 'http://localhost:8014/process_post' as the URL). </li>");
        out.write("<li> To retrieve a single Employee object from the database, write an existing Employee Number in the RESTClient Body. (Use 'http://localhost:8014/process_search' as the URL).
        out.write("<li> To update an Employee record based on its Employee Number, place the full record in the RESTClient Body in JSON format, and alter the attributes. (Use 'http://localhost:80
        out.write("<li> To delete a single Employee record from the database, write its Employee Number in the RESTClient Body. (Use 'http://localhost:8014/process_delete' as the URL). </li>");

        out.close();
    }
}
```

The HomeHandler class includes code for outputting a system homepage/ instruction page to the user, with instructions displayed using embedded HTML code. The ControllerHttpServer class allows this content to display by creating an instance of the HomeHandler class, and placing it at the "/" server context.

Browser Output – HomeHandler:

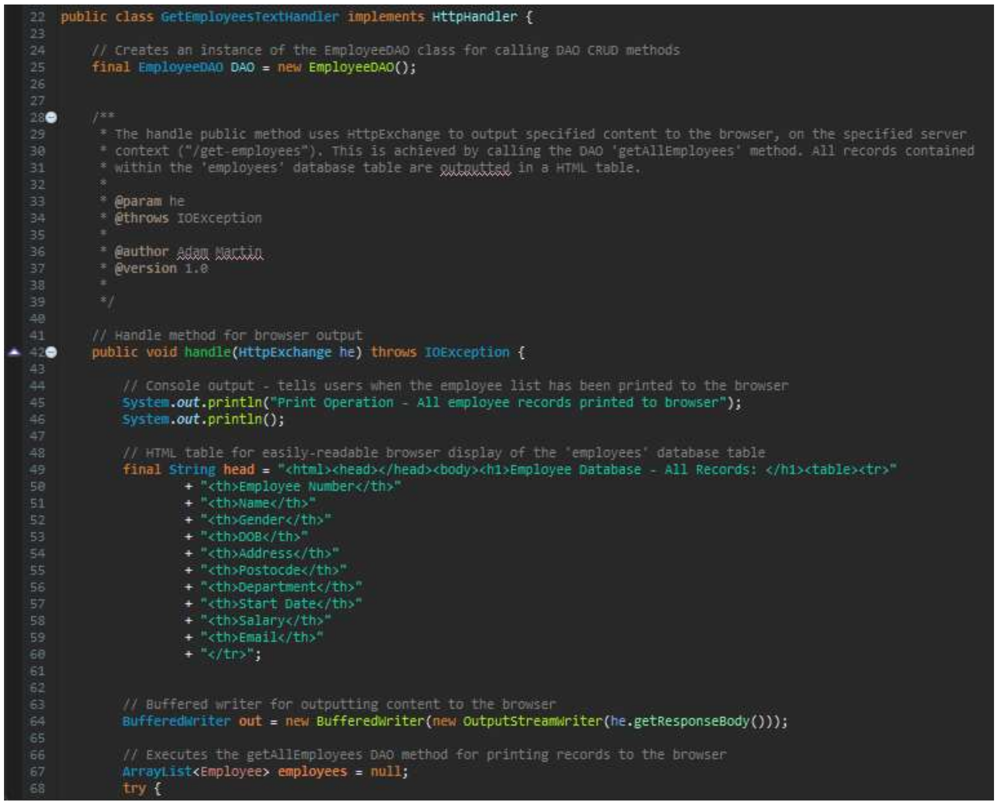


Welcome to the Employee database service, running on server port 8014!

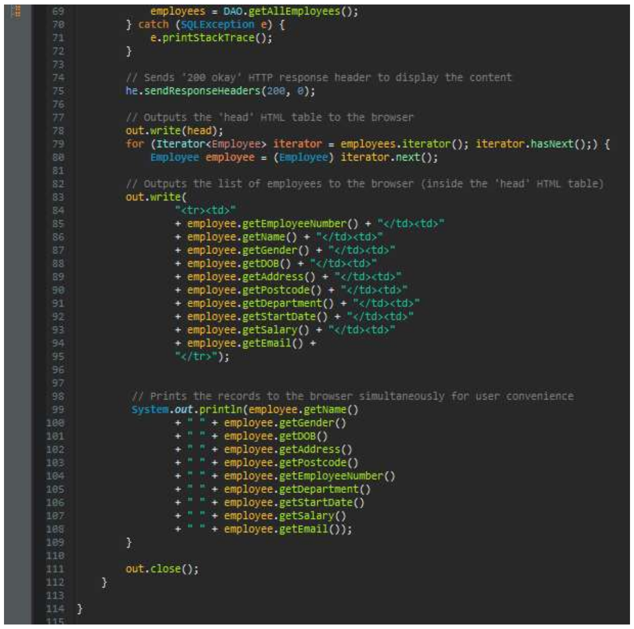
Instructions for using the online Employee Database System (Use the 'POST' method in the RESTClient for all requests):

- Go to 'http://localhost:8014/get-employees' to open a list of all Employees in the database in text format.
- Go to 'http://localhost:8014/get-json' to open a list of all Employees in the database in JSON format.
- To post an Employee object to the database in JSON format, write a JSON string in the RESTClient Body. (Use 'http://localhost:8014/process_post' as the URL).
- To retrieve a single Employee object from the database, write an existing Employee Number in the RESTClient Body. (Use 'http://localhost:8014/process_search' as the URL).
- To update an Employee record based on its Employee Number, place the full record in the RESTClient Body in JSON format, and alter the attributes. (Use 'http://localhost:8014/process_update' as the URL).
- To delete a single Employee record from the database, write its Employee Number in the RESTClient Body. (Use 'http://localhost:8014/process_delete' as the URL).

Step 7 – 'GetEmployeesTextHandler' Class:



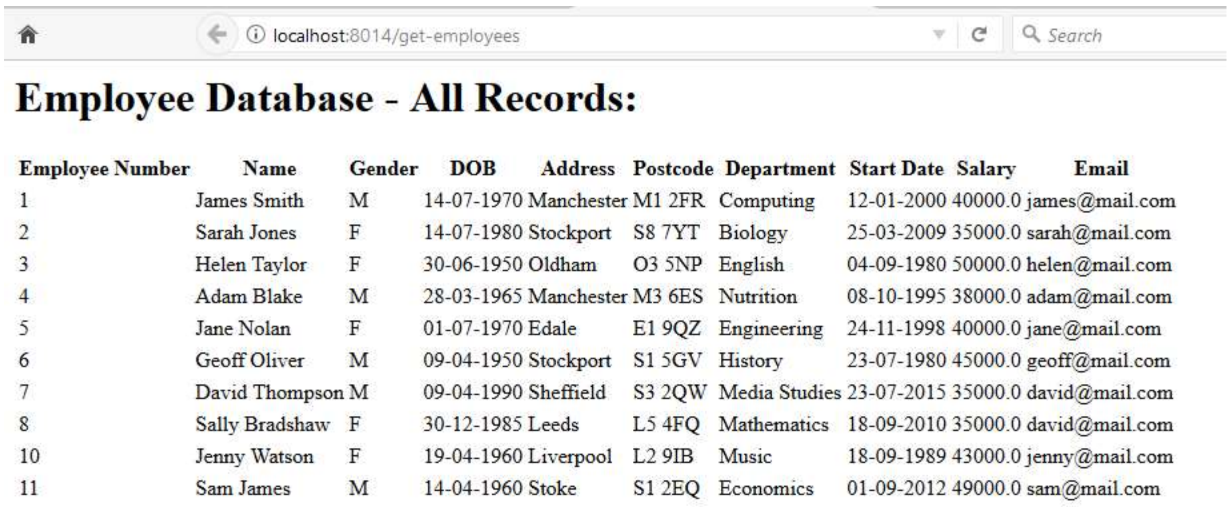
```java
public class GetEmployeesTextHandler implements HttpHandler {

    // Creates an instance of the EmployeeDAO class for calling DAO CRUD methods
    final EmployeeDAO DAO = new EmployeeDAO();


    /**
     * The handle public method uses HttpExchange to output specified content to the browser, on the specified server
     * context ("/get-employees"). This is achieved by calling the DAO 'getAllEmployees' method. All records contained
     * within the 'employees' database table are outputted in a HTML table.
     *
     * @param he
     * @throws IOException
     *
     * @author Adam Martin
     * @version 1.0
     *
     */

    // Handle method for browser output
    public void handle(HttpExchange he) throws IOException {

        // Console output - tells users when the employee list has been printed to the browser
        System.out.println("Print Operation - All employee records printed to browser");
        System.out.println();

        // HTML table for easily-readable browser display of the 'employees' database table
        final String head = "<html><head></head><body><h1>Employee Database - All Records: </h1><table><tr>"
                + "<th>Employee Number</th>"
                + "<th>Name</th>"
                + "<th>Gender</th>"
                + "<th>DOB</th>"
                + "<th>Address</th>"
                + "<th>Postocde</th>"
                + "<th>Department</th>"
                + "<th>Start Date</th>"
                + "<th>Salary</th>"
                + "<th>Email</th>"
                + "</tr>";


        // Buffered writer for outputting content to the browser
        BufferedWriter out = new BufferedWriter(new OutputStreamWriter(he.getResponseBody()));

        // Executes the getAllEmployees DAO method for printing records to the browser
        ArrayList<Employee> employees = null;
        try {
```



```java
            employees = DAO.getAllEmployees();
        } catch (SQLException e) {
            e.printStackTrace();
        }

        // Sends '200 okay' HTTP response header to display the content
        he.sendResponseHeaders(200, 0);

        // Outputs the 'head' HTML table to the browser
        out.write(head);
        for (Iterator<Employee> iterator = employees.iterator(); iterator.hasNext();) {
            Employee employee = (Employee) iterator.next();

        // Outputs the list of employees to the browser (inside the 'head' HTML table)
        out.write(
                "<tr><td>"
                + employee.getEmployeeNumber() + "</td><td>"
                + employee.getName() + "</td><td>"
                + employee.getGender() + "</td><td>"
                + employee.getDOB() + "</td><td>"
                + employee.getAddress() + "</td><td>"
                + employee.getPostcode() + "</td><td>"
                + employee.getDepartment() + "</td><td>"
                + employee.getStartDate() + "</td><td>"
                + employee.getSalary() + "</td><td>"
                + employee.getEmail() +
                "</tr>");


         // Prints the records to the browser simultaneously for user convenience
         System.out.println(employee.getName()
                + " " + employee.getGender()
                + " " + employee.getDOB()
                + " " + employee.getAddress()
                + " " + employee.getPostcode()
                + " " + employee.getEmployeeNumber()
                + " " + employee.getDepartment()
                + " " + employee.getStartDate()
                + " " + employee.getSalary()
                + " " + employee.getEmail());
        }

