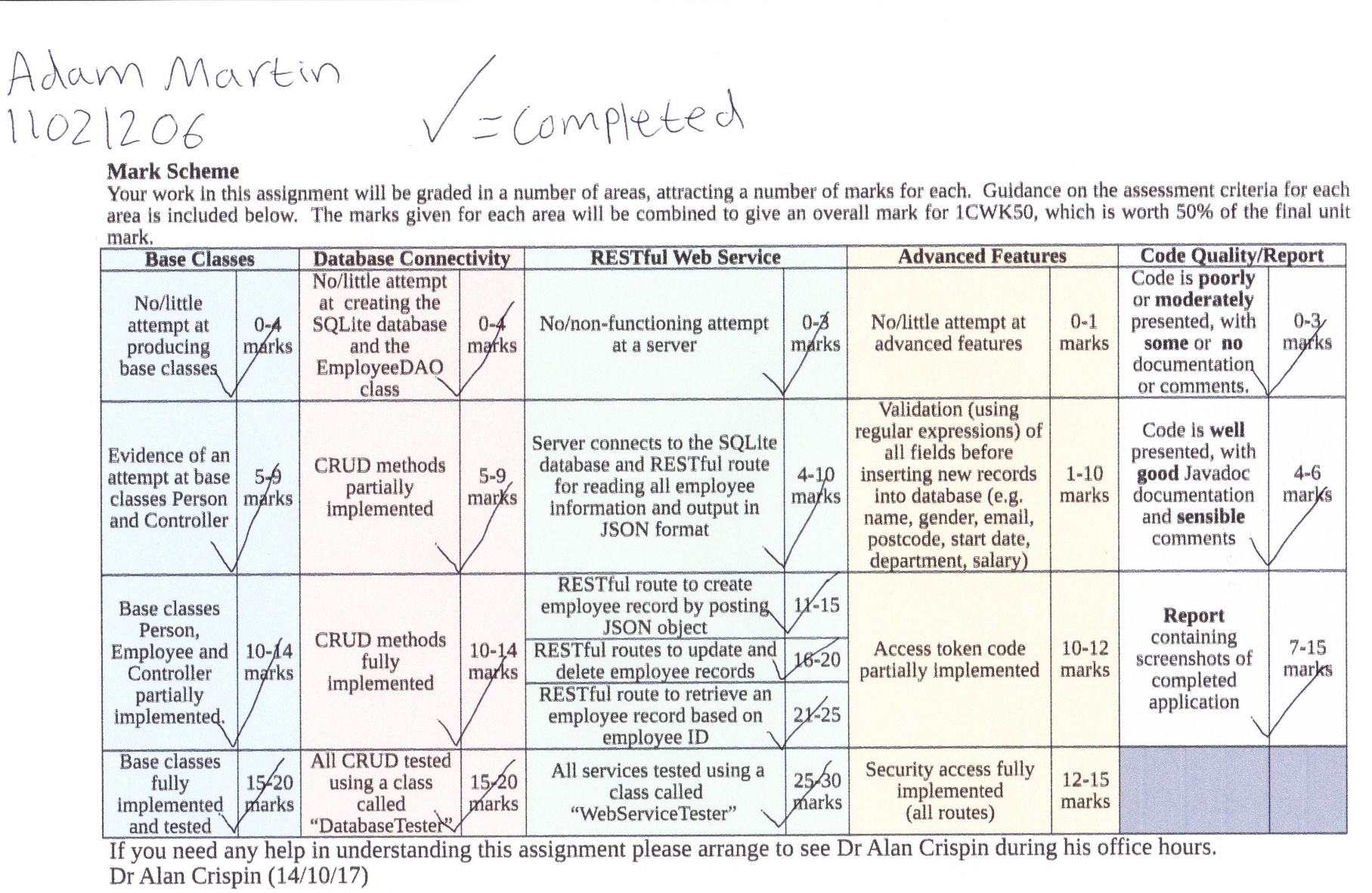
**Name:** Adam Martin

**Student Number:** 11021206

**Module:** Object Oriented and Web Programming

**Assignment:** Report - Java Employee RESTful Web Service Coursework

**Annotated Marking Grid:**

****

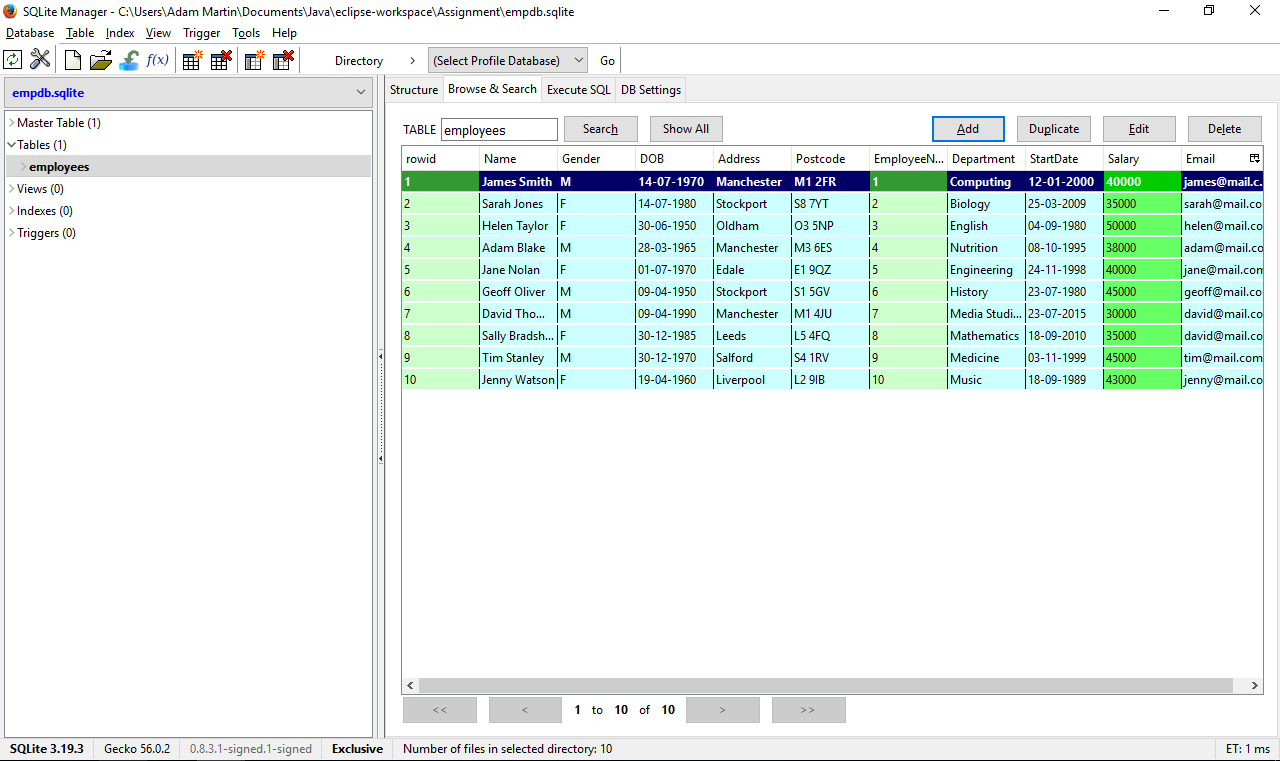
**Report – Java Employee RESTful Web Service Coursework**

**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.

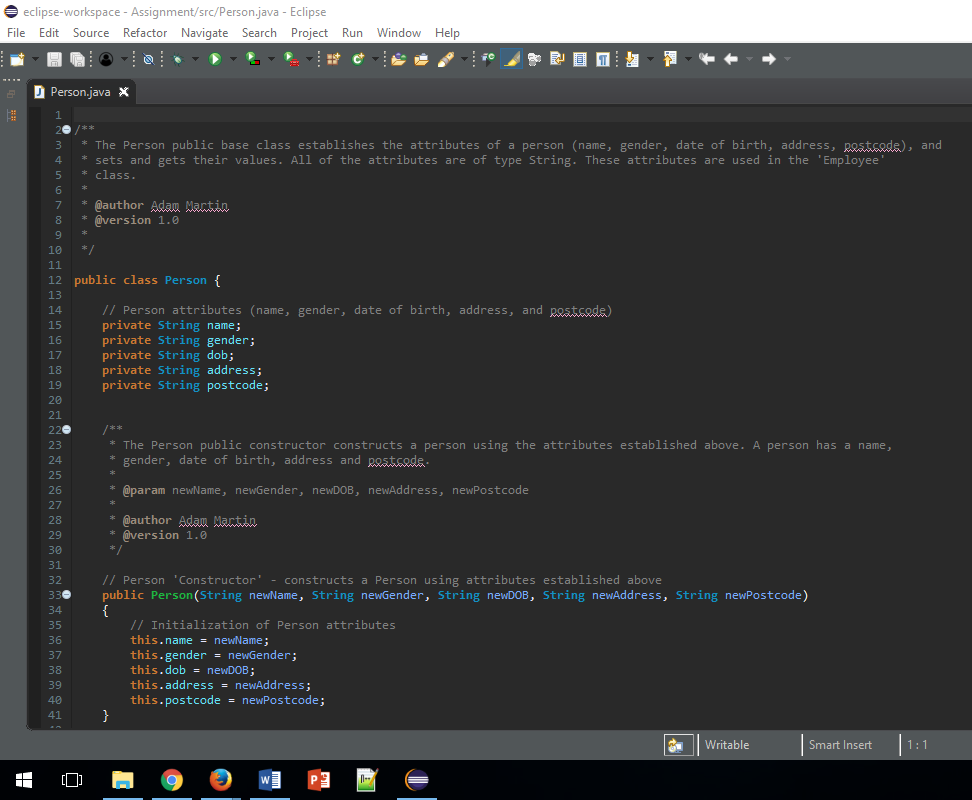
**Functionality, and Steps Taken to Achieve Functionality:**

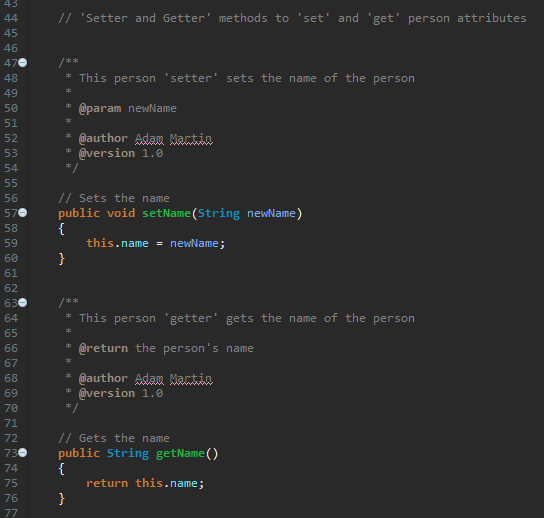
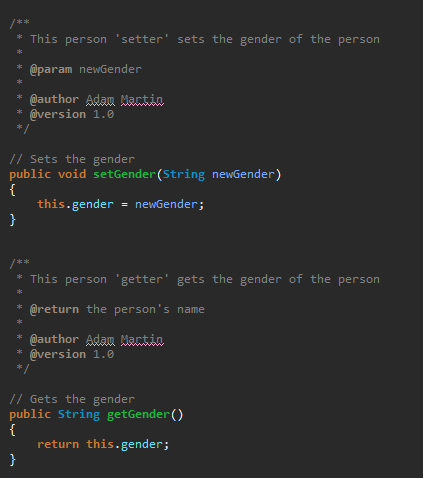
**Step 1 – Creation of the ‘Employees’ SQLite Database:**