        out.close();
    }

}
```

The GetEmployeesTextHandler class includes code for outputting HTML table of all employees to the user. This code involved calling the GetAllEmployees method from the EmployeeDAO class. The ControllerHttpServer class allows this content to display by creating an instance of the GetEmployeesTextHandler class, and placing it at the "/get-employees" server context.

Browser Output – GetEmployeesTextHandler:

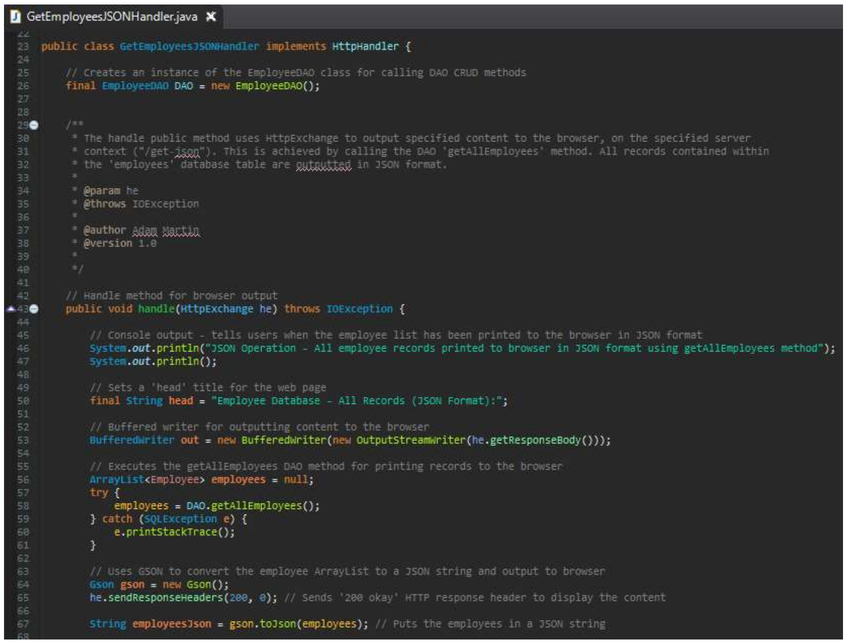

localhost:8014/get-employees

Employee Database - All Records:

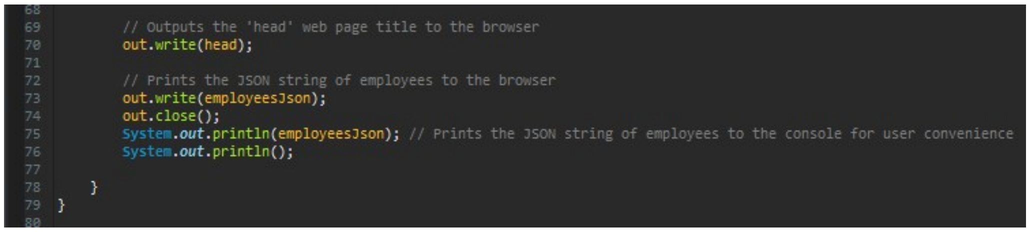
Employee Number	Name	Gender	DOB	Address	Postcode	Department	Start Date	Salary	Email
1	James Smith	M	14-07-1970	Manchester	M1 2FR	Computing	12-01-2000	40000.0	james@mail.com
2	Sarah Jones	F	14-07-1980	Stockport	S8 7YT	Biology	25-03-2009	35000.0	sarah@mail.com
3	Helen Taylor	F	30-06-1950	Oldham	O3 5NP	English	04-09-1980	50000.0	helen@mail.com
4	Adam Blake	M	28-03-1965	Manchester	M3 6ES	Nutrition	08-10-1995	38000.0	adam@mail.com
5	Jane Nolan	F	01-07-1970	Edale	E1 9QZ	Engineering	24-11-1998	40000.0	jane@mail.com
6	Geoff Oliver	M	09-04-1950	Stockport	S1 5GV	History	23-07-1980	45000.0	geoff@mail.com
7	David Thompson	M	09-04-1990	Sheffield	S3 2QW	Media Studies	23-07-2015	35000.0	david@mail.com
8	Sally Bradshaw	F	30-12-1985	Leeds	L5 4FQ	Mathematics	18-09-2010	35000.0	david@mail.com
10	Jenny Watson	F	19-04-1960	Liverpool	L2 9IB	Music	18-09-1989	43000.0	jenny@mail.com
11	Sam James	M	14-04-1960	Stoke	S1 2EQ	Economics	01-09-2012	49000.0	sam@mail.com

The above screenshot depicts browser output at the "/get-employees" server context. As you can see, the records are updated to reflect the results of the CRUD tests conducted using the DatabaseTester class.

Step 8 – 'GetEmployeesJSONHandler' Class:



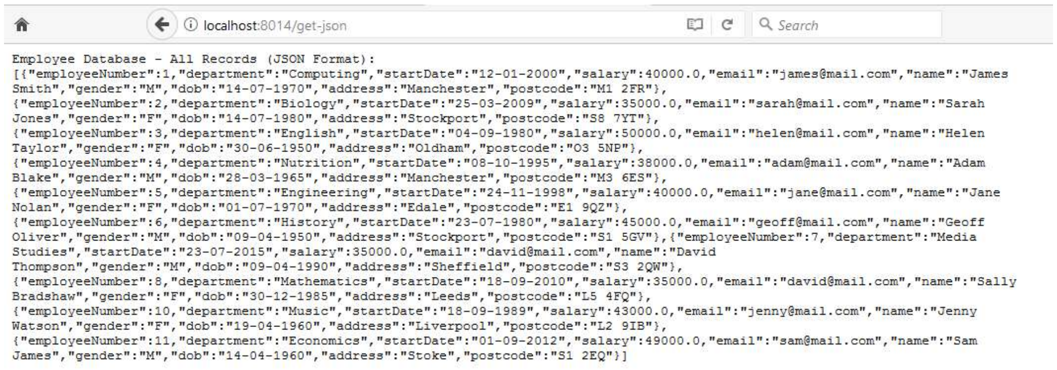
```java
public class GetEmployeesJSONHandler implements HttpHandler {

    // Creates an instance of the EmployeeDAO class for calling DAO CRUD methods
    final EmployeeDAO DAO = new EmployeeDAO();


    /**
     * The handle public method uses HttpExchange to output specified content to the browser, on the specified server
     * context ("/get-json"). This is achieved by calling the DAO 'getAllEmployees' method. All records contained within
     * the 'employees' database table are outputted in JSON format.
     *
     * @param he
     * @throws IOException
     *
     * @author Adam Martin
     * @version 1.0
     *
     */

    // Handle method for browser output
    public void handle(HttpExchange he) throws IOException {

        // Console output - tells users when the employee list has been printed to the browser in JSON format
        System.out.println("JSON Operation - All employee records printed to browser in JSON format using getAllEmployees method");
        System.out.println();

        // Sets a 'head' title for the web page
        final String head = "Employee Database - All Records (JSON Format):";

        // Buffered writer for outputting content to the browser
        BufferedWriter out = new BufferedWriter(new OutputStreamWriter(he.getResponseBody()));

        // Executes the getAllEmployees DAO method for printing records to the browser
        ArrayList<Employee> employees = null;
        try {
            employees = DAO.getAllEmployees();
        } catch (SQLException e) {
            e.printStackTrace();
        }

        // Uses GSON to convert the employee ArrayList to a JSON string and output to browser
        Gson gson = new Gson();
        he.sendResponseHeaders(200, 0); // Sends '200 okay' HTTP response header to display the content

        String employeesJson = gson.toJson(employees); // Puts the employees in a JSON string
```



```java

        // Outputs the 'head' web page title to the browser
        out.write(head);

        // Prints the JSON string of employees to the browser
        out.write(employeesJson);
        out.close();
        System.out.println(employeesJson); // Prints the JSON string of employees to the console for user convenience
        System.out.println();

    }
}
```

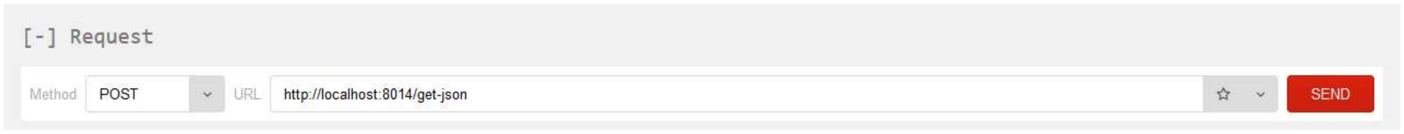
The GetEmployeesJSONHandler class includes code for outputting all employee records to the user in JSON (JavaScript Object Notation) format. This code involved calling the GetAllEmployees method from the EmployeeDAO class, converting a String of these employees 'toJson', and outputting it to the user. This functionality can be tested in a web browser, at http://localhost:8014/get-json , or using Firefox RESTClient (see below screenshots). The ControllerHttpServer class allows this content to display by creating an instance of the GetEmployeesJSONHandler class, and placing it at the "/get-json" server context.

Browser Output – GetEmployeesJSONHandler:

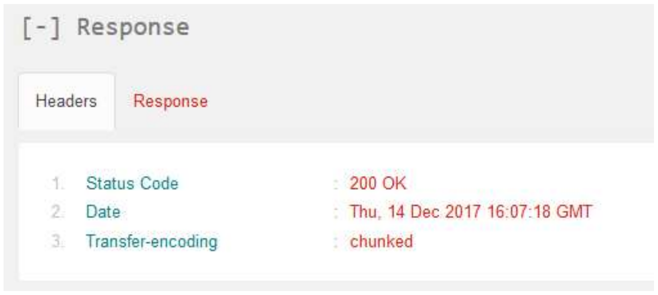


```
Employee Database - All Records (JSON Format):
[{"employeeNumber":1,"department":"Computing","startDate":"12-01-2000","salary":40000.0,"email":"james@mail.com","name":"James
Smith","gender":"M","dob":"14-07-1970","address":"Manchester","postcode":"M1 2FR"},
{"employeeNumber":2,"department":"Biology","startDate":"25-03-2009","salary":35000.0,"email":"sarah@mail.com","name":"Sarah
Jones","gender":"F","dob":"14-07-1980","address":"Stockport","postcode":"S8 7YT"},
{"employeeNumber":3,"department":"English","startDate":"04-09-1980","salary":50000.0,"email":"helen@mail.com","name":"Helen
Taylor","gender":"F","dob":"30-06-1950","address":"Oldham","postcode":"O3 5NP"},
{"employeeNumber":4,"department":"Nutrition","startDate":"08-10-1995","salary":38000.0,"email":"adam@mail.com","name":"Adam
Blake","gender":"M","dob":"28-03-1965","address":"Manchester","postcode":"M3 6ES"},
{"employeeNumber":5,"department":"Engineering","startDate":"24-11-1998","salary":40000.0,"email":"jane@mail.com","name":"Jane
Nolan","gender":"F","dob":"01-07-1970","address":"Edale","postcode":"E1 9QZ"},
{"employeeNumber":6,"department":"History","startDate":"23-07-1980","salary":45000.0,"email":"geoff@mail.com","name":"Geoff
Oliver","gender":"M","dob":"09-04-1950","address":"Stockport","postcode":"S1 5GV"},{"employeeNumber":7,"department":"Media
Studies","startDate":"23-07-2015","salary":35000.0,"email":"david@mail.com","name":"David
Thompson","gender":"M","dob":"09-04-1990","address":"Sheffield","postcode":"S3 2QW"},
{"employeeNumber":8,"department":"Mathematics","startDate":"18-09-2010","salary":35000.0,"email":"david@mail.com","name":"Sally
Bradshaw","gender":"F","dob":"30-12-1985","address":"Leeds","postcode":"L5 4FQ"},
{"employeeNumber":10,"department":"Music","startDate":"18-09-1989","salary":43000.0,"email":"jenny@mail.com","name":"Jenny
Watson","gender":"F","dob":"19-04-1960","address":"Liverpool","postcode":"L2 9IB"},
{"employeeNumber":11,"department":"Economics","startDate":"01-09-2012","salary":49000.0,"email":"sam@mail.com","name":"Sam
James","gender":"M","dob":"14-04-1960","address":"Stoke","postcode":"S1 2EQ"}]
```

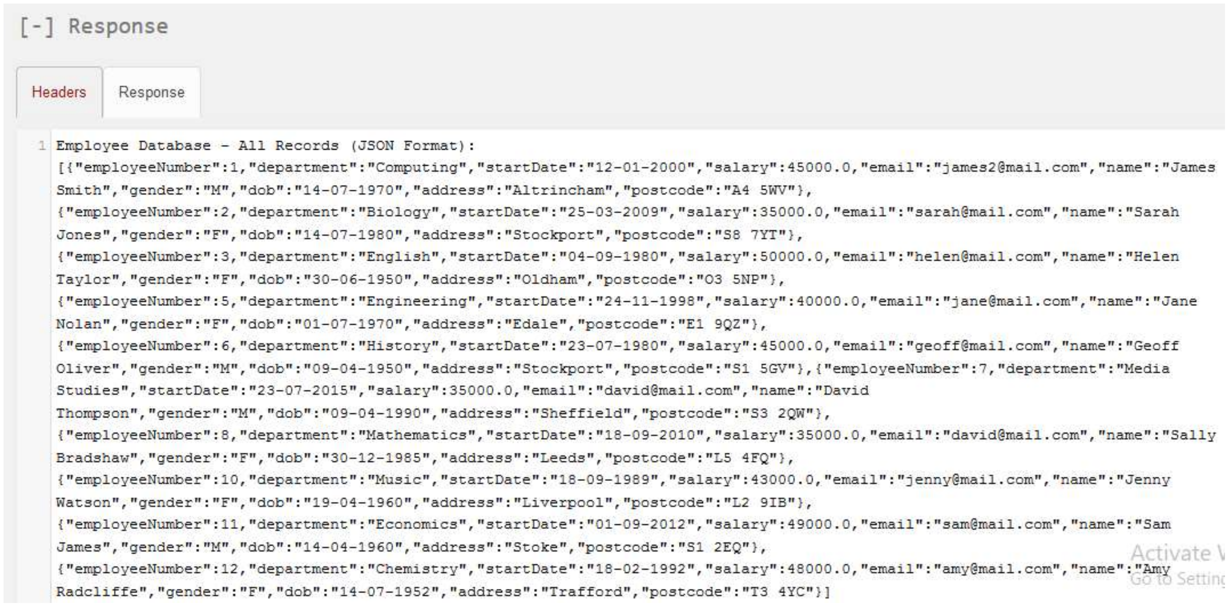
Testing of 'GetEmployeesJSON, Using the Firefox RESTclient:



```
POST  http://localhost:8014/get-json
```



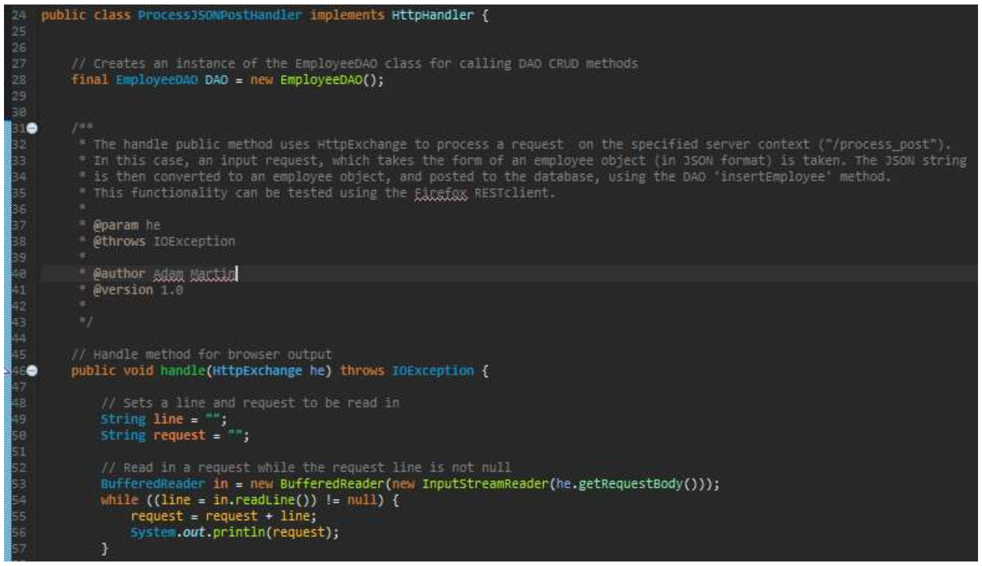
```
1. Status Code        : 200 OK
2. Date               : Thu, 14 Dec 2017 16:07:18 GMT
3. Transfer-encoding  : chunked
```



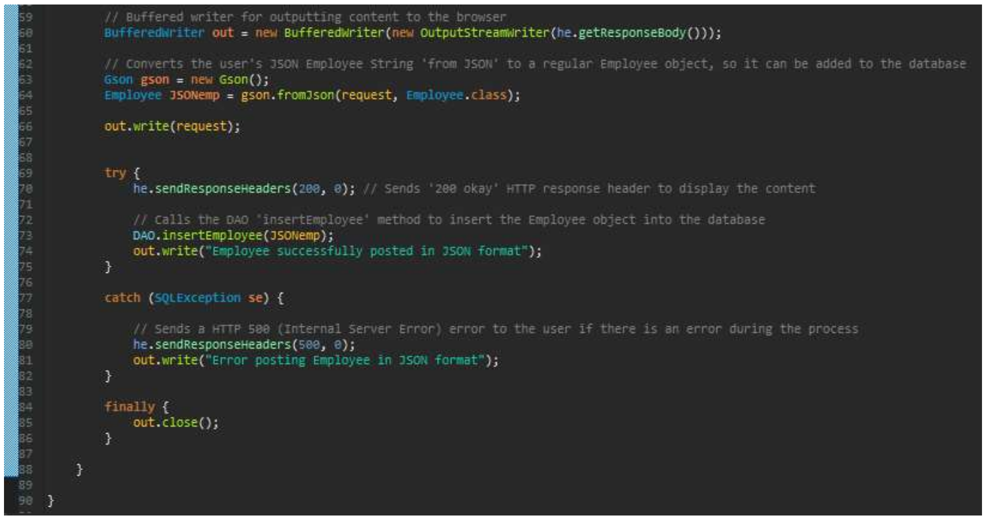
```
Employee Database - All Records (JSON Format):
[{"employeeNumber":1,"department":"Computing","startDate":"12-01-2000","salary":45000.0,"email":"james2@mail.com","name":"James
Smith","gender":"M","dob":"14-07-1970","address":"Altrincham","postcode":"A4 5WV"},
{"employeeNumber":2,"department":"Biology","startDate":"25-03-2009","salary":35000.0,"email":"sarah@mail.com","name":"Sarah
Jones","gender":"F","dob":"14-07-1980","address":"Stockport","postcode":"S8 7YT"},
{"employeeNumber":3,"department":"English","startDate":"04-09-1980","salary":50000.0,"email":"helen@mail.com","name":"Helen
Taylor","gender":"F","dob":"30-06-1950","address":"Oldham","postcode":"O3 5NP"},
{"employeeNumber":5,"department":"Engineering","startDate":"24-11-1998","salary":40000.0,"email":"jane@mail.com","name":"Jane
Nolan","gender":"F","dob":"01-07-1970","address":"Edale","postcode":"E1 9QZ"},
{"employeeNumber":6,"department":"History","startDate":"23-07-1980","salary":45000.0,"email":"geoff@mail.com","name":"Geoff
Oliver","gender":"M","dob":"09-04-1950","address":"Stockport","postcode":"S1 5GV"},{"employeeNumber":7,"department":"Media
Studies","startDate":"23-07-2015","salary":35000.0,"email":"david@mail.com","name":"David
Thompson","gender":"M","dob":"09-04-1990","address":"Sheffield","postcode":"S3 2QW"},
{"employeeNumber":8,"department":"Mathematics","startDate":"18-09-2010","salary":35000.0,"email":"david@mail.com","name":"Sally
Bradshaw","gender":"F","dob":"30-12-1985","address":"Leeds","postcode":"L5 4FQ"},
{"employeeNumber":10,"department":"Music","startDate":"18-09-1989","salary":43000.0,"email":"jenny@mail.com","name":"Jenny
Watson","gender":"F","dob":"19-04-1960","address":"Liverpool","postcode":"L2 9IB"},
{"employeeNumber":11,"department":"Economics","startDate":"01-09-2012","salary":49000.0,"email":"sam@mail.com","name":"Sam
James","gender":"M","dob":"14-04-1960","address":"Stoke","postcode":"S1 2EQ"},
{"employeeNumber":12,"department":"Chemistry","startDate":"18-02-1992","salary":48000.0,"email":"amy@mail.com","name":"Amy
Radcliffe","gender":"F","dob":"14-07-1952","address":"Trafford","postcode":"T3 4YC"}]
```

These screenshots depict the successful testing of the retrieval of all employees in JSON format. As you can see, the JSON employee records were successfully received, and a positive '200 OK' response header was returned.

Step 9 – ‘ProcessJSONPostHandler’ Class:



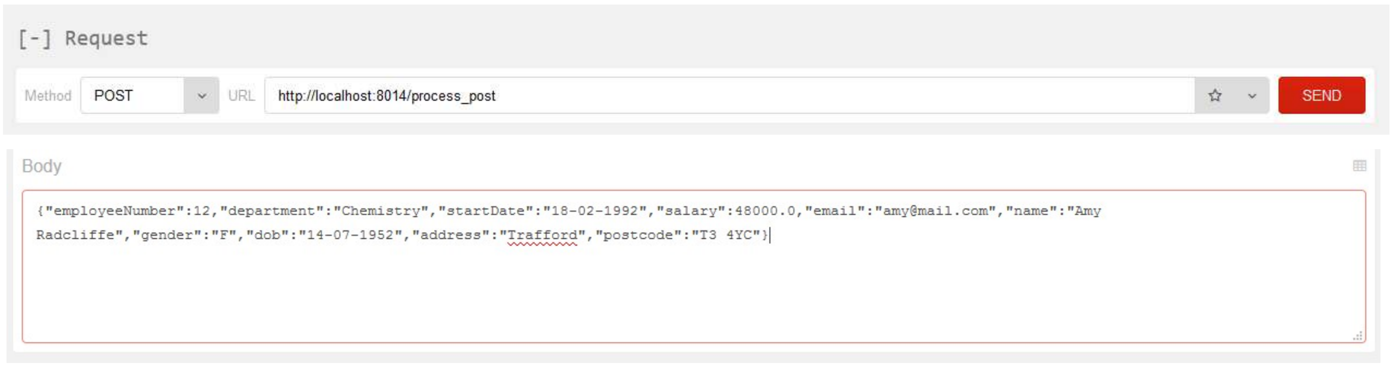
```java
public class ProcessJSONPostHandler implements HttpHandler {


    // Creates an instance of the EmployeeDAO class for calling DAO CRUD methods
    final EmployeeDAO DAO = new EmployeeDAO();


    /**
     * The handle public method uses HttpExchange to process a request  on the specified server context ("/process_post").
     * In this case, an input request, which takes the form of an employee object (in JSON format) is taken. The JSON string
     * is then converted to an employee object, and posted to the database, using the DAO 'insertEmployee' method.
     * This functionality can be tested using the Firefox RESTclient.
     *
     * @param he
     * @throws IOException
     *
     * @author Adam Martin
     * @version 1.0
     *
     */

    // Handle method for browser output
    public void handle(HttpExchange he) throws IOException {

        // Sets a line and request to be read in
        String line = "";
        String request = "";

        // Read in a request while the request line is not null
        BufferedReader in = new BufferedReader(new InputStreamReader(he.getRequestBody()));
        while ((line = in.readLine()) != null) {
            request = request + line;
            System.out.println(request);
        }
```



```java
        // Buffered writer for outputting content to the browser
        BufferedWriter out = new BufferedWriter(new OutputStreamWriter(he.getResponseBody()));

        // Converts the user's JSON Employee String 'from JSON' to a regular Employee object, so it can be added to the database
        Gson gson = new Gson();
        Employee JSONemp = gson.fromJson(request, Employee.class);

        out.write(request);


        try {
            he.sendResponseHeaders(200, 0); // Sends '200 okay' HTTP response header to display the content

            // Calls the DAO 'insertEmployee' method to insert the Employee object into the database
            DAO.insertEmployee(JSONemp);
            out.write("Employee successfully posted in JSON format");
        }

        catch (SQLException se) {

            // Sends a HTTP 500 (Internal Server Error) error to the user if there is an error during the process
            he.sendResponseHeaders(500, 0);
            out.write("Error posting Employee in JSON format");
        }

        finally {
            out.close();
        }

    }

}
```

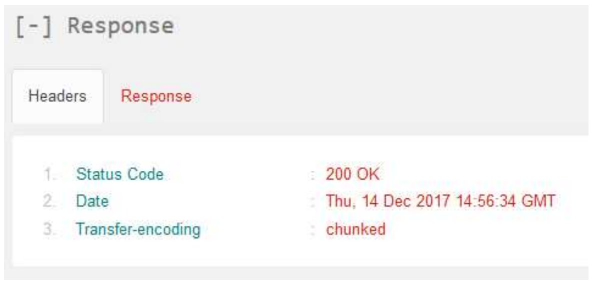
The ProcessJSONPostHandler class includes code for allowing the user to post an Employee record to the database, in JSON format. This code takes in a request, which takes the form of a JSON Employee string. The JSON string request is then converted to a regular employee object, and added to the database via the calling of the EmployeeDAO insertEmployee method. This functionality can be tested using the Firefox RESTClient (see below screenshots). The ControllerHttpServer class sets up this handler by creating an instance of the ProcessJSONPostHandler class, and placing it at the “/process_post” server context.

Testing of JSON Post, Using the Firefox RESTclient:

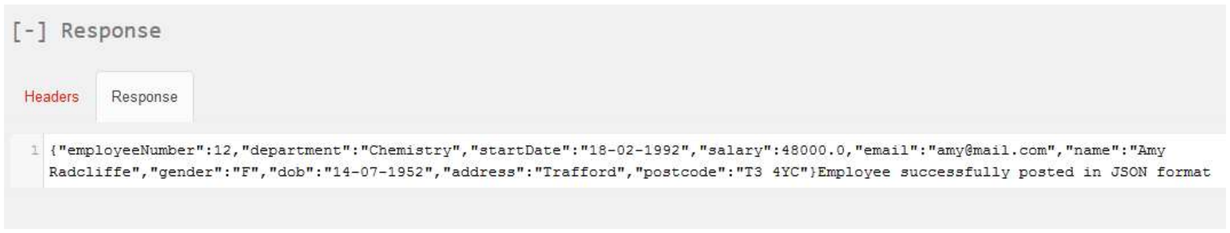


POST http://localhost:8014/process_post

```
{"employeeNumber":12,"department":"Chemistry","startDate":"18-02-1992","salary":48000.0,"email":"amy@mail.com","name":"Amy
Radcliffe","gender":"F","dob":"14-07-1952","address":"Trafford","postcode":"T3 4YC"}
```



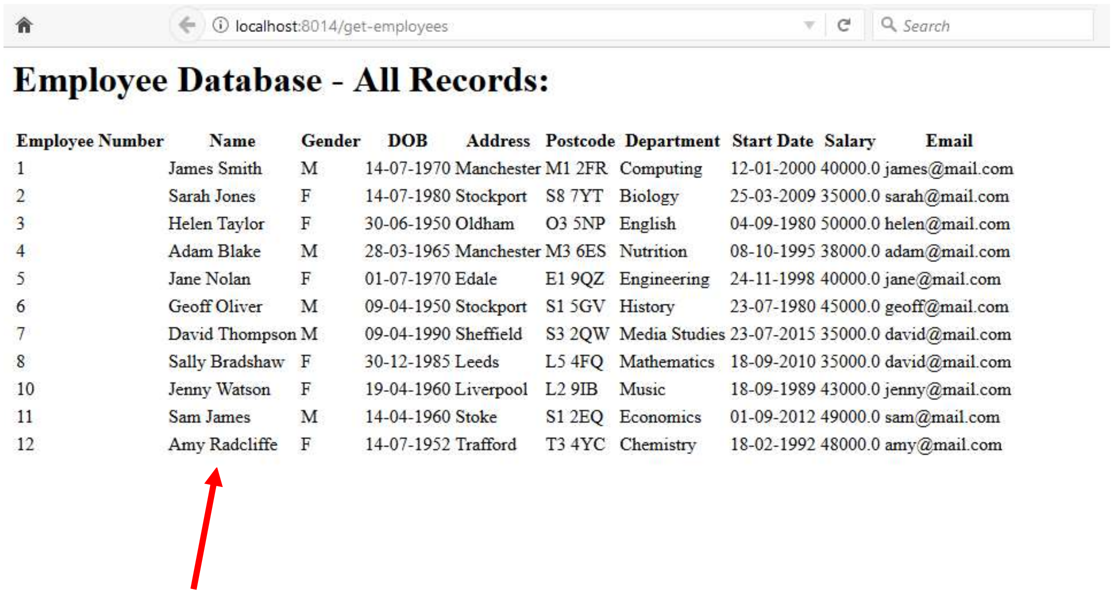
```
1. Status Code         : 200 OK
2. Date                : Thu, 14 Dec 2017 14:56:34 GMT
3. Transfer-encoding   : chunked
```