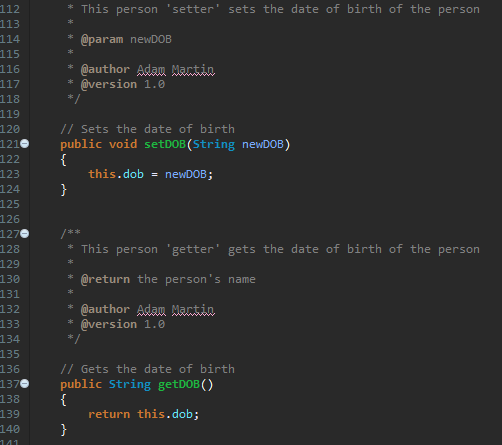
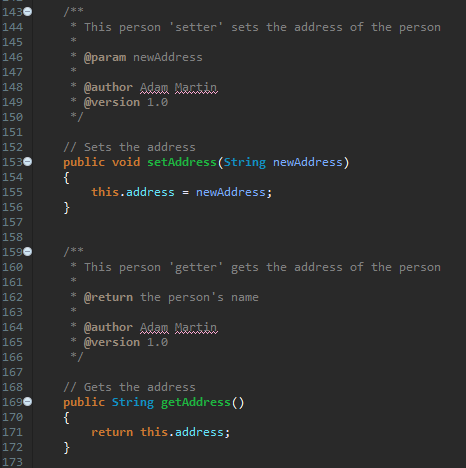


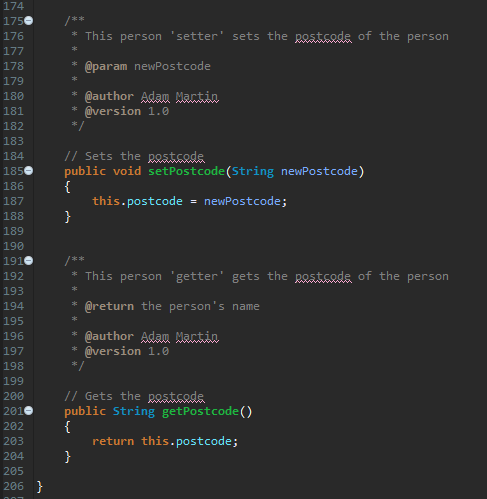
The first step was to create the database, using the Firefox ‘SQLite Manager’ tool. The database contains a single ‘Employees’ table, populated with 10 fictitious records. In this scenario, the fictitious people are employees of a local university.

**Step 2 – ‘Person’ Class:**



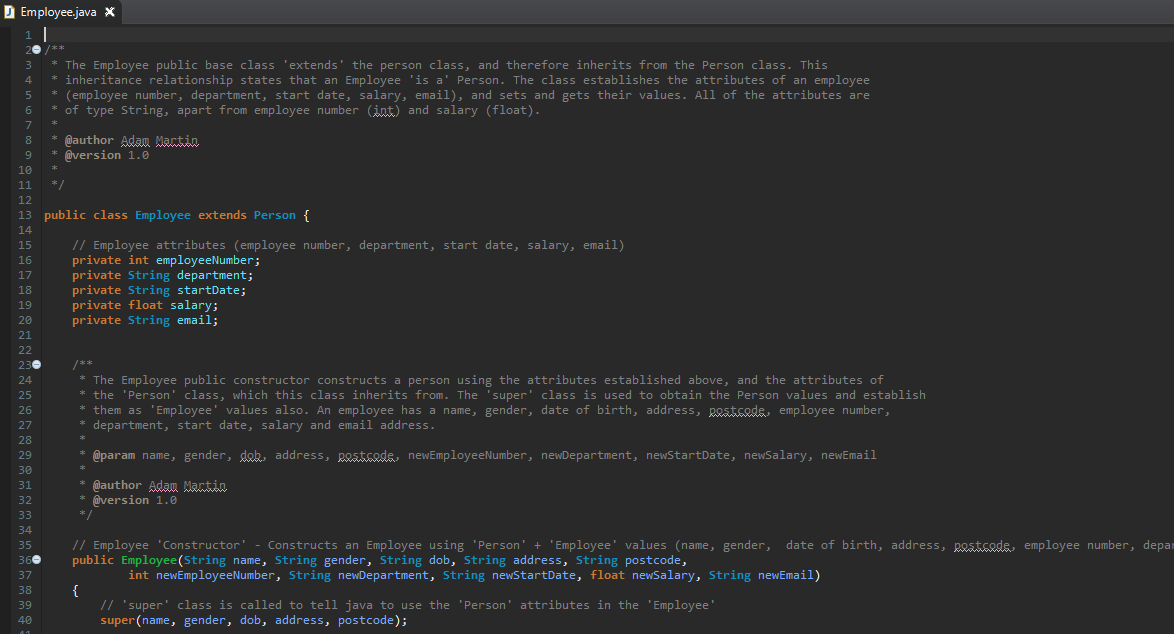
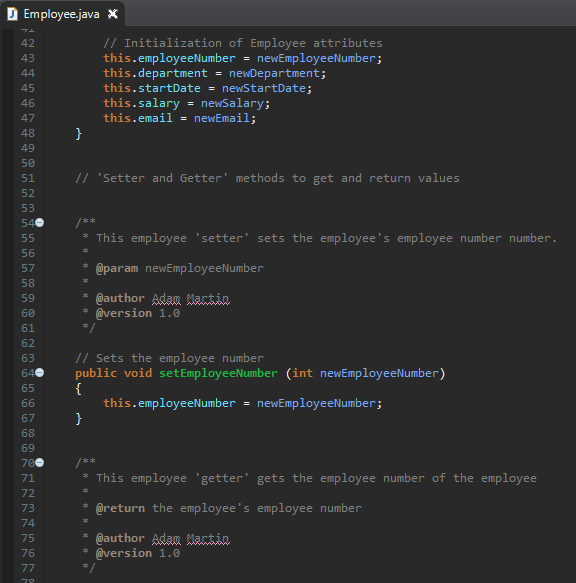
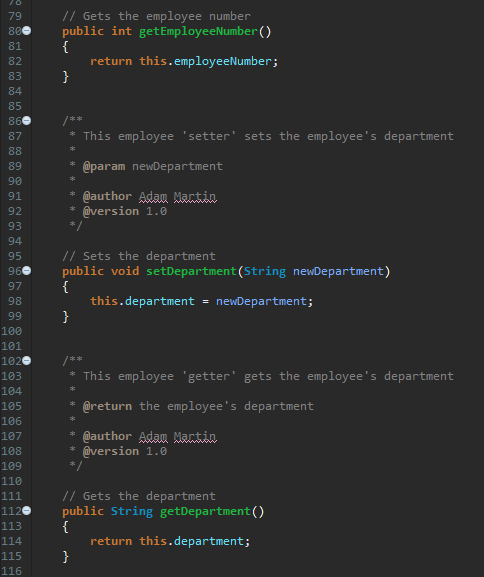


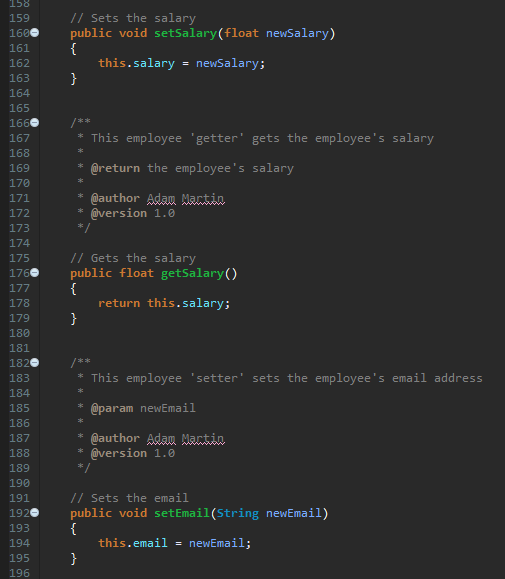


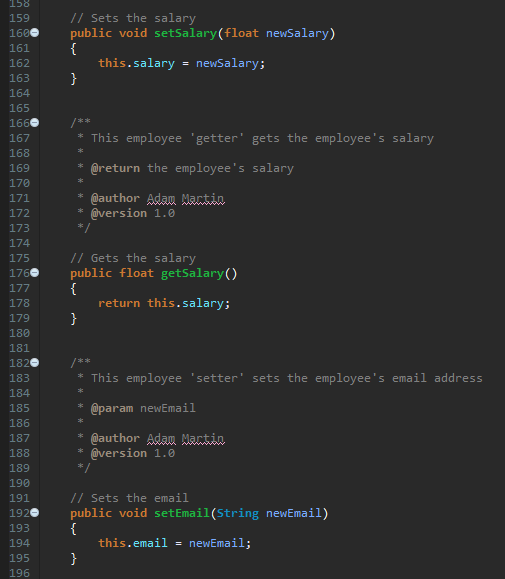
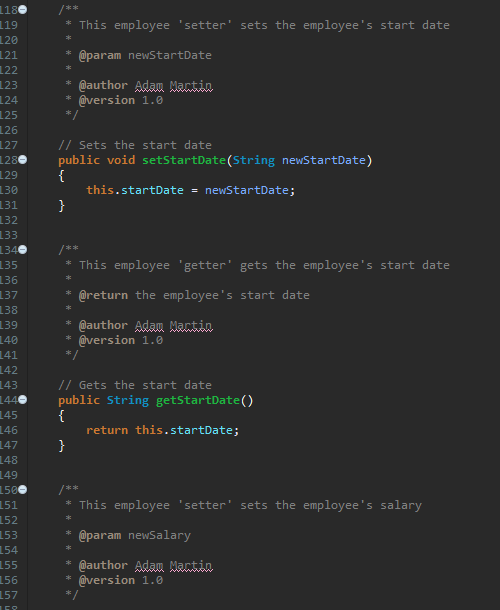
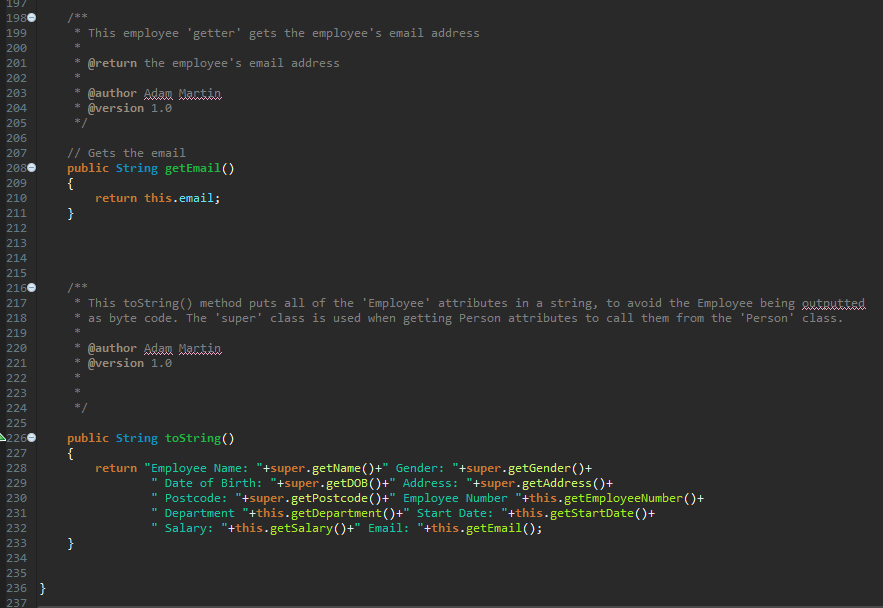


The person base class sets the attributes of a person (name, gender, date of birth, address, postcode), which are all of type string. The class also implements ‘setter’ and ‘getter’ methods to set and get the person’s values. This class underpins the ‘Employee’ class, which inherits its values.