```
{"employeeNumber":12,"department":"Chemistry","startDate":"18-02-1992","salary":48000.0,"email":"amy@mail.com","name":"Amy
Radcliffe","gender":"F","dob":"14-07-1952","address":"Trafford","postcode":"T3 4YC"}Employee successfully posted in JSON format
```

These screenshots depict the successful testing of the JSON post. As you can see, the employee was successfully posted, and a positive '200 OK' response header was returned.

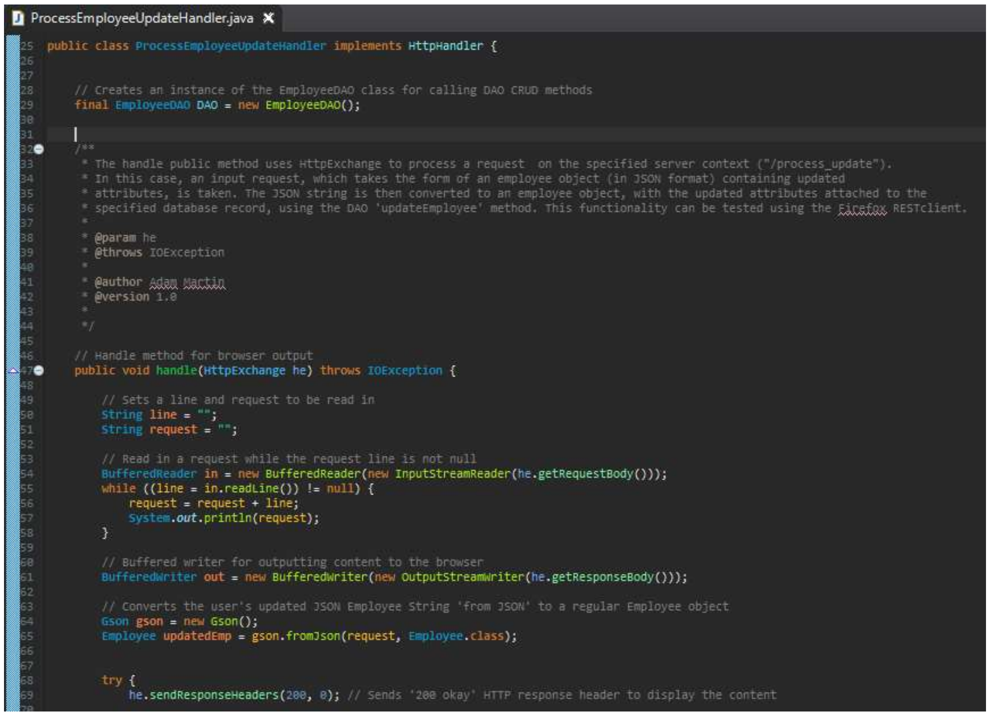
Evidence of the New Employee Record (12, Amy Radcliffe) in the Database:



Employee Database - All Records:

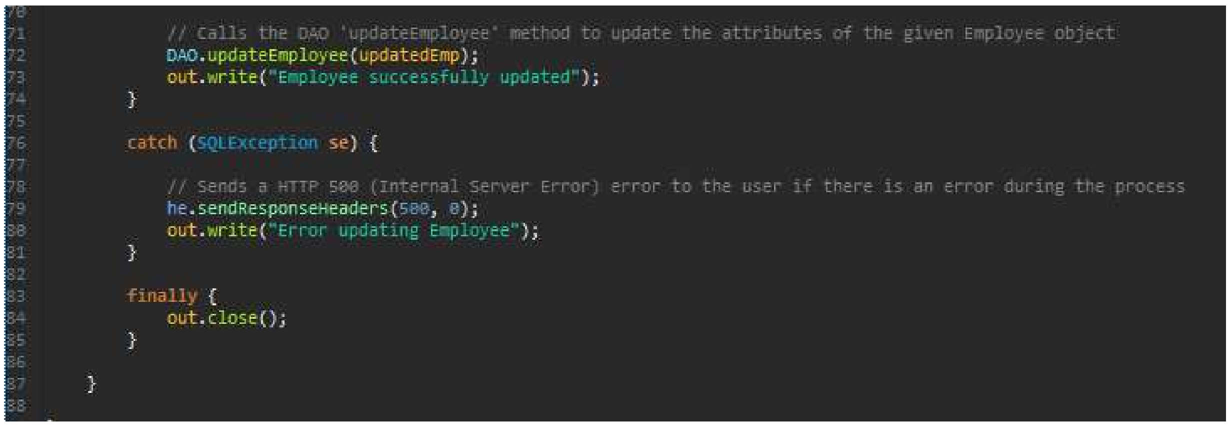
Employee Number	Name	Gender	DOB	Address	Postcode	Department	Start Date	Salary	Email
1	James Smith	M	14-07-1970	Manchester	M1 2FR	Computing	12-01-2000	40000.0	james@mail.com
2	Sarah Jones	F	14-07-1980	Stockport	S8 7YT	Biology	25-03-2009	35000.0	sarah@mail.com
3	Helen Taylor	F	30-06-1950	Oldham	O3 5NP	English	04-09-1980	50000.0	helen@mail.com
4	Adam Blake	M	28-03-1965	Manchester	M3 6ES	Nutrition	08-10-1995	38000.0	adam@mail.com
5	Jane Nolan	F	01-07-1970	Edale	E1 9QZ	Engineering	24-11-1998	40000.0	jane@mail.com
6	Geoff Oliver	M	09-04-1950	Stockport	S1 5GV	History	23-07-1980	45000.0	geoff@mail.com
7	David Thompson	M	09-04-1990	Sheffield	S3 2QW	Media Studies	23-07-2015	35000.0	david@mail.com
8	Sally Bradshaw	F	30-12-1985	Leeds	L5 4FQ	Mathematics	18-09-2010	35000.0	david@mail.com
10	Jenny Watson	F	19-04-1960	Liverpool	L2 9IB	Music	18-09-1989	43000.0	jenny@mail.com
11	Sam James	M	14-04-1960	Stoke	S1 2EQ	Economics	01-09-2012	49000.0	sam@mail.com
12	Amy Radcliffe	F	14-07-1952	Trafford	T3 4YC	Chemistry	18-02-1992	48000.0	amy@mail.com

Step 10 – ‘ProcessEmployeeUpdateHandler’ Class:



ProcessEmployeeUpdateHandler.java

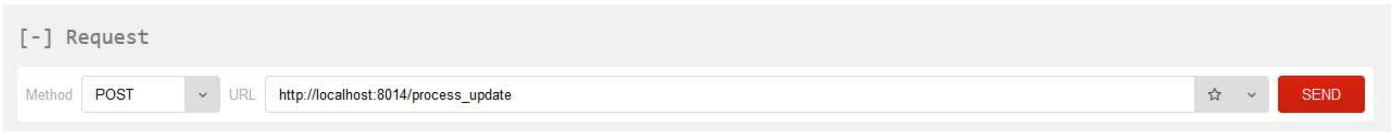
```java
public class ProcessEmployeeUpdateHandler implements HttpHandler {


    // Creates an instance of the EmployeeDAO class for calling DAO CRUD methods
    final EmployeeDAO DAO = new EmployeeDAO();

    
    /**
     * The handle public method uses HttpExchange to process a request  on the specified server context ("/process_update").
     * In this case, an input request, which takes the form of an employee object (in JSON format) containing updated
     * attributes, is taken. The JSON string is then converted to an employee object, with the updated attributes attached to the
     * specified database record, using the DAO 'updateEmployee' method. This functionality can be tested using the Firefox RESTclient.
     *
     * @param he
     * @throws IOException
     *
     * @author Adam Martin
     * @version 1.0
     *
     */

    // Handle method for browser output
    public void handle(HttpExchange he) throws IOException {

        // Sets a line and request to be read in
        String line = "";
        String request = "";

        // Read in a request while the request line is not null
        BufferedReader in = new BufferedReader(new InputStreamReader(he.getRequestBody()));
        while ((line = in.readLine()) != null) {
            request = request + line;
            System.out.println(request);
        }

        // Buffered writer for outputting content to the browser
        BufferedWriter out = new BufferedWriter(new OutputStreamWriter(he.getResponseBody()));

        // Converts the user's updated JSON Employee String 'from JSON' to a regular Employee object
        Gson gson = new Gson();
        Employee updatedEmp = gson.fromJson(request, Employee.class);


        try {
            he.sendResponseHeaders(200, 0); // Sends '200 okay' HTTP response header to display the content
```



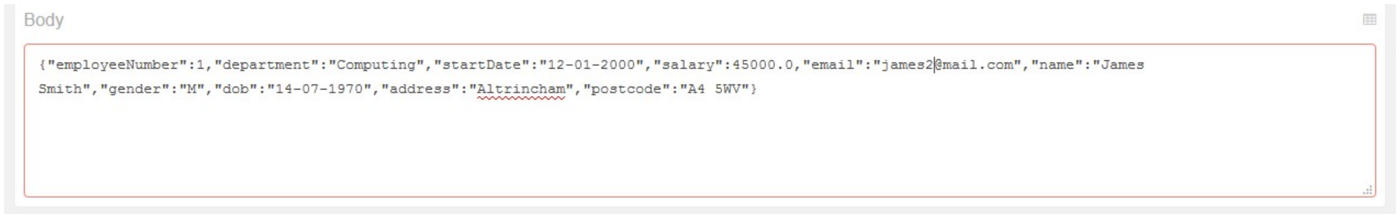
```java

            // Calls the DAO 'updateEmployee' method to update the attributes of the given Employee object
            DAO.updateEmployee(updatedEmp);
            out.write("Employee successfully updated");
        }

        catch (SQLException se) {

            // Sends a HTTP 500 (Internal Server Error) error to the user if there is an error during the process
            he.sendResponseHeaders(500, 0);
            out.write("Error updating Employee");
        }

        finally {
            out.close();
        }

    }

```

The ProcessEmployeeUpdateHandler class includes code for allowing the user to update all of the attributes of an employee record, with a specified employee number. This code takes in a request, which takes the form of an existing employee record, in JSON format. The attributes of the employee can then be edited, and posted to the database, via the calling of the EmployeeDAO updateEmployee method. This functionality can be tested using the Firefox RESTClient (see below screenshots). The ControllerHttpServer class sets up this handler by creating an instance of the ProcessEmployeeUpdateHandler class, and placing it at the “/process_update” server context.

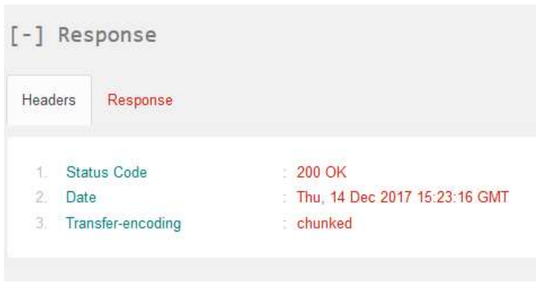
Testing of the Server-Side Update, Using the Firefox RESTclient:


[-] Request

Method POST URL http://localhost:8014/process_update SEND

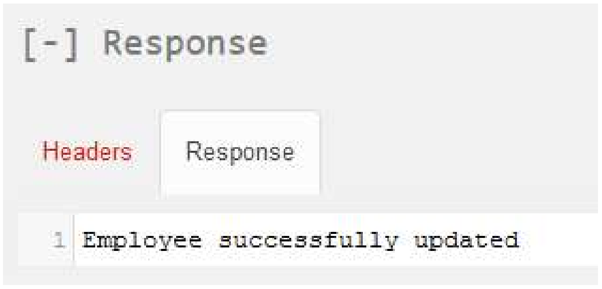

Body

```
{"employeeNumber":1,"department":"Computing","startDate":"12-01-2000","salary":45000.0,"email":"james2@mail.com","name":"James Smith","gender":"M","dob":"14-07-1970","address":"Altrincham","postcode":"A4 5WV"}
```


[-] Response

Headers Response

1.	Status Code	: 200 OK
2.	Date	: Thu, 14 Dec 2017 15:23:16 GMT
3.	Transfer-encoding	: chunked


[-] Response

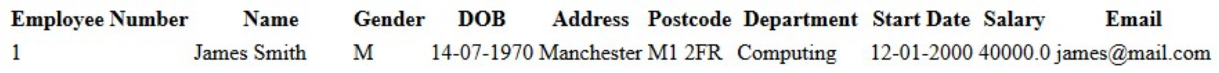
Headers Response

```
Employee successfully updated
```

These screenshots depict the successful testing of the employee update. A request to update James Smith's (Employee 1) salary, email address, address and postcode was sent (it is possible for the user to update other attributes if they wish). As you can see, the employee was successfully updated, and a positive '200 OK' response header was returned.

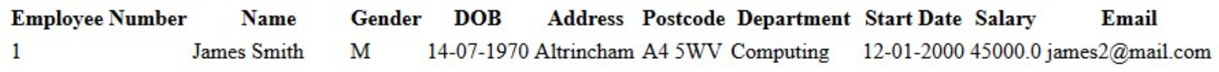
Evidence of the Updated Employee Record (1, James Smith) in the Database:

Existing:



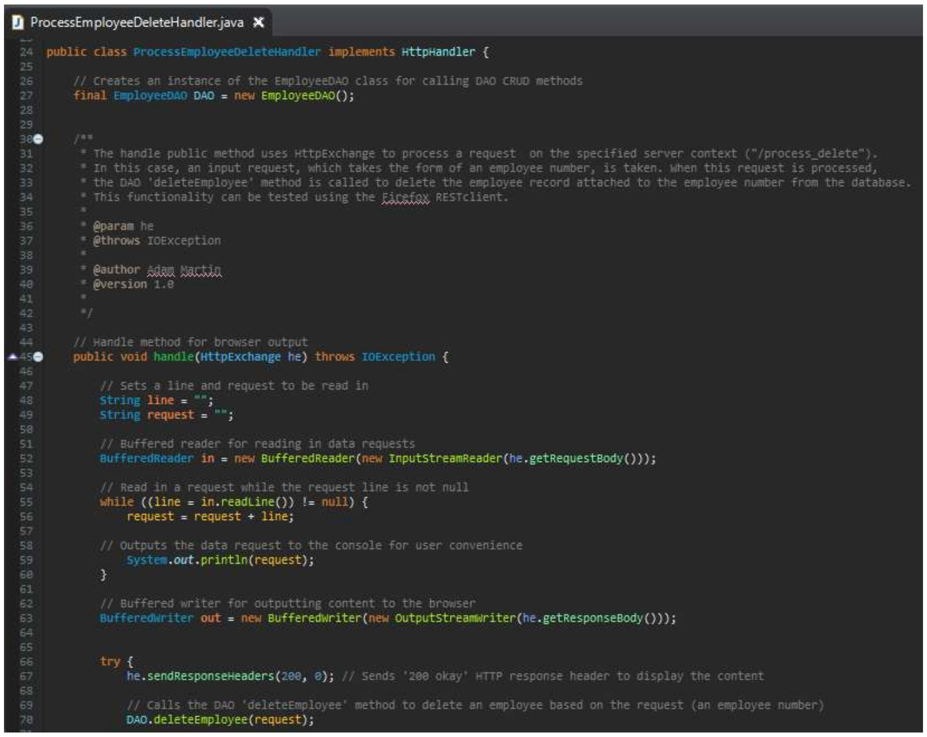
Employee Number	Name	Gender	DOB	Address	Postcode	Department	Start Date	Salary	Email
1	James Smith	M	14-07-1970	Manchester	M1 2FR	Computing	12-01-2000	40000.0	james@mail.com

New (updated salary, email address, address and postcode):



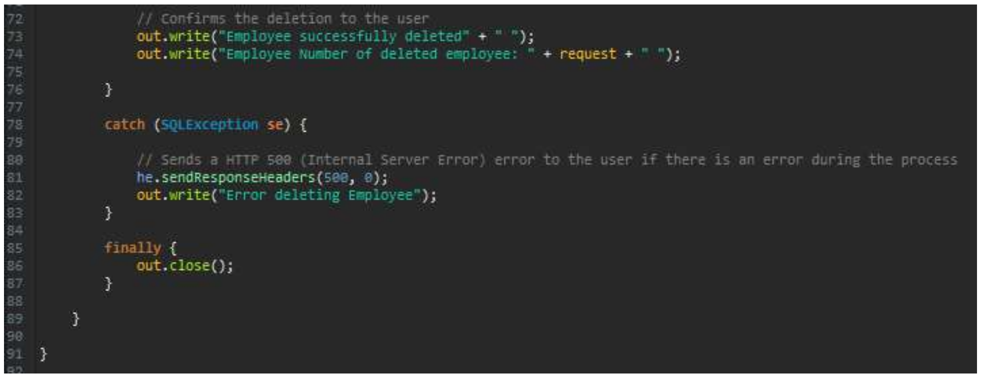
Employee Number	Name	Gender	DOB	Address	Postcode	Department	Start Date	Salary	Email
1	James Smith	M	14-07-1970	Altrincham	A4 5WV	Computing	12-01-2000	45000.0	james2@mail.com

Step 11 – ‘ProcessEmployeeDeleteHandler’ Class:



ProcessEmployeeDeleteHandler.java

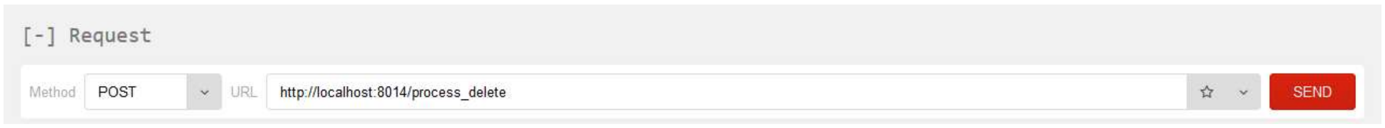
```java
public class ProcessEmployeeDeleteHandler implements HttpHandler {

    // Creates an instance of the EmployeeDAO class for calling DAO CRUD methods
    final EmployeeDAO DAO = new EmployeeDAO();


    /**
     * The handle public method uses HttpExchange to process a request  on the specified server context ("/process_delete").
     * In this case, an input request, which takes the form of an employee number, is taken. When this request is processed,
     * the DAO 'deleteEmployee' method is called to delete the employee record attached to the employee number from the database.
     * This functionality can be tested using the Firefox RESTclient.
     *
     * @param he
     * @throws IOException
     *
     * @author Adam Martin
     * @version 1.0
     *
     */

    // Handle method for browser output
    public void handle(HttpExchange he) throws IOException {

        // Sets a line and request to be read in
        String line = "";
        String request = "";

        // Buffered reader for reading in data requests
        BufferedReader in = new BufferedReader(new InputStreamReader(he.getRequestBody()));

        // Read in a request while the request line is not null
        while ((line = in.readLine()) != null) {
            request = request + line;

        // Outputs the data request to the console for user convenience
            System.out.println(request);
        }

        // Buffered writer for outputting content to the browser
        BufferedWriter out = new BufferedWriter(new OutputStreamWriter(he.getResponseBody()));


        try {
            he.sendResponseHeaders(200, 0); // Sends '200 okay' HTTP response header to display the content

            // Calls the DAO 'deleteEmployee' method to delete an employee based on the request (an employee number)
            DAO.deleteEmployee(request);
```



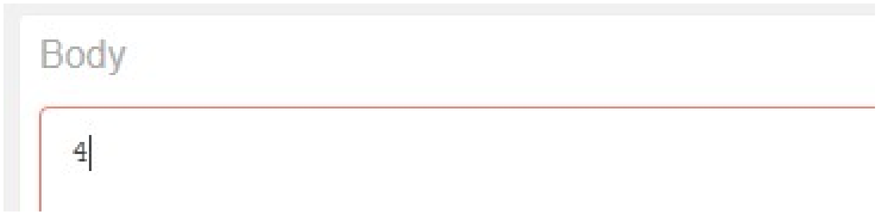
```java
            // Confirms the deletion to the user
            out.write("Employee successfully deleted" + " ");
            out.write("Employee Number of deleted employee: " + request + " ");

        }

        catch (SQLException se) {

            // Sends a HTTP 500 (Internal Server Error) error to the user if there is an error during the process
            he.sendResponseHeaders(500, 0);
            out.write("Error deleting Employee");
        }

        finally {
            out.close();
        }

    }

}
```

The ProcessEmployeeDeleteHandler class includes code for allowing the user to delete an employee record with a specified employee number. This code takes in a request, which takes the form of an existing employee number. The employee record attached to that employee number is then deleted, , via the calling of the EmployeeDAO deleteEmployee method. This functionality can be tested using the Firefox RESTClient (see below screenshots). The ControllerHttpServer class sets up this handler by creating an instance of the ProcessEmployeeDeleteHandler class, and placing it at the “/process_delete” server context.

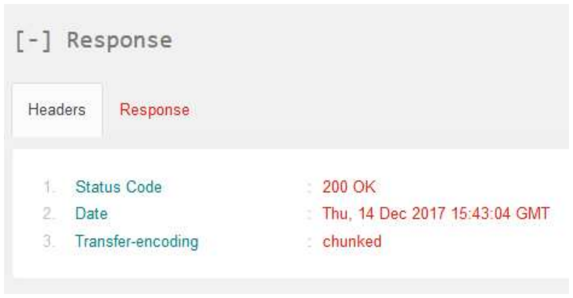
Testing of the Server-Side Delete, Using the Firefox RESTclient:



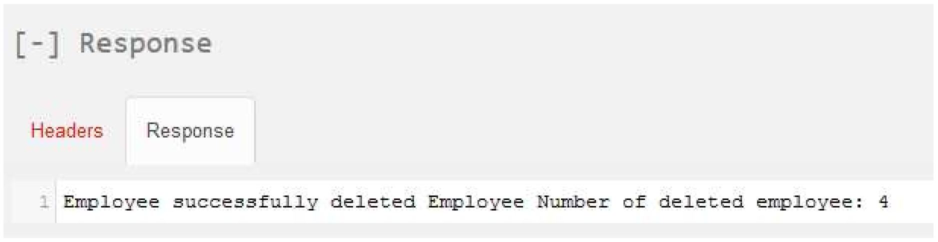
Method: POST
URL: http://localhost:8014/process_delete



```
4
```



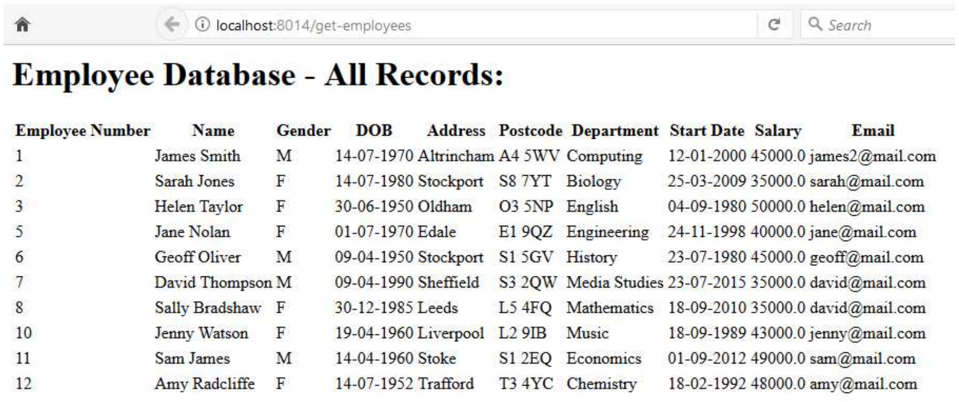
```
1. Status Code          : 200 OK
2. Date                 : Thu, 14 Dec 2017 15:43:04 GMT
3. Transfer-encoding    : chunked
```