**Step 3 – ‘Employee’ Class:**

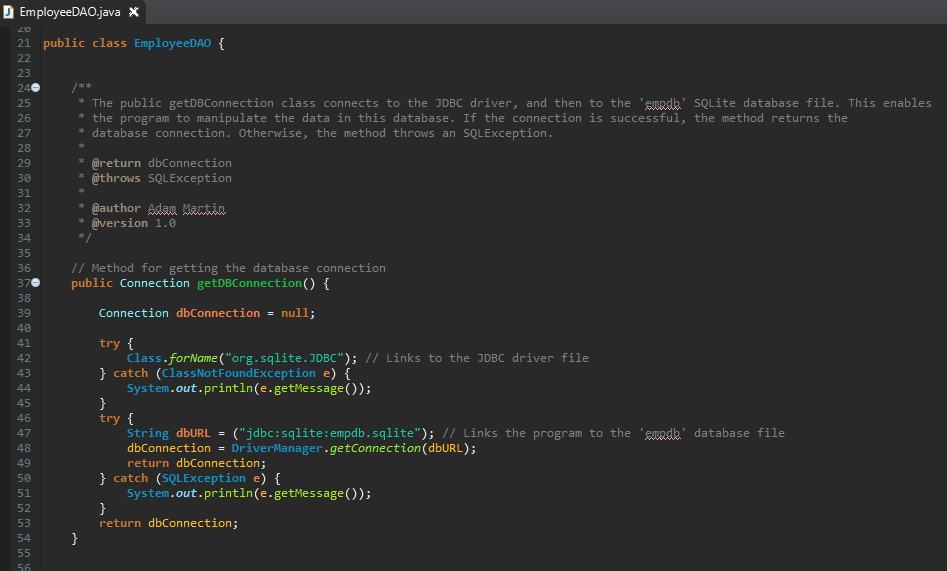


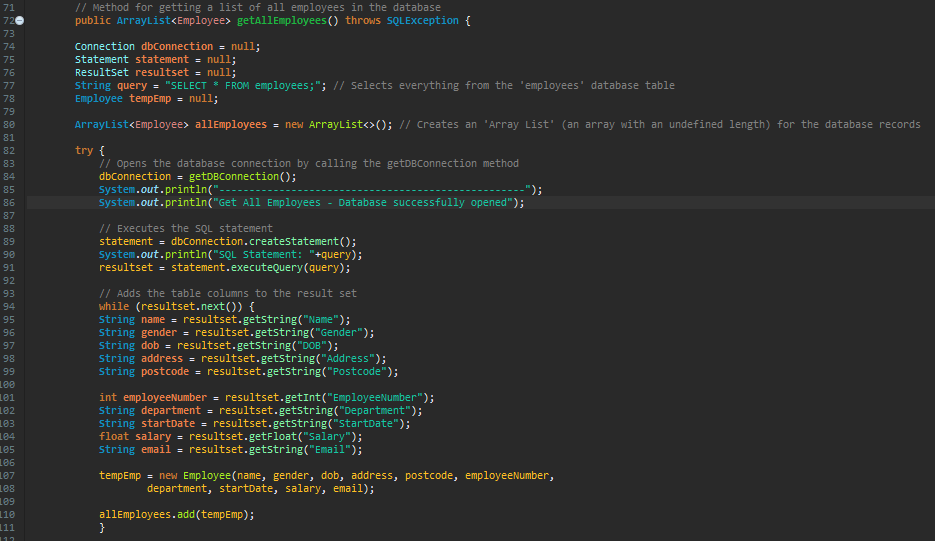


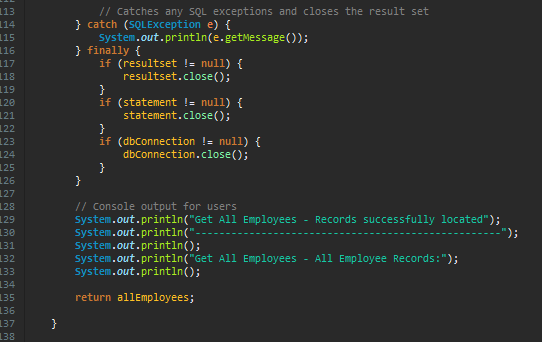


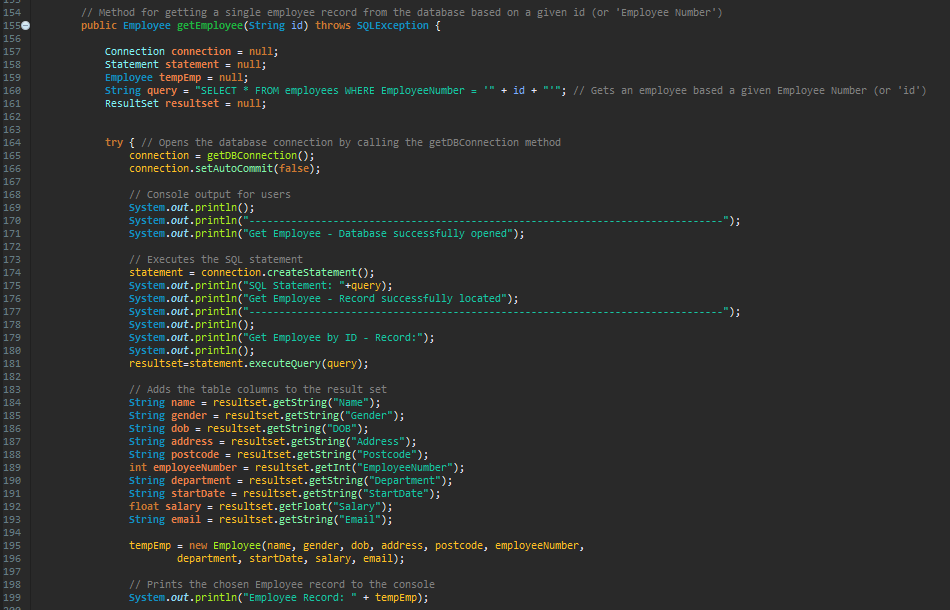
The employee base class ‘extends’ the person class, to inherit its attributes. The class contains setter and getter methods for employee-specific attributes (employee number, department, start date, salary email). These attributes are placed in a toString method, in order to print them to the console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