```
Employee successfully deleted Employee Number of deleted employee: 4
```

These screenshots depict the successful testing of the employee delete. A request to delete the employee record with an employee number of '4' was sent. As you can see, the employee was successfully deleted, and a positive '200 OK' response header was returned.

Evidence of the Deletion of Employee Record (4) from the Database:

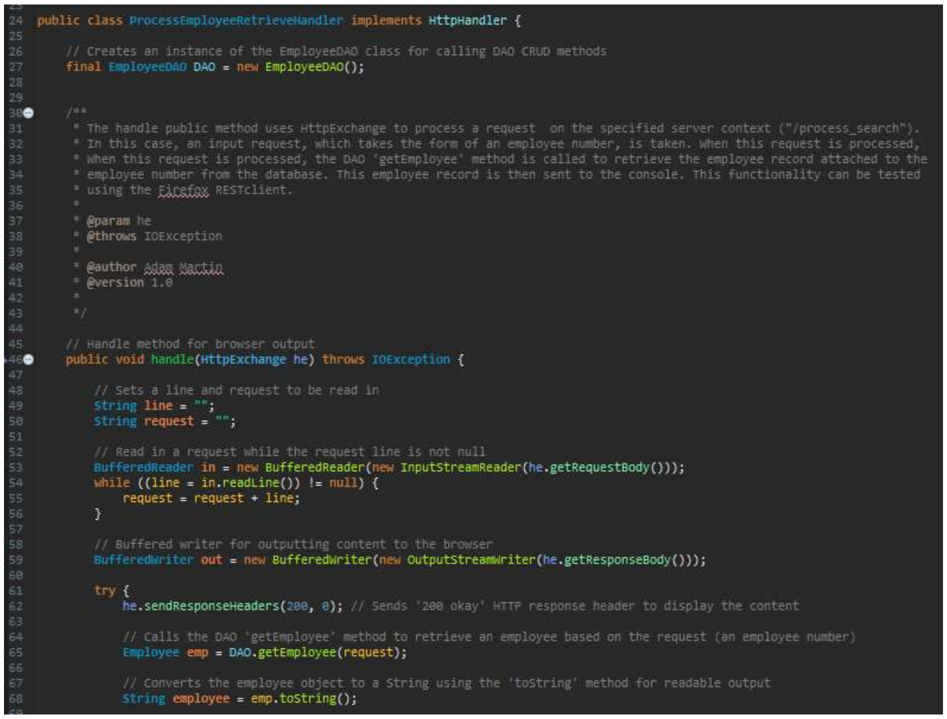


Employee Database - All Records:

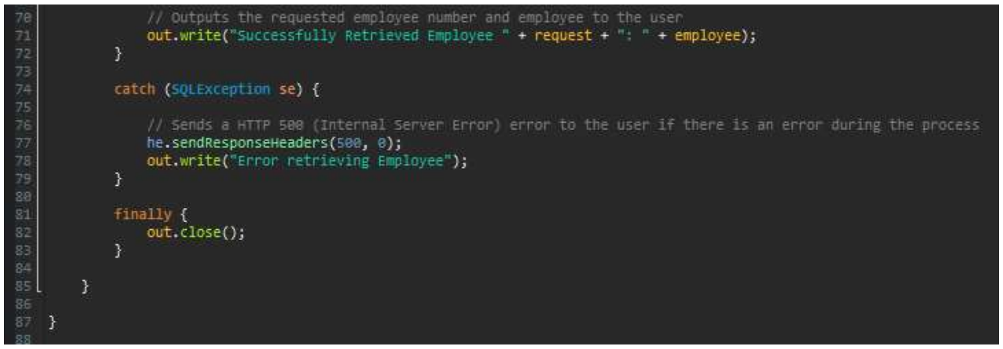
Employee Number	Name	Gender	DOB	Address	Postcode	Department	Start Date	Salary	Email
1	James Smith	M	14-07-1970	Altrincham	A4 5WV	Computing	12-01-2000	45000.0	james2@mail.com
2	Sarah Jones	F	14-07-1980	Stockport	S8 7YT	Biology	25-03-2009	35000.0	sarah@mail.com
3	Helen Taylor	F	30-06-1950	Oldham	O3 5NP	English	04-09-1980	50000.0	helen@mail.com
5	Jane Nolan	F	01-07-1970	Edale	E1 9QZ	Engineering	24-11-1998	40000.0	jane@mail.com
6	Geoff Oliver	M	09-04-1950	Stockport	S1 5GV	History	23-07-1980	45000.0	geoff@mail.com
7	David Thompson	M	09-04-1990	Sheffield	S3 2QW	Media Studies	23-07-2015	35000.0	david@mail.com
8	Sally Bradshaw	F	30-12-1985	Leeds	L5 4FQ	Mathematics	18-09-2010	35000.0	david@mail.com
10	Jenny Watson	F	19-04-1960	Liverpool	L2 9IB	Music	18-09-1989	43000.0	jenny@mail.com
11	Sam James	M	14-04-1960	Stoke	S1 2EQ	Economics	01-09-2012	49000.0	sam@mail.com
12	Amy Radcliffe	F	14-07-1952	Trafford	T3 4YC	Chemistry	18-02-1992	48000.0	amy@mail.com

As you can see, an employee record with an employee number of '4' no longer exists in the database.

Step 12 – 'ProcessEmployeeRetrieveHandler' Class:



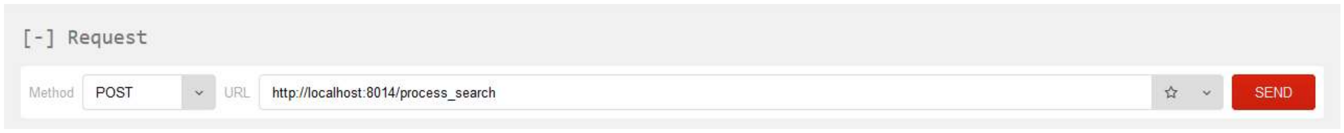
```java
public class ProcessEmployeeRetrieveHandler implements HttpHandler {

    // Creates an instance of the EmployeeDAO class for calling DAO CRUD methods
    final EmployeeDAO DAO = new EmployeeDAO();


    /**
     * The handle public method uses HttpExchange to process a request  on the specified server context ("/process_search").
     * In this case, an input request, which takes the form of an employee number, is taken. When this request is processed,
     * When this request is processed, the DAO 'getEmployee' method is called to retrieve the employee record attached to the
     * employee number from the database. This employee record is then sent to the console. This functionality can be tested
     * using the Firefox RESTclient.
     *
     * @param he
     * @throws IOException
     *
     * @author Adam Martin
     * @version 1.0
     *
     */

    // Handle method for browser output
    public void handle(HttpExchange he) throws IOException {

        // Sets a line and request to be read in
        String line = "";
        String request = "";

        // Read in a request while the request line is not null
        BufferedReader in = new BufferedReader(new InputStreamReader(he.getRequestBody()));
        while ((line = in.readLine()) != null) {
            request = request + line;
        }

        // Buffered writer for outputting content to the browser
        BufferedWriter out = new BufferedWriter(new OutputStreamWriter(he.getResponseBody()));

        try {
            he.sendResponseHeaders(200, 0); // Sends '200 okay' HTTP response header to display the content

            // Calls the DAO 'getEmployee' method to retrieve an employee based on the request (an employee number)
            Employee emp = DAO.getEmployee(request);

            // Converts the employee object to a String using the 'toString' method for readable output
            String employee = emp.toString();
```

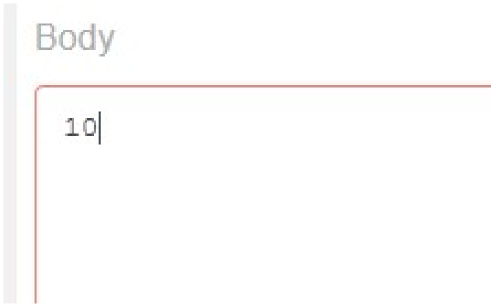


```java
            // Outputs the requested employee number and employee to the user
            out.write("Successfully Retrieved Employee " + request + ": " + employee);
        }

        catch (SQLException se) {

            // Sends a HTTP 500 (Internal Server Error) error to the user if there is an error during the process
            he.sendResponseHeaders(500, 0);
            out.write("Error retrieving Employee");
        }

        finally {
            out.close();
        }

    }

}
```

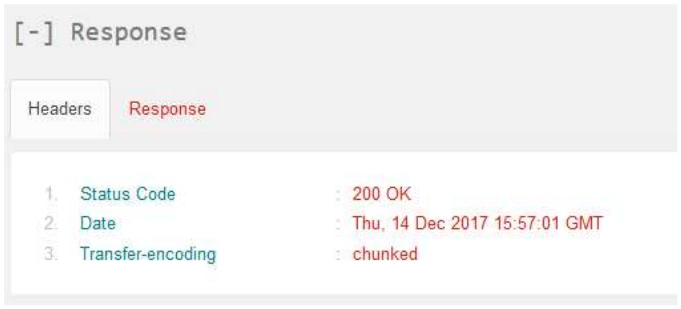
The ProcessEmployeeRetrieveHandler class includes code for allowing the user to search the database for a single employee record with a specified employee number. This code takes in a request, which takes the form of an existing employee number. The employee record attached to that employee number is then located, and posted in the RESTclient body and the Eclipse console, via the calling of the EmployeeDAO getEmployee method. This functionality can be tested using the Firefox RESTClient (see below screenshots). The ControllerHttpServer class sets up this handler by creating an instance of the ProcessEmployeeRetrieveHandler class, and placing it at the "/process_search" server context.

Testing of the Server-Side Single Employee Retrieve, Using the Firefox RESTclient:

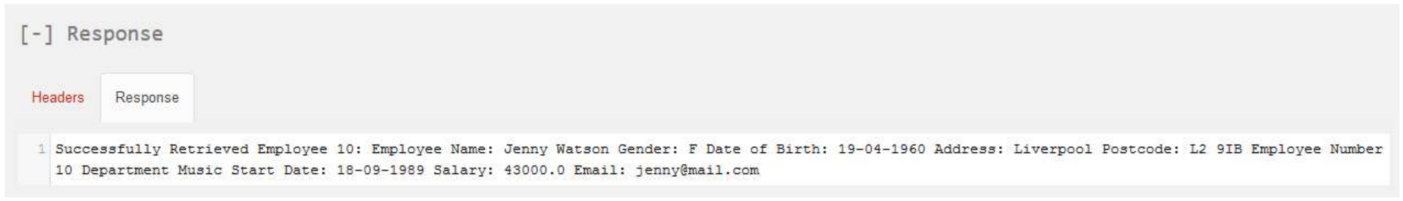




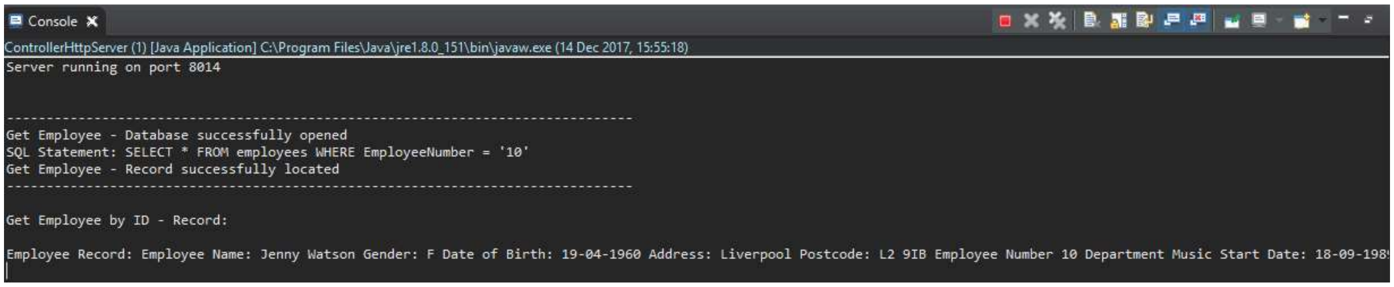
```
10
```



```
1. Status Code          : 200 OK
2. Date                 : Thu, 14 Dec 2017 15:57:01 GMT
3. Transfer-encoding    : chunked
```



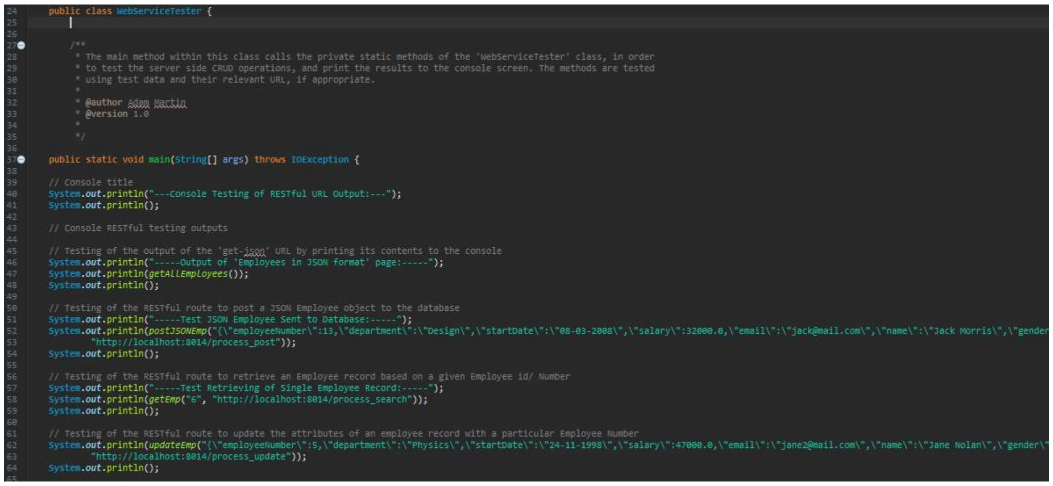
```
Successfully Retrieved Employee 10: Employee Name: Jenny Watson Gender: F Date of Birth: 19-04-1960 Address: Liverpool Postcode: L2 9IB Employee Number
10 Department Music Start Date: 18-09-1989 Salary: 43000.0 Email: jenny@mail.com
```



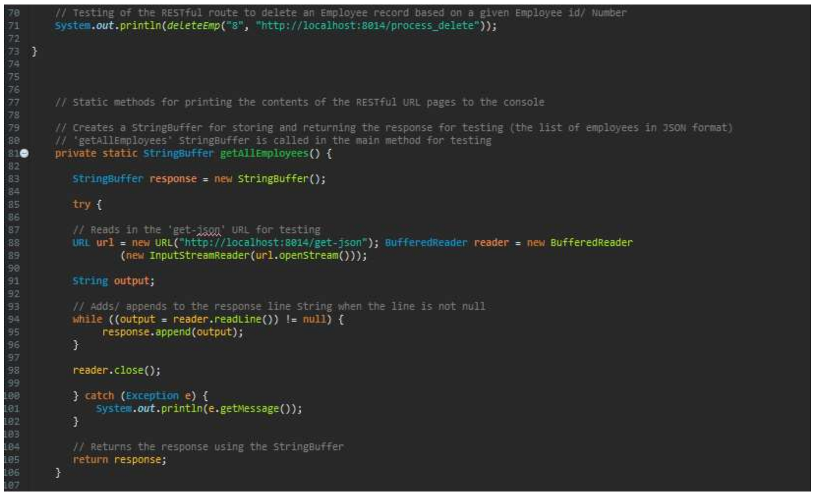
```
ControllerHttpServer (1) [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (14 Dec 2017, 15:55:18)
Server running on port 8014

------------------------------------------------------------------------------
Get Employee - Database successfully opened
SQL Statement: SELECT * FROM employees WHERE EmployeeNumber = '10'
Get Employee - Record successfully located
------------------------------------------------------------------------------

Get Employee by ID - Record:

Employee Record: Employee Name: Jenny Watson Gender: F Date of Birth: 19-04-1960 Address: Liverpool Postcode: L2 9IB Employee Number 10 Department Music Start Date: 18-09-198
```

These screenshots depict the successful testing of the single employee retrieve. A request to delete the employee record with an employee number of ‘4’ was sent. As you can see, the employee was successfully retrieved, and a positive ‘200 OK’ response header was returned. The employee record displays in the ‘response’ section of the RESTclient, and in the Eclipse console.

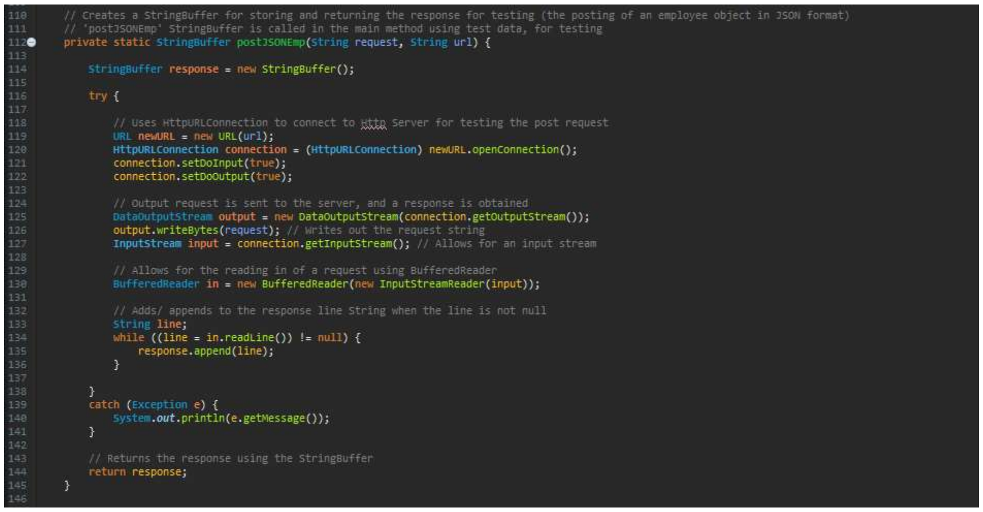
Step 13 – ‘WebServiceTester’ Class:



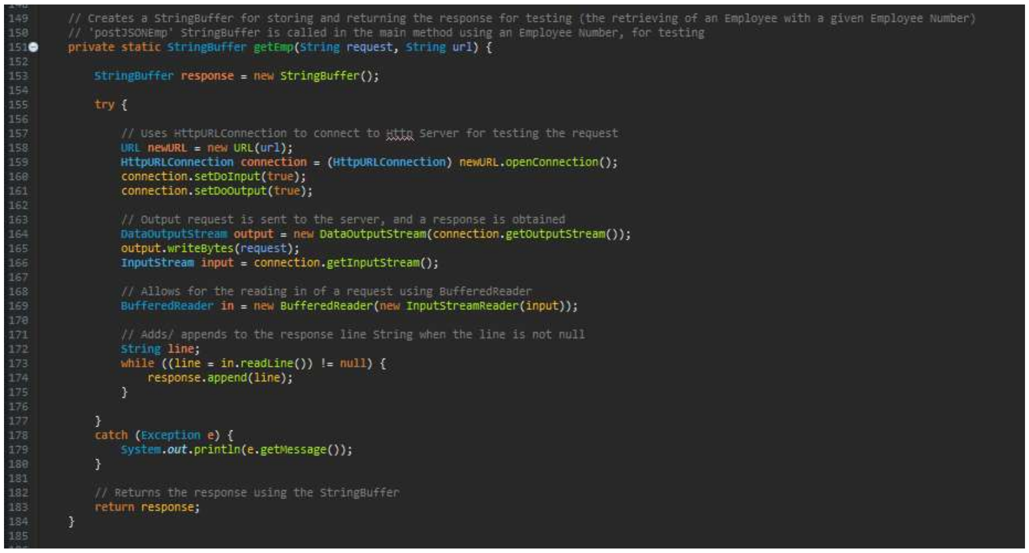
```
    public class WebServiceTester {
        


        /**
         * The main method within this class calls the private static methods of the 'WebServiceTester' class, in order
         * to test the server side CRUD operations, and print the results to the console screen. The methods are tested
         * using test data and their relevant URL, if appropriate.
         *
         * @author Adam Martin
         * @version 1.0
         *
         */

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

    // Console title
    System.out.println("---Console Testing of RESTful URL Output:---");
    System.out.println();

    // Console RESTful testing outputs

    // Testing of the output of the 'get-json' URL by printing its contents to the console
    System.out.println("-----Output of 'Employees in JSON format' page:-----");
    System.out.println(getAllEmployees());
    System.out.println();

    // Testing of the RESTful route to post a JSON Employee object to the database
    System.out.println("-----Test JSON Employee Sent to Database:-----");
    System.out.println(postJSONEmp("{\"employeeNumber\":13,\"department\":\"Design\",\"startDate\":\"08-03-2008\",\"salary\":32000.0,\"email\":\"jack@mail.com\",\"name\":\"Jack Morris\",\"gender
            "http://localhost:8014/process_post"));
    System.out.println();

    // Testing of the RESTful route to retrieve an Employee record based on a given Employee id/ Number
    System.out.println("-----Test Retrieving of Single Employee Record:-----");
    System.out.println(getEmp("6", "http://localhost:8014/process_search"));
    System.out.println();

    // Testing of the RESTful route to update the attributes of an employee record with a particular Employee Number
    System.out.println(updateEmp("{\"employeeNumber\":5,\"department\":\"Physics\",\"startDate\":\"24-11-1998\",\"salary\":47000.0,\"email\":\"jane2@mail.com\",\"name\":\"Jane Nolan\",\"gender\"
            "http://localhost:8014/process_update"));
    System.out.println();
```



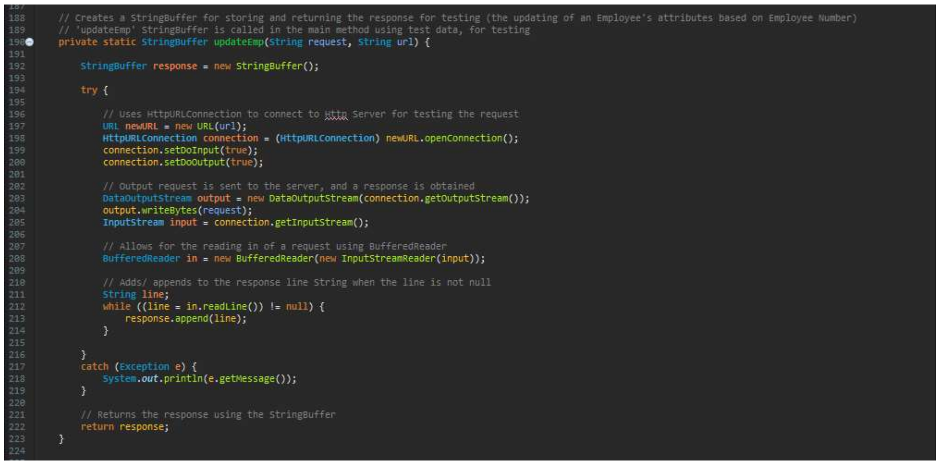
```
    // Testing of the RESTful route to delete an Employee record based on a given Employee id/ Number
    System.out.println(deleteEmp("8", "http://localhost:8014/process_delete"));

}



    // Static methods for printing the contents of the RESTful URL pages to the console

    // Creates a StringBuffer for storing and returning the response for testing (the list of employees in JSON format)
    // 'getAllEmployees' StringBuffer is called in the main method for testing
    private static StringBuffer getAllEmployees() {

        StringBuffer response = new StringBuffer();

        try {

        // Reads in the 'get-json' URL for testing
        URL url = new URL("http://localhost:8014/get-json"); BufferedReader reader = new BufferedReader
                (new InputStreamReader(url.openStream()));

        String output;

        // Adds/ appends to the response line String when the line is not null
        while ((output = reader.readLine()) != null) {
             response.append(output);
        }

        reader.close();

        } catch (Exception e) {
            System.out.println(e.getMessage());
        }

        // Returns the response using the StringBuffer
        return response;
    }
```



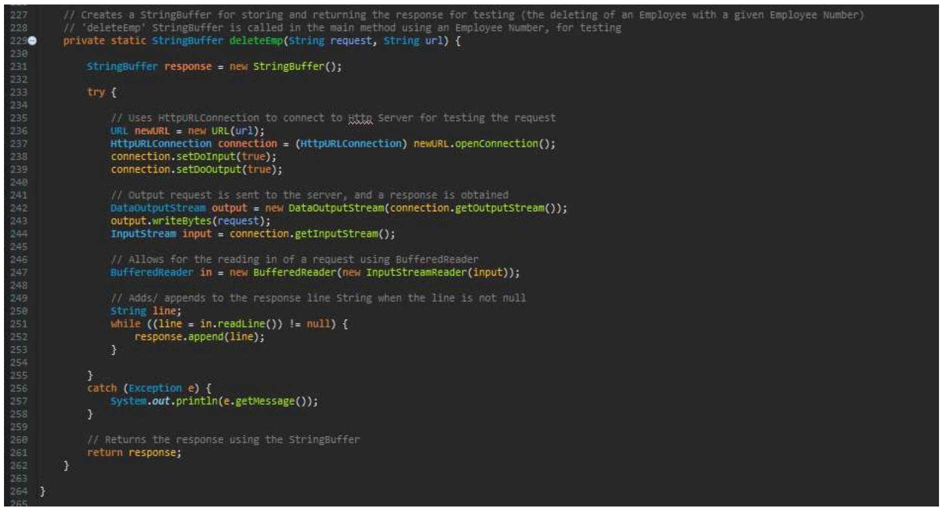
```
    // Creates a StringBuffer for storing and returning the response for testing (the posting of an employee object in JSON format)
    // 'postJSONEmp' StringBuffer is called in the main method using test data, for testing
    private static StringBuffer postJSONEmp(String request, String url) {

        StringBuffer response = new StringBuffer();

        try {

            // Uses HttpURLConnection to connect to Http Server for testing the post request
            URL newURL = new URL(url);
            HttpURLConnection connection = (HttpURLConnection) newURL.openConnection();
            connection.setDoInput(true);
            connection.setDoOutput(true);

            // Output request is sent to the server, and a response is obtained
            DataOutputStream output = new DataOutputStream(connection.getOutputStream());
            output.writeBytes(request); // Writes out the request string
            InputStream input = connection.getInputStream(); // Allows for an input stream

            // Allows for the reading in of a request using BufferedReader
            BufferedReader in = new BufferedReader(new InputStreamReader(input));

            // Adds/ appends to the response line String when the line is not null
            String line;
            while ((line = in.readLine()) != null) {
                response.append(line);
            }

        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }

        // Returns the response using the StringBuffer
        return response;
    }
```



```java
    // Creates a StringBuffer for storing and returning the response for testing (the retrieving of an Employee with a given Employee Number)
    // 'postJSONEmp' StringBuffer is called in the main method using an Employee Number, for testing
    private static StringBuffer getEmp(String request, String url) {

        StringBuffer response = new StringBuffer();

        try {

            // Uses HttpURLConnection to connect to Http Server for testing the request
            URL newURL = new URL(url);
            HttpURLConnection connection = (HttpURLConnection) newURL.openConnection();
            connection.setDoInput(true);
            connection.setDoOutput(true);

            // Output request is sent to the server, and a response is obtained
            DataOutputStream output = new DataOutputStream(connection.getOutputStream());
            output.writeBytes(request);
            InputStream input = connection.getInputStream();

            // Allows for the reading in of a request using BufferedReader
            BufferedReader in = new BufferedReader(new InputStreamReader(input));

            // Adds/ appends to the response line String when the line is not null
            String line;
            while ((line = in.readLine()) != null) {
                response.append(line);
            }