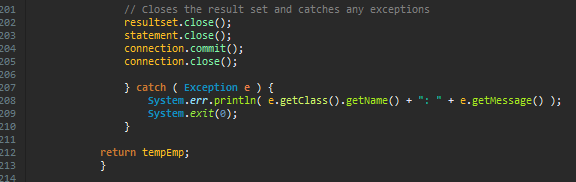
**Step 3 – ‘EmployeeDAO’ Class:**

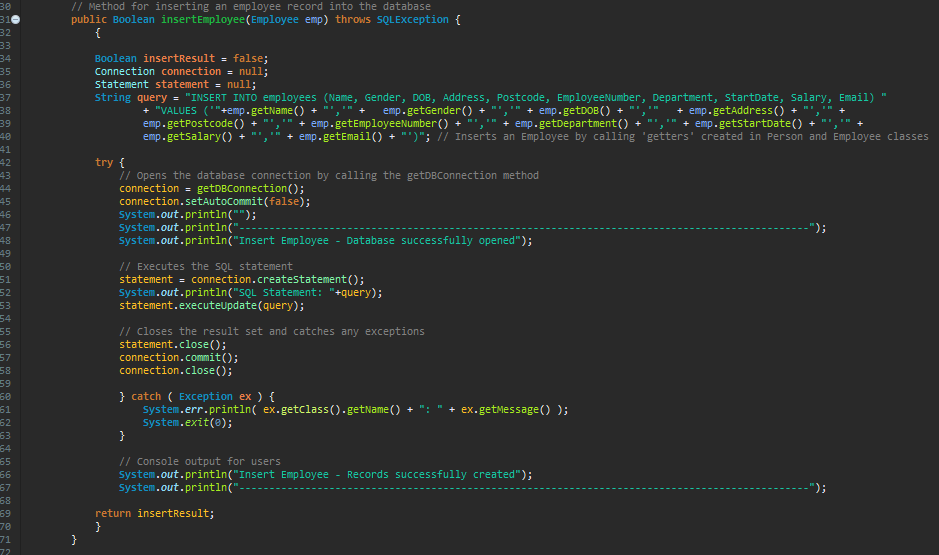


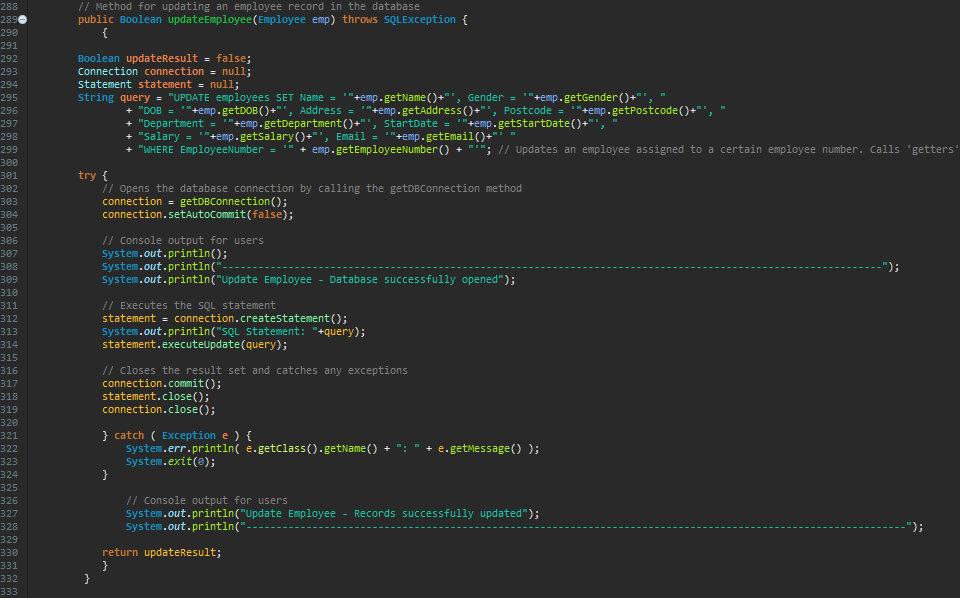


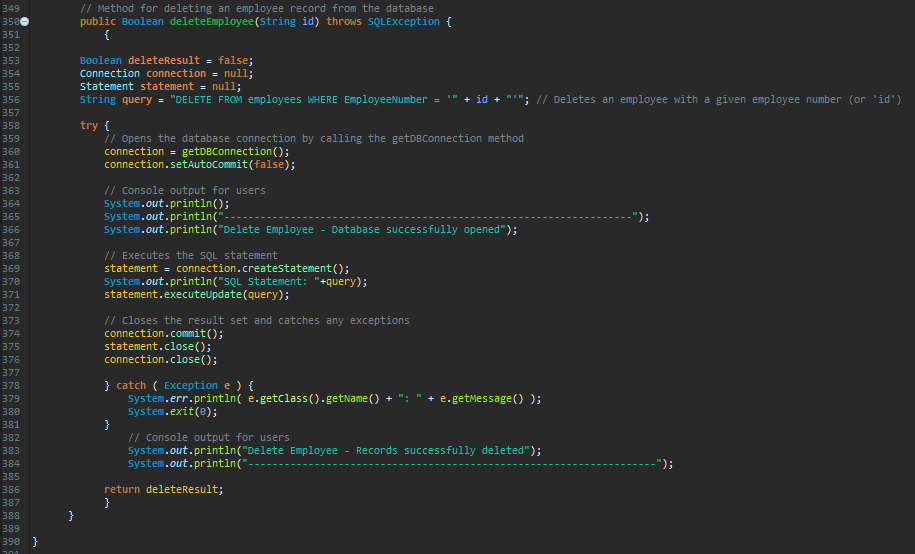






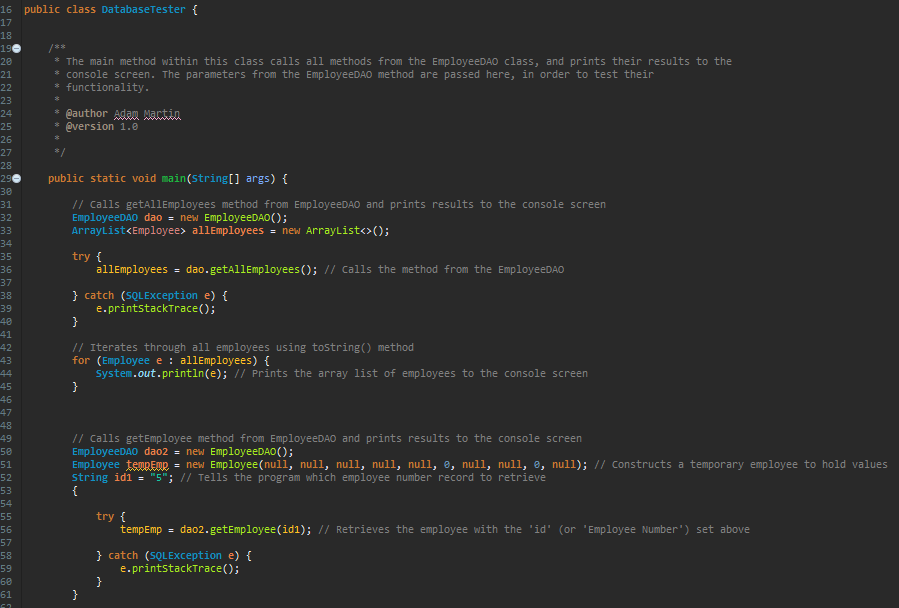


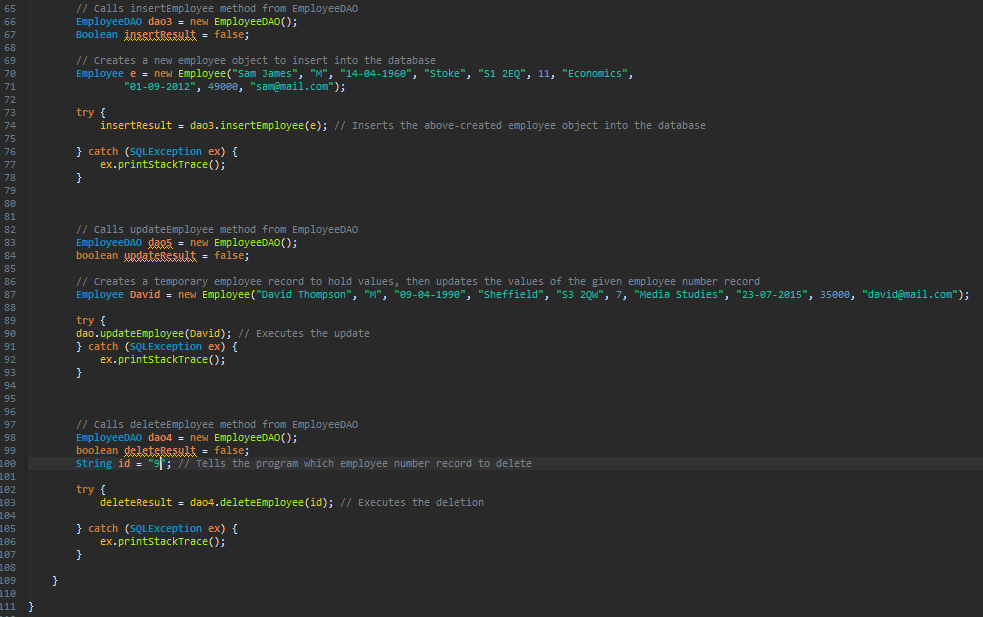




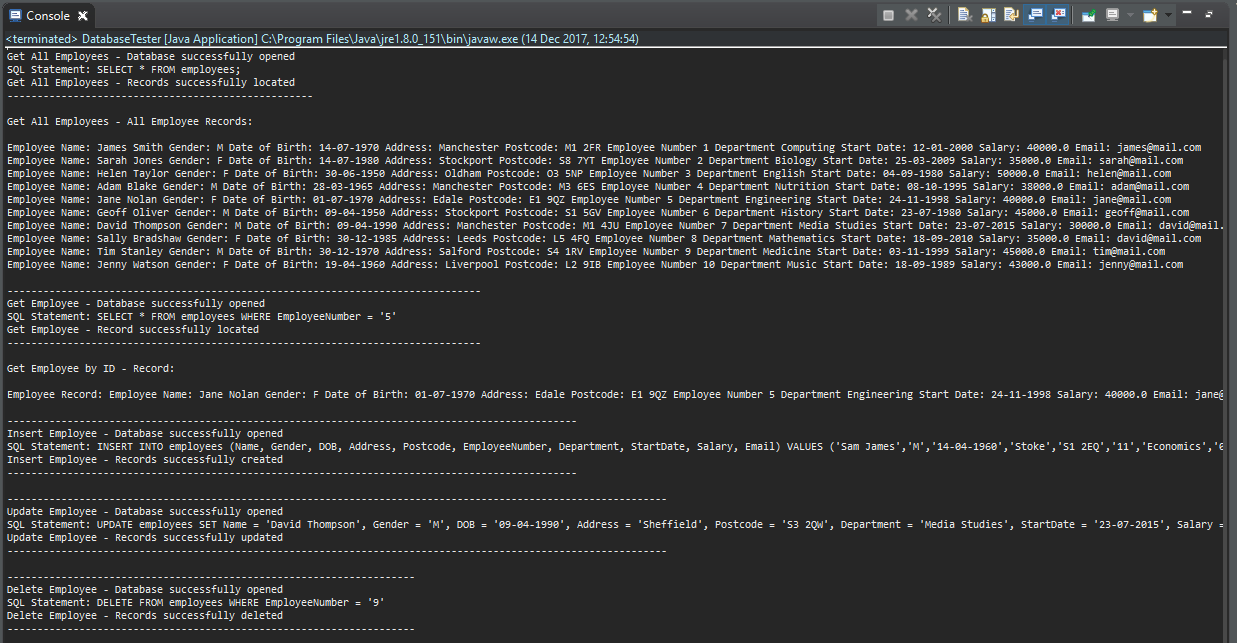
The EmployeeDAO class contains a method for connecting to the database, and the CRUD methods for getting all employees, getting a single employee, inserting an employee, updating an employee and deleting an employee. The CRUD operations are actioned using a range of SQL statements. This class is crucial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

**Step 4 – ‘DatabaseTester’ Class:**

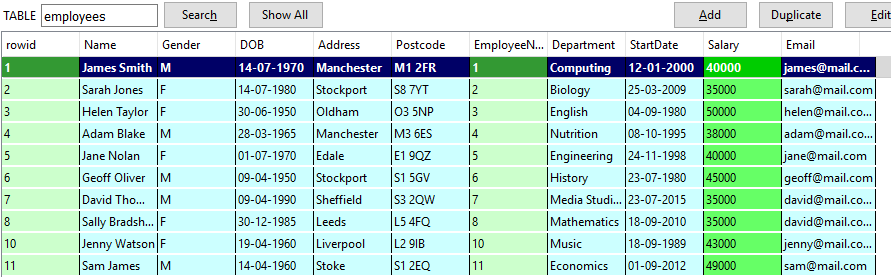




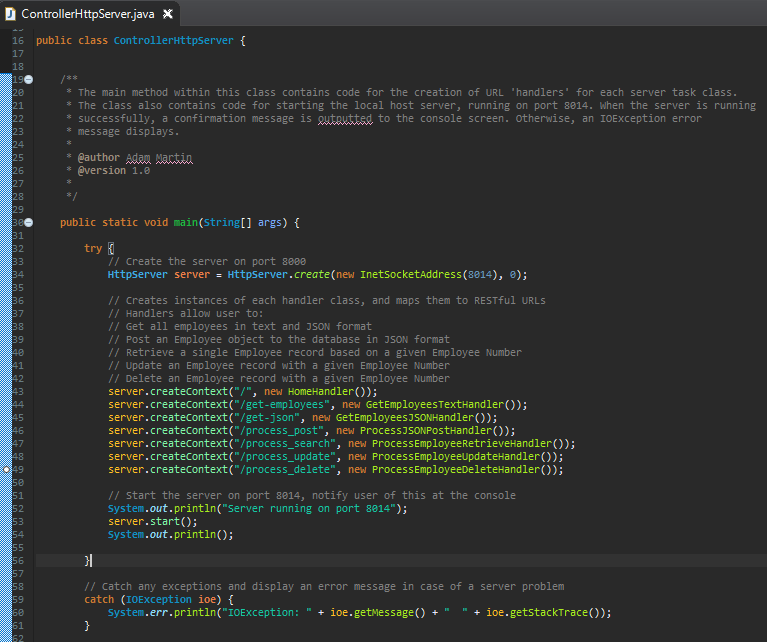
The DatabaseTester class calls the CRUD methods from the EmployeeDAO class, by passing in test data. The results of the test are printed to the console screen.