        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }

        // Returns the response using the StringBuffer
        return response;
    }
```



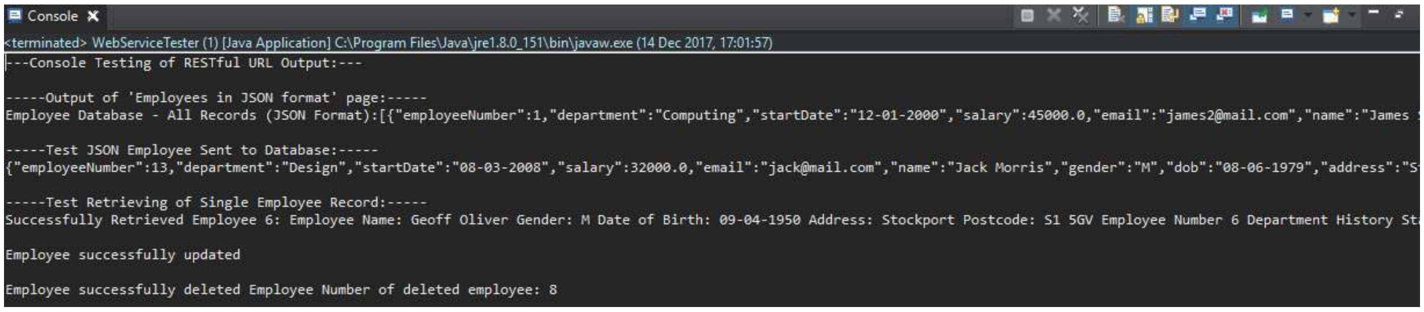
```java
    // Creates a StringBuffer for storing and returning the response for testing (the updating of an Employee's attributes based on Employee Number)
    // 'updateEmp' StringBuffer is called in the main method using test data, for testing
    private static StringBuffer updateEmp(String request, String url) {

        StringBuffer response = new StringBuffer();

        try {

            // Uses HttpURLConnection to connect to Http Server for testing the request
            URL newURL = new URL(url);
            HttpURLConnection connection = (HttpURLConnection) newURL.openConnection();
            connection.setDoInput(true);
            connection.setDoOutput(true);

            // Output request is sent to the server, and a response is obtained
            DataOutputStream output = new DataOutputStream(connection.getOutputStream());
            output.writeBytes(request);
            InputStream input = connection.getInputStream();

            // Allows for the reading in of a request using BufferedReader
            BufferedReader in = new BufferedReader(new InputStreamReader(input));

            // Adds/ appends to the response line String when the line is not null
            String line;
            while ((line = in.readLine()) != null) {
                response.append(line);
            }

        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }

        // Returns the response using the StringBuffer
        return response;
    }
```



```java
    // Creates a StringBuffer for storing and returning the response for testing (the deleting of an Employee with a given Employee Number)
    // 'deleteEmp' StringBuffer is called in the main method using an Employee Number, for testing
    private static StringBuffer deleteEmp(String request, String url) {

        StringBuffer response = new StringBuffer();

        try {

            // Uses HttpURLConnection to connect to Http Server for testing the request
            URL newURL = new URL(url);
            HttpURLConnection connection = (HttpURLConnection) newURL.openConnection();
            connection.setDoInput(true);
            connection.setDoOutput(true);

            // Output request is sent to the server, and a response is obtained
            DataOutputStream output = new DataOutputStream(connection.getOutputStream());
            output.writeBytes(request);
            InputStream input = connection.getInputStream();

            // Allows for the reading in of a request using BufferedReader
            BufferedReader in = new BufferedReader(new InputStreamReader(input));

            // Adds/ appends to the response line String when the line is not null
            String line;
            while ((line = in.readLine()) != null) {
                response.append(line);
            }

        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }

        // Returns the response using the StringBuffer
        return response;
    }
}
```

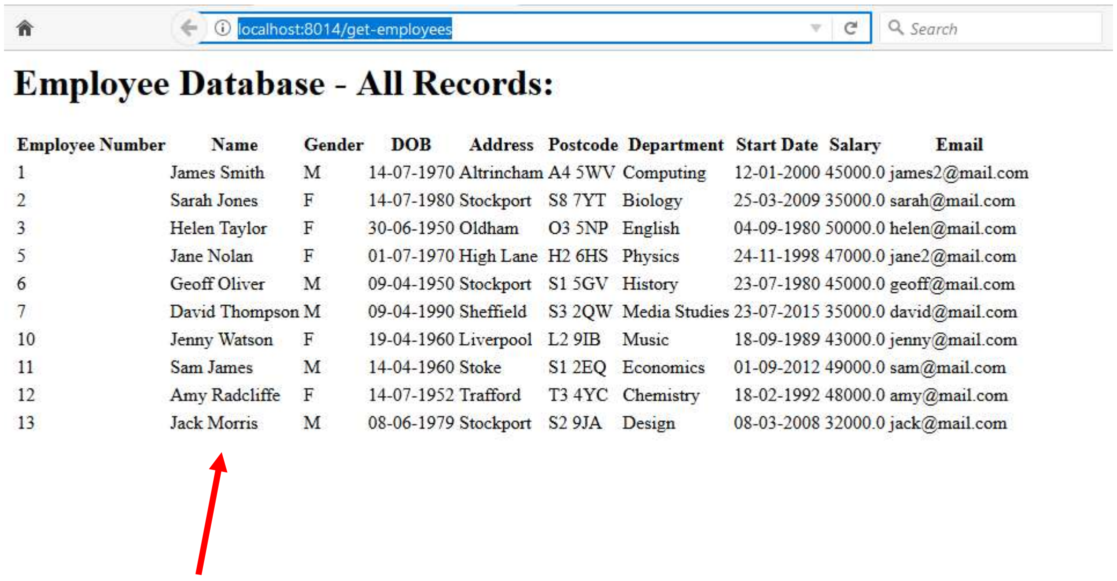
The WebServiceTester class includes code for testing the functionality of the server-side handler CRUD methods at the console. The class tests methods for the getting of all employees in JSON format, the posting of a JSON employee object, the retrieval of a single employee object, the updating of an employee record, and the deletion of an employee record. The class consists of private static methods for each operation. These methods use HttpURLConnection to connect to the server, and String Buffers for storing and returning the responses. These methods are called in a main method, using test data. Confirmation of the methods' successful testing is printed to the console screen. To test the JSON post, a new JSON employee record with an employee number of '13' is posted. Employee number 6 is retrieved, employee number 5's department, salary, email address, address and postcode is updated, and employee number 8 is deleted (see below screenshots for evidence that these operations were successful).

Console Output – WebServiceTester (CRUD operations actioned, and employee 6 retrieved)



```
Console
<terminated> WebServiceTester (1) [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (14 Dec 2017, 17:01:57)
---Console Testing of RESTful URL Output:---

-----Output of 'Employees in JSON format' page:-----
Employee Database - All Records (JSON Format):[{"employeeNumber":1,"department":"Computing","startDate":"12-01-2000","salary":45000.0,"email":"james2@mail.com","name":"James

-----Test JSON Employee Sent to Database:-----
{"employeeNumber":13,"department":"Design","startDate":"08-03-2008","salary":32000.0,"email":"jack@mail.com","name":"Jack Morris","gender":"M","dob":"08-06-1979","address":"S

-----Test Retrieving of Single Employee Record:-----
Successfully Retrieved Employee 6: Employee Name: Geoff Oliver Gender: M Date of Birth: 09-04-1950 Address: Stockport Postcode: S1 5GV Employee Number 6 Department History St

Employee successfully updated

Employee successfully deleted Employee Number of deleted employee: 8
```

Evidence of the New Employee Record (13, Jack Morris) in the Database:



localhost:8014/get-employees

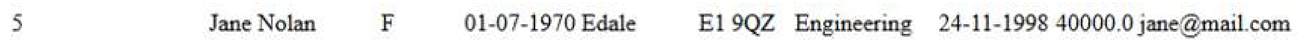
Search

Employee Database - All Records:

Employee Number	Name	Gender	DOB	Address	Postcode	Department	Start Date	Salary	Email
1	James Smith	M	14-07-1970	Altrincham	A4 5WV	Computing	12-01-2000	45000.0	james2@mail.com
2	Sarah Jones	F	14-07-1980	Stockport	S8 7YT	Biology	25-03-2009	35000.0	sarah@mail.com
3	Helen Taylor	F	30-06-1950	Oldham	O3 5NP	English	04-09-1980	50000.0	helen@mail.com
5	Jane Nolan	F	01-07-1970	High Lane	H2 6HS	Physics	24-11-1998	47000.0	jane2@mail.com
6	Geoff Oliver	M	09-04-1950	Stockport	S1 5GV	History	23-07-1980	45000.0	geoff@mail.com
7	David Thompson	M	09-04-1990	Sheffield	S3 2QW	Media Studies	23-07-2015	35000.0	david@mail.com
10	Jenny Watson	F	19-04-1960	Liverpool	L2 9IB	Music	18-09-1989	43000.0	jenny@mail.com
11	Sam James	M	14-04-1960	Stoke	S1 2EQ	Economics	01-09-2012	49000.0	sam@mail.com
12	Amy Radcliffe	F	14-07-1952	Trafford	T3 4YC	Chemistry	18-02-1992	48000.0	amy@mail.com
13	Jack Morris	M	08-06-1979	Stockport	S2 9JA	Design	08-03-2008	32000.0	jack@mail.com

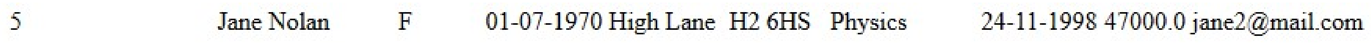
Evidence of the Updated Employee Record (5, Jane Nolan) in the Database:

Existing:



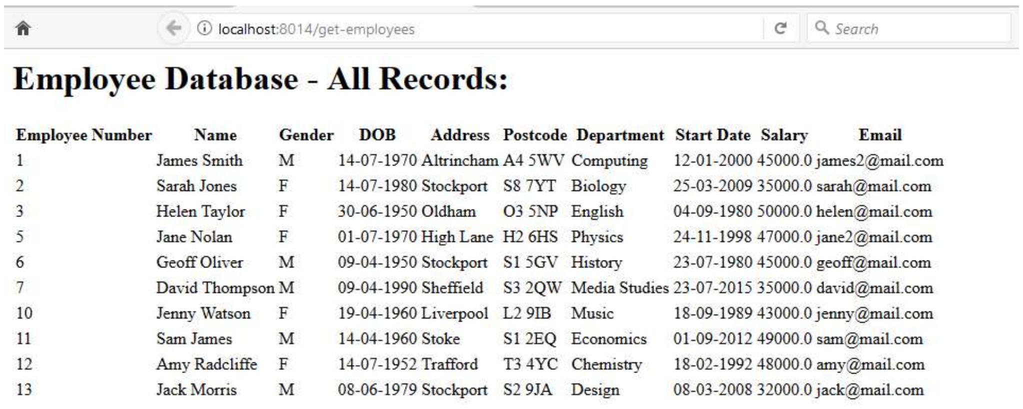
5	Jane Nolan	F	01-07-1970	Edale	E1 9QZ	Engineering	24-11-1998	40000.0	jane@mail.com

New (updated department, salary, email address, address and postcode):



5	Jane Nolan	F	01-07-1970	High Lane	H2 6HS	Physics	24-11-1998	47000.0	jane2@mail.com

Evidence of the Deletion of Employee Record (8) from the Database:



localhost:8014/get-employees

Employee Database - All Records:

Employee Number	Name	Gender	DOB	Address	Postcode	Department	Start Date	Salary	Email
1	James Smith	M	14-07-1970	Altrincham	A4 5WV	Computing	12-01-2000	45000.0	james2@mail.com
2	Sarah Jones	F	14-07-1980	Stockport	S8 7YT	Biology	25-03-2009	35000.0	sarah@mail.com
3	Helen Taylor	F	30-06-1950	Oldham	O3 5NP	English	04-09-1980	50000.0	helen@mail.com
5	Jane Nolan	F	01-07-1970	High Lane	H2 6HS	Physics	24-11-1998	47000.0	jane2@mail.com
6	Geoff Oliver	M	09-04-1950	Stockport	S1 5GV	History	23-07-1980	45000.0	geoff@mail.com
7	David Thompson	M	09-04-1990	Sheffield	S3 2QW	Media Studies	23-07-2015	35000.0	david@mail.com
10	Jenny Watson	F	19-04-1960	Liverpool	L2 9IB	Music	18-09-1989	43000.0	jenny@mail.com
11	Sam James	M	14-04-1960	Stoke	S1 2EQ	Economics	01-09-2012	49000.0	sam@mail.com
12	Amy Radcliffe	F	14-07-1952	Trafford	T3 4YC	Chemistry	18-02-1992	48000.0	amy@mail.com
13	Jack Morris	M	08-06-1979	Stockport	S2 9JA	Design	08-03-2008	32000.0	jack@mail.com

As you can see, an employee record with an employee number of ‘8’ no longer exists in the database.

How could this system be improved?

The above screenshots and accompanied source code illustrate the successful creation of a Java RESTful web service, which can manipulate database data in both offline and online server-side contexts. However, it is important to note that there is room for improvement. Security is a hugely important issue in web application development. However, due to time constraints, this was not implemented. Also, data validation, which would have assured that all data entered was error-free, was also not added. Taking time to implement the advanced features of the assignment, which include these features, will address these issues.