**DatabaseTester – Console Output:**

**DatabaseTester – Updated ‘Employees’ Database Table:**

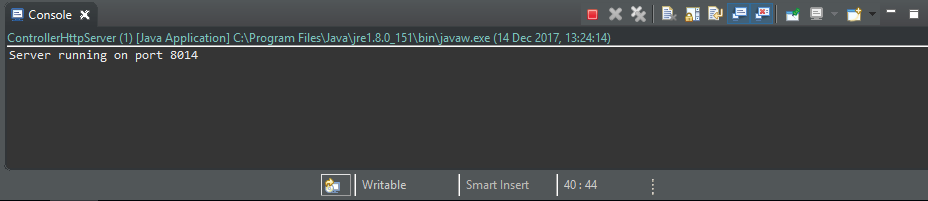


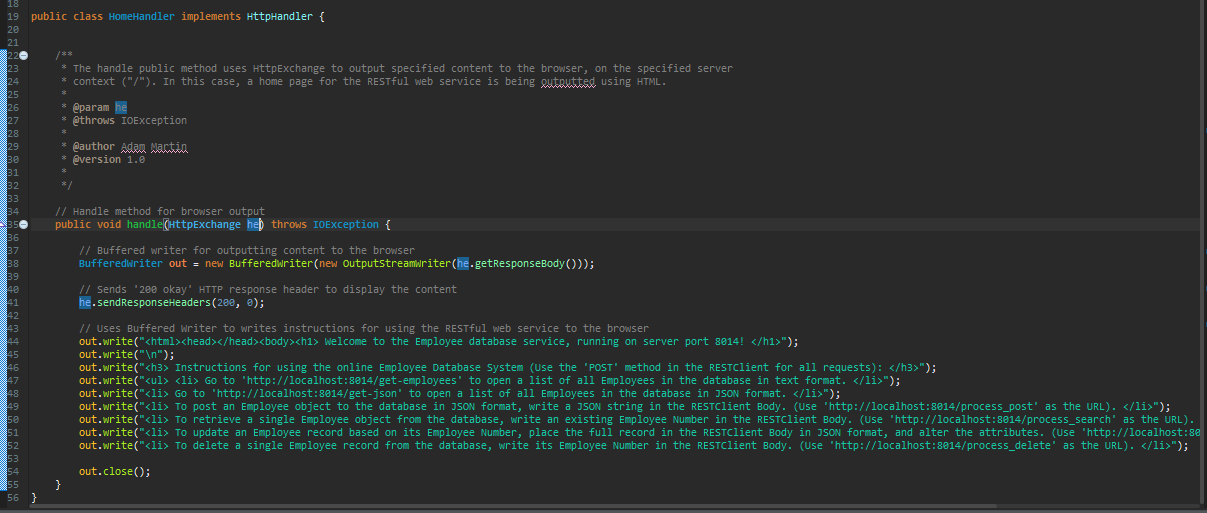
This screenshot depicts the ‘employees’ SQLite database table, after the DatabaseTester class was run in Eclipse. As you can see, all of the operations worked successfully. All employee records were printed to the console, and a single employee, with an employee number of ‘5’ was also printed to the 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:**

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).

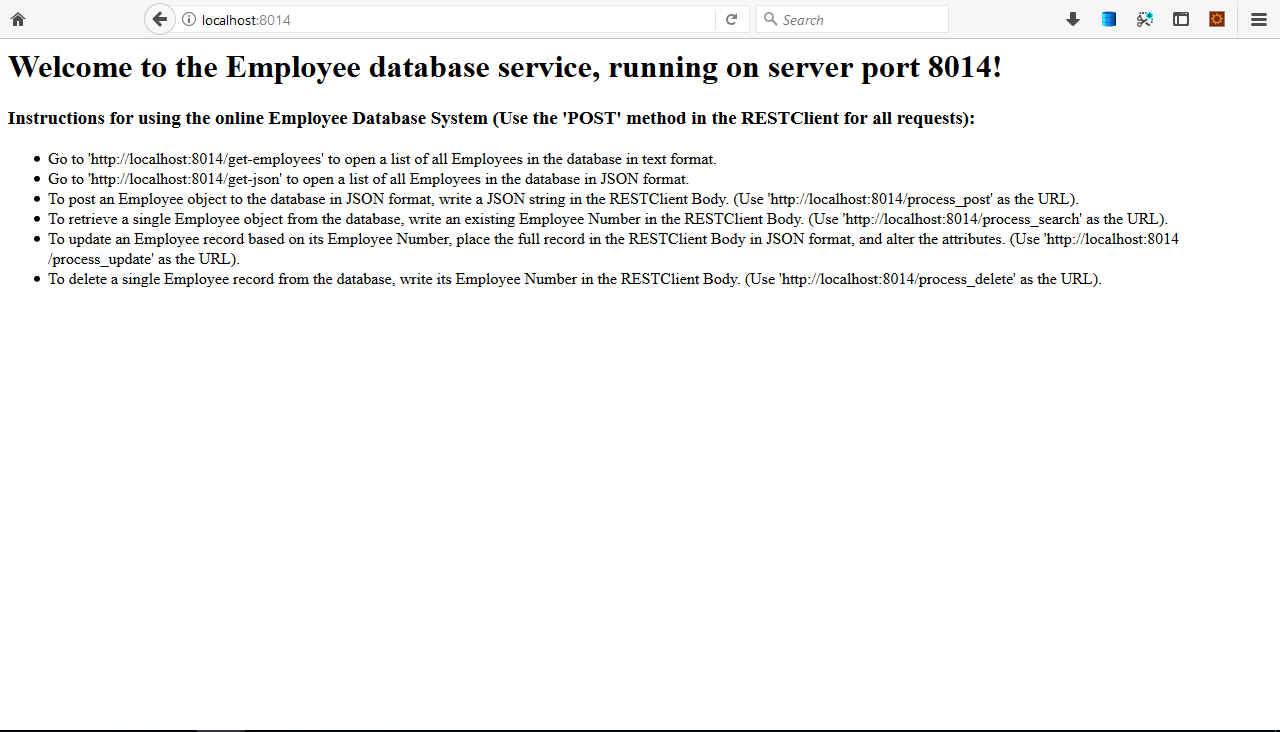
**Console Output – ControllerHttpServer:**



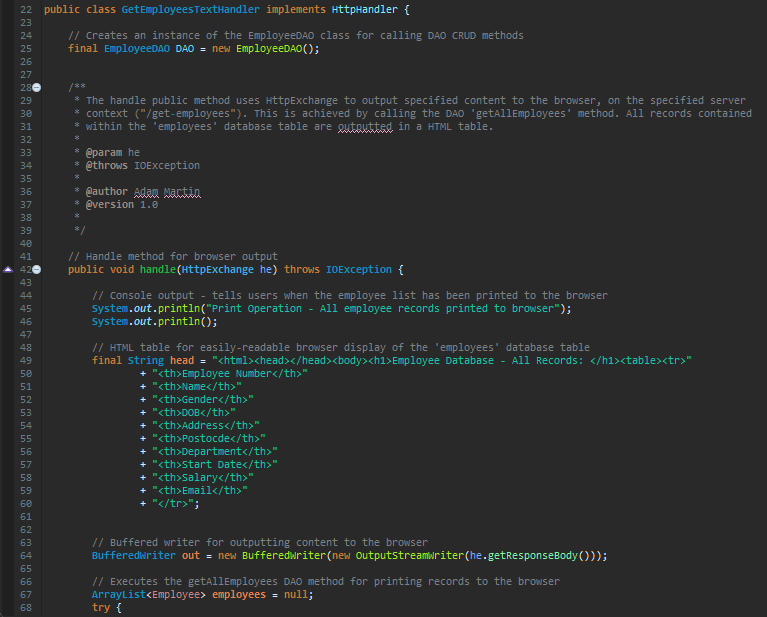
**Step 6 – ‘Home Handler’ Class:**

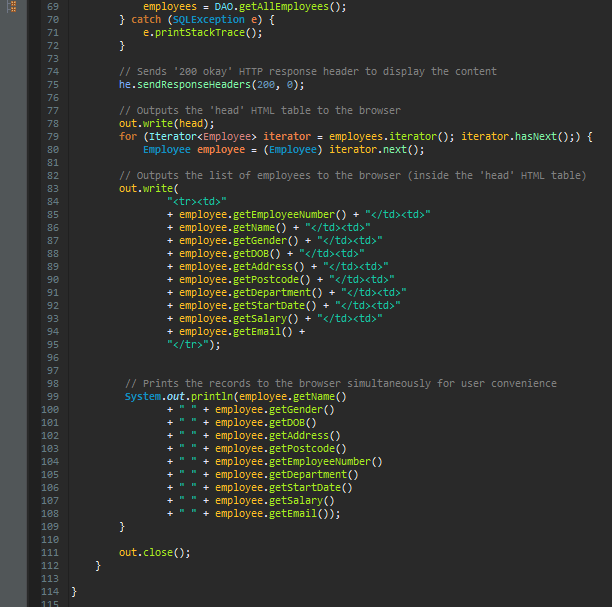
The HomeHandler class includes code for outputting a system homepage/ instruction page to the user, with instructions displayed using embedded HTML code. The ControllerHttpServer class allows this content to display by creating an instance of the HomeHandler class, and placing it at the “/” server context.

**Browser Output – HomeHandler:**



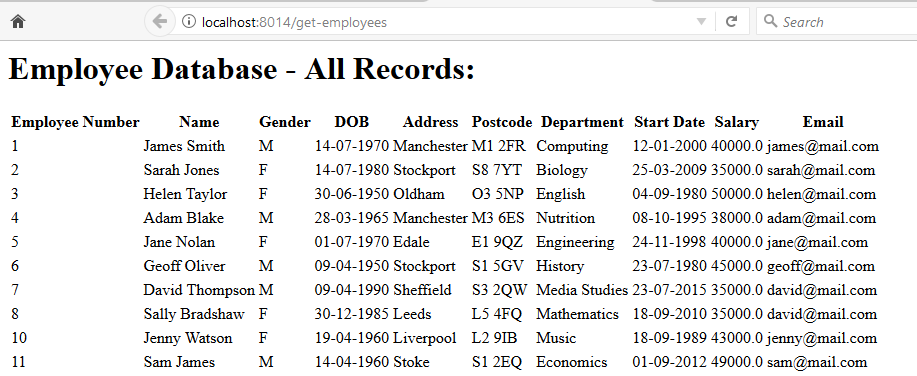
**Step 7 – ‘GetEmployeesTextHandler’ Class:**





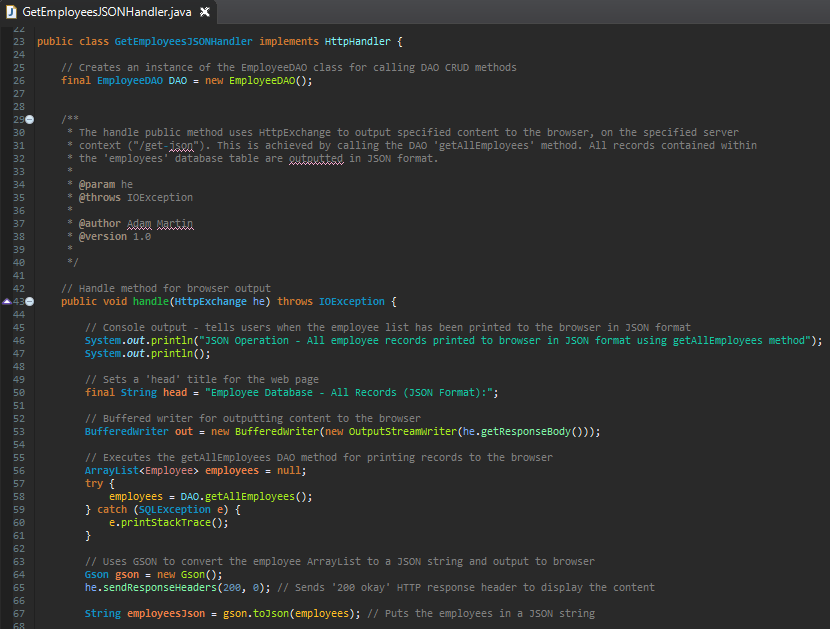
The GetEmployeesTextHandler class includes code for outputting HTML table of all employees to the user. This code involved calling the GetAllEmployees method from the EmployeeDAO class. The ControllerHttpServer class allows this content to display by creating an instance of the GetEmployeesTextHandler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

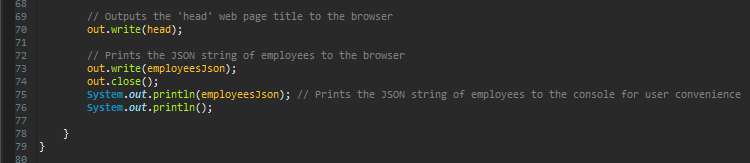
**Browser Output – GetEmployeesTextHandler:**



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.

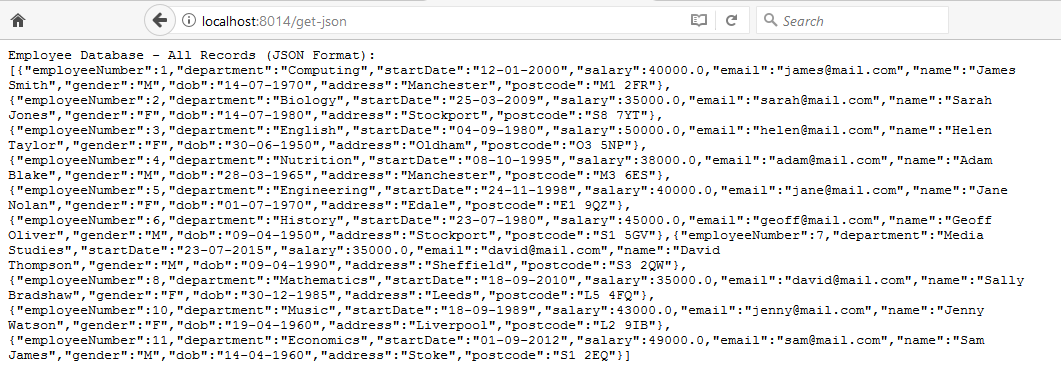
**Step 8 – ‘GetEmployeesJSONHandler’ Class:**

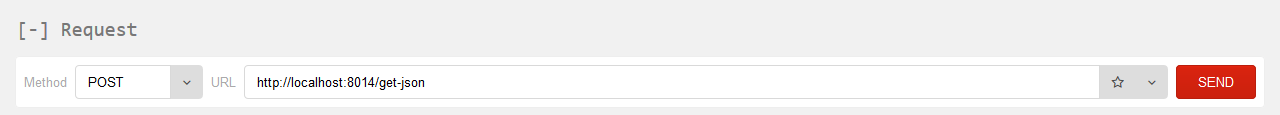


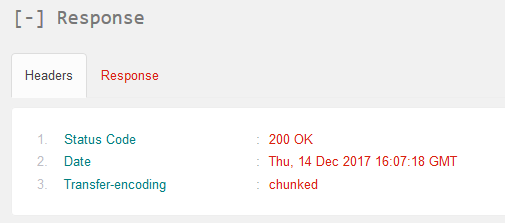


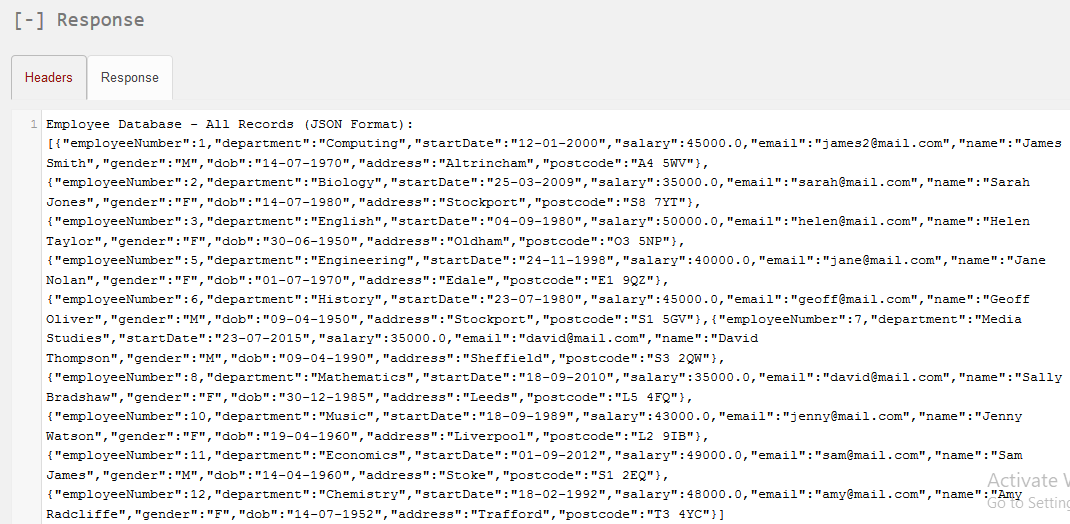
The GetEmployeesJSONHandler class includes code for outputting all employee records to the user in JSON (JavaScript Object Notation) format. This code involved calling the GetAllEmployees method from the EmployeeDAO class, converting a String of these employees ‘toJson’, and outputting it to the user. 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:**



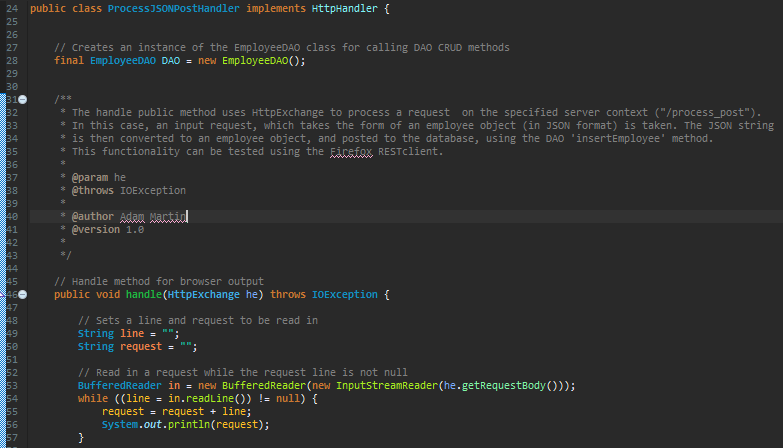
**Testing of ‘GetEmployeesJSON, Using the Firefox RESTclient:**

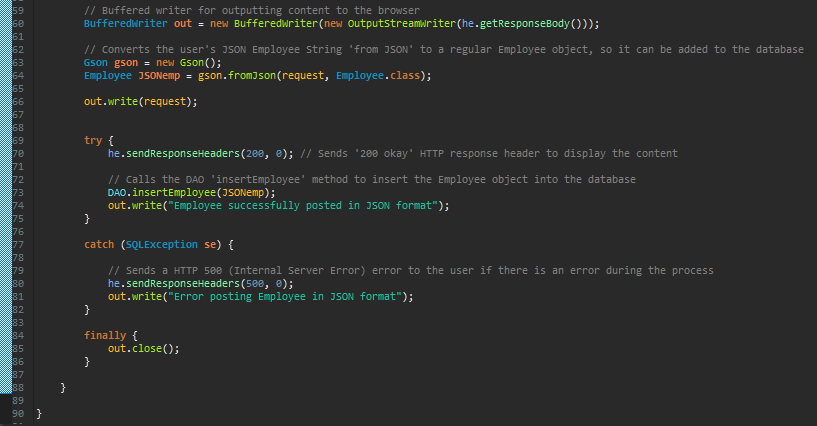




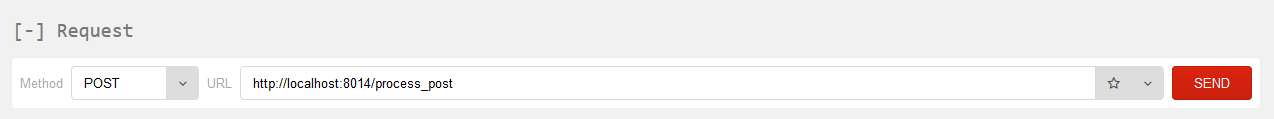
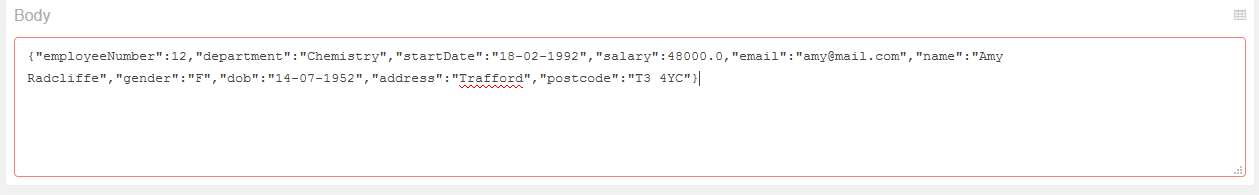
These screenshots depict the successful testing of the retrieval of all employees in JSON format. As you can see, the JSON employee records were successfully received, and a positive ‘200 OK’ response header was returned.

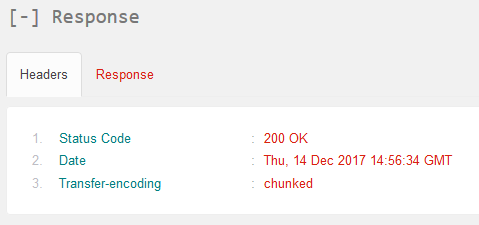
**Step 9 – ‘ProcessJSONPostHandler’ Class:**

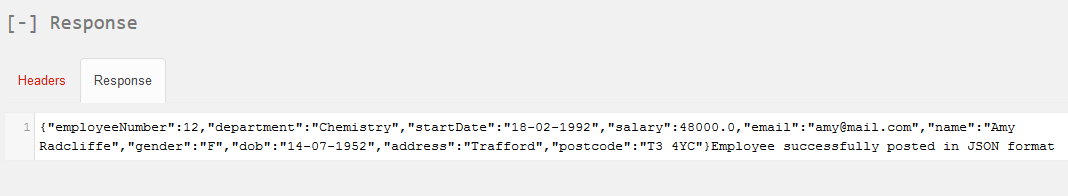




The ProcessJSONPostHandler class includes code for allowing the user to post an Employee record to the database, in JSON format. This code takes in a request, which takes the form of a JSON Employee string. The JSON string request is then converted to a regular employee object, and added to the database 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:**



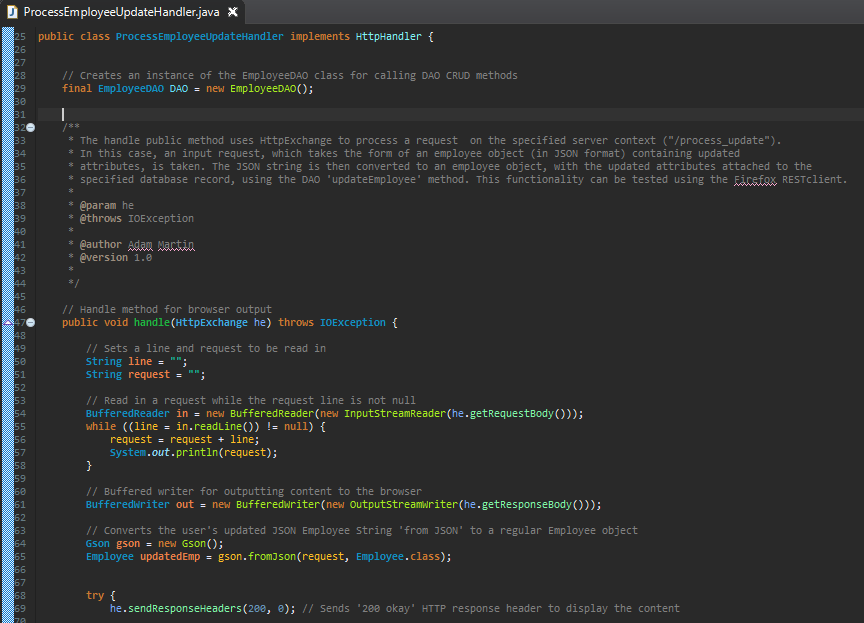


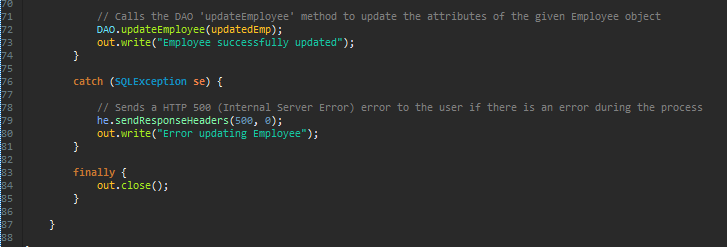
These screenshots depict the successful testing of the JSON post. As you can see, the employee was successfully posted, and a positive ‘200 OK’ response header was returned.

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

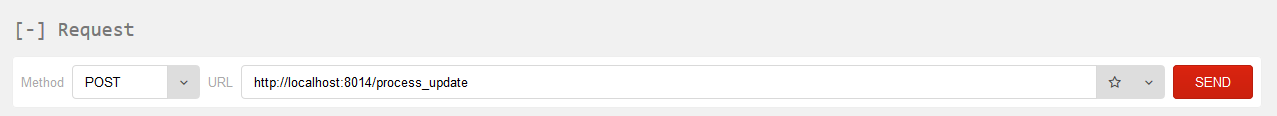
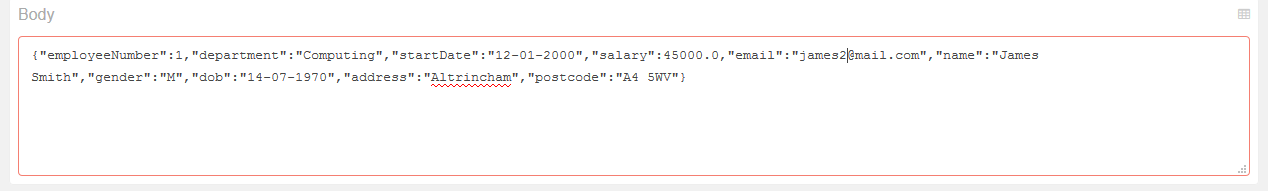


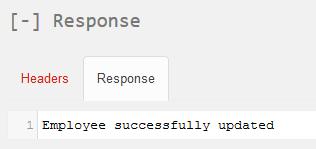
**Step 10 – ‘ProcessEmployeeUpdateHandler’ Class:**





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:**



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:**

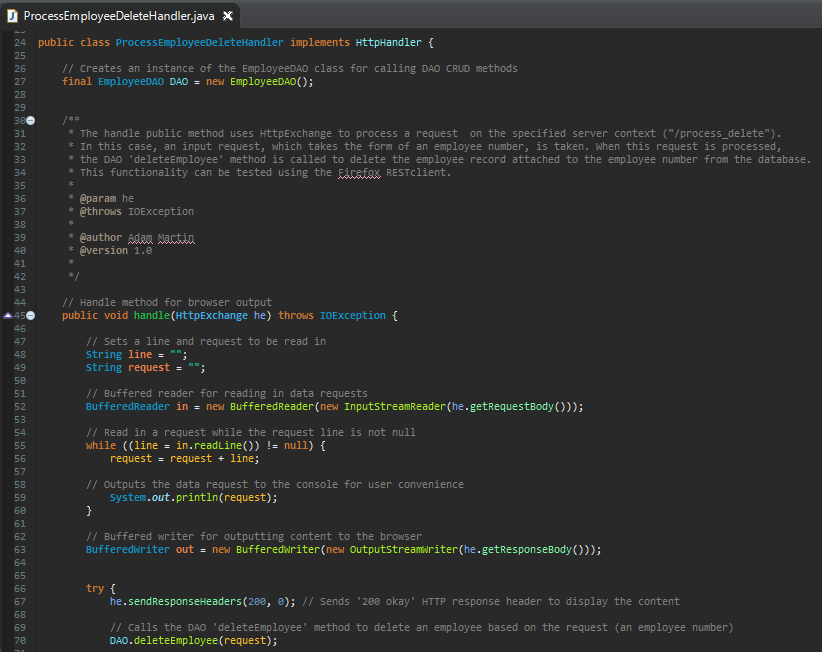
**Existing:**

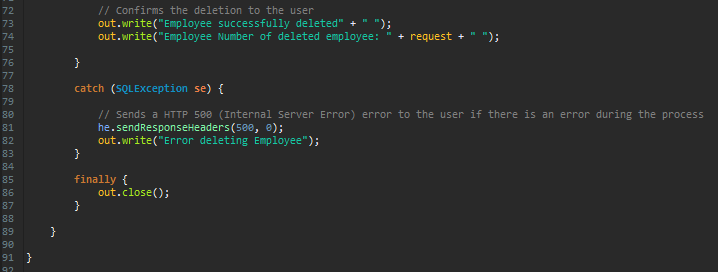


**New (updated salary, email address, address and postcode):**

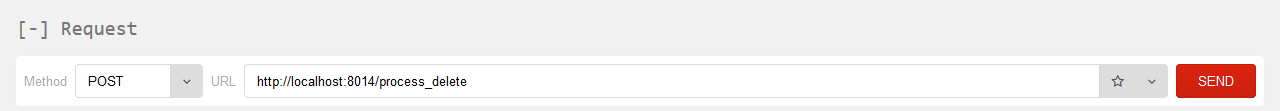


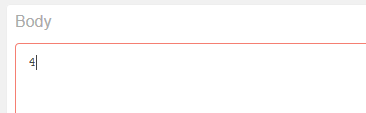
**Step 11 – ‘ProcessEmployeeDeleteHandler’ Class:**

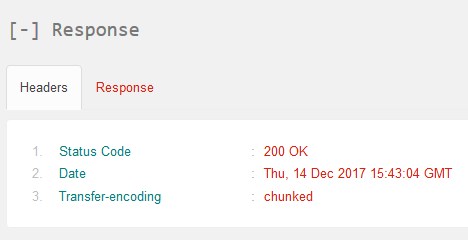


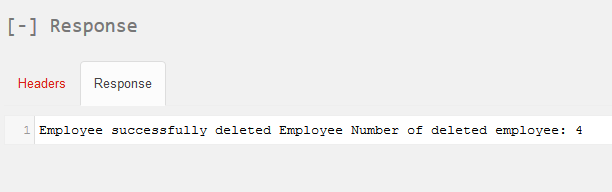


The ProcessEmployeeDeleteHandler class includes code for allowing the user to delete an employee record with a specified employee number. This code takes in a request, which takes the form of an existing employee number. The employee record attached to that employee number is then deleted, , via the 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:**

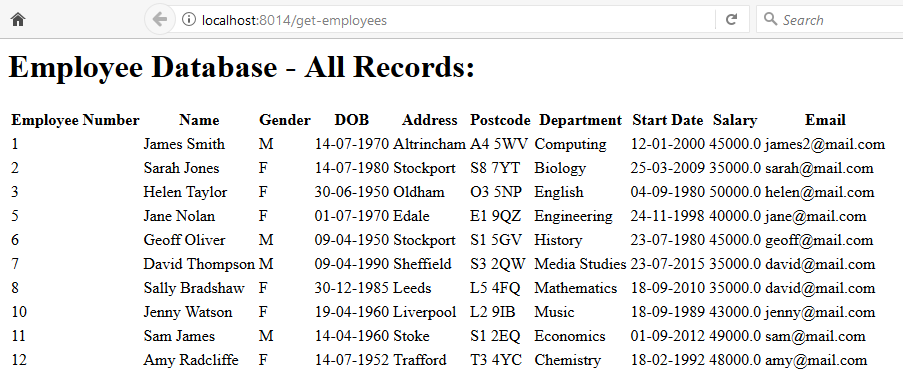






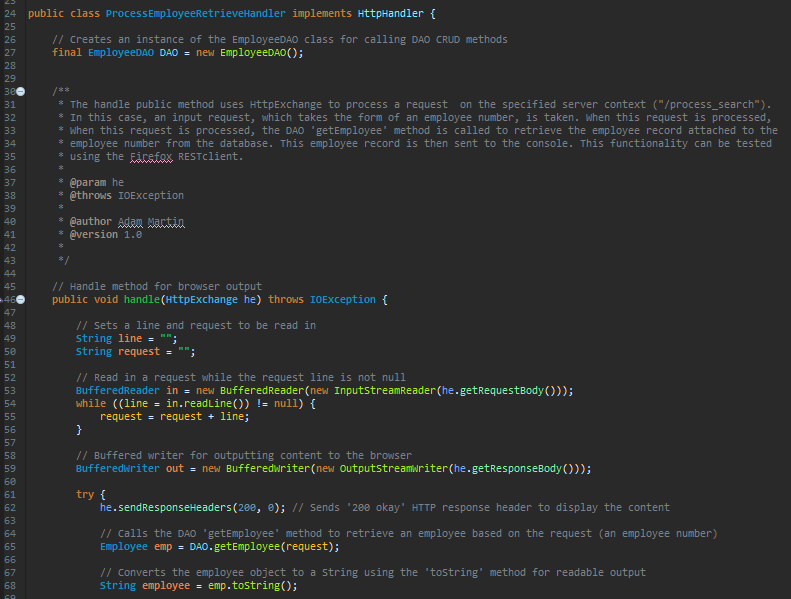
These screenshots depict the successful testing of the employee delete. A request to delete the employee record with an employee number of ‘4’ was sent. As you can see, the employee was successfully deleted, and a positive ‘200 OK’ response header was returned.

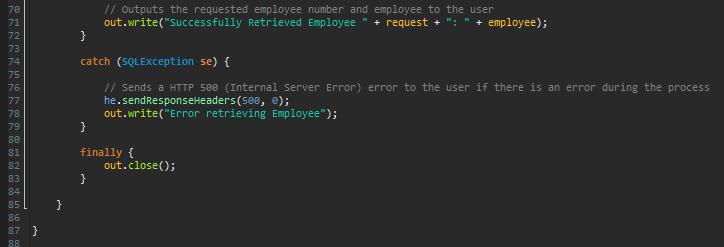
**Evidence of the Deletion of Employee Record (4) from the Database:**



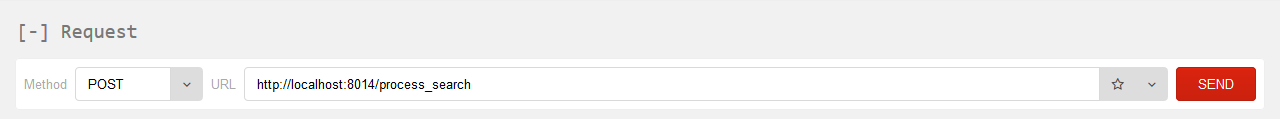
As you can see, an employee record with an employee number of ‘4’ no longer exists in the database.

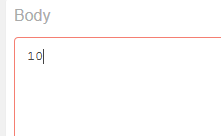
**Step 12 – ‘ProcessEmployeeRetrieveHandler’ Class:**

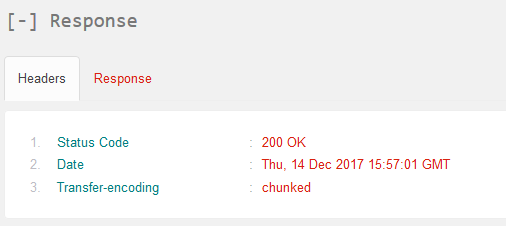


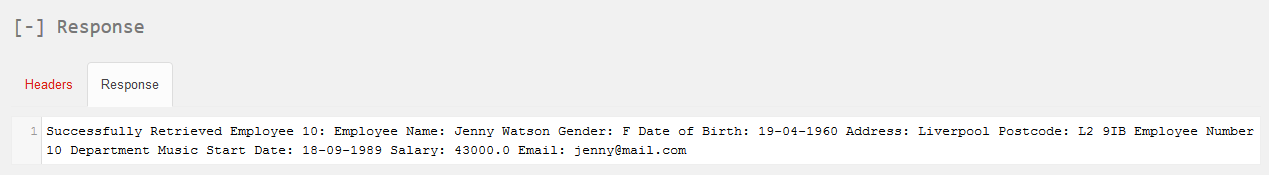
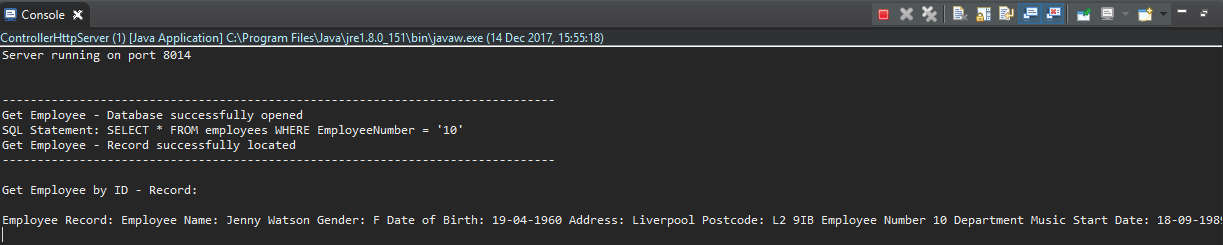


The ProcessEmployeeRetrieveHandler class includes code for allowing the user to search the database for a single employee record with a specified employee number. This code takes in a request, which takes the form of an existing employee number. The employee record attached to that employee number is 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:**

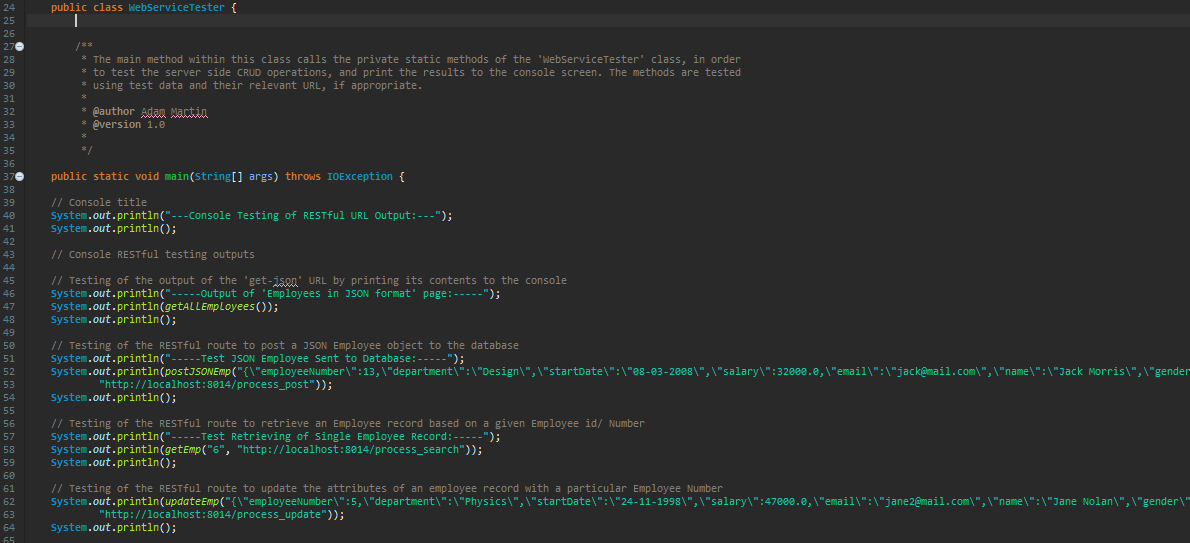


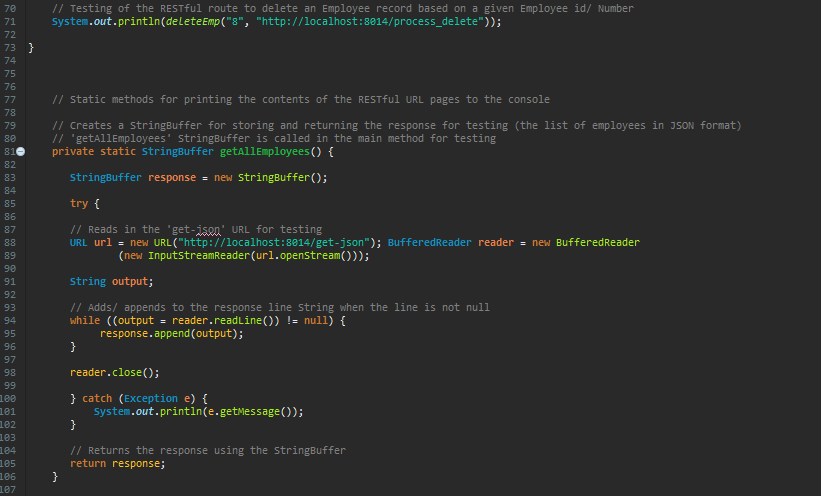


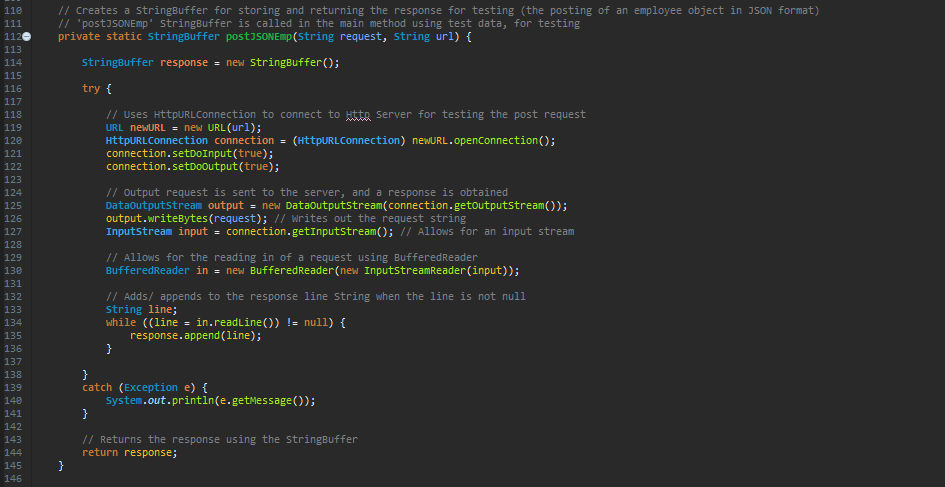


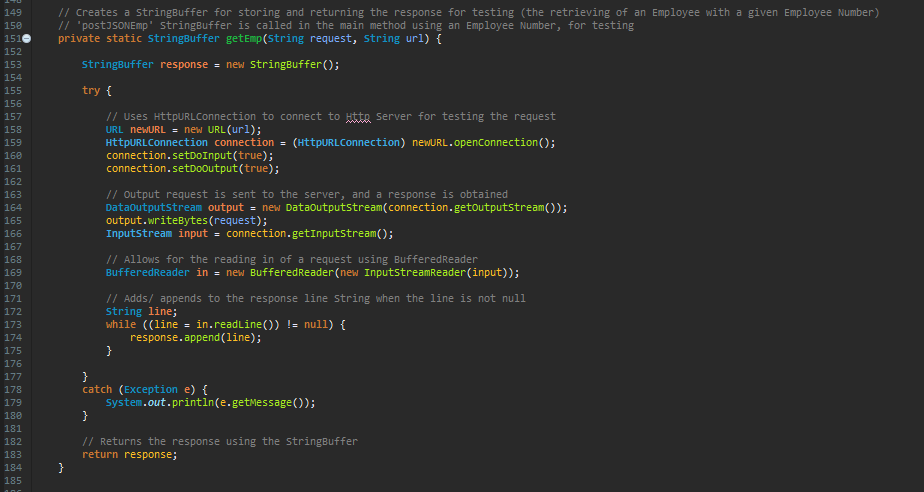
These screenshots depict the successful testing of the single employee retrieve. A request to delete the employee record with an employee number of ‘4’ was sent. As you can see, the employee was successfully retrieved, and a positive ‘200 OK’ response header was returned. The employee record displays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

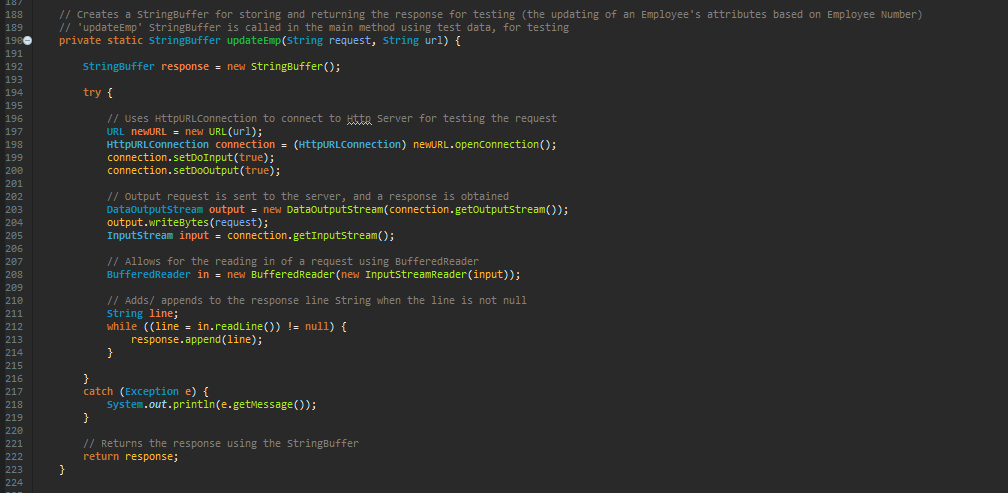
**Step 13 – ‘WebServiceTester’ Class:**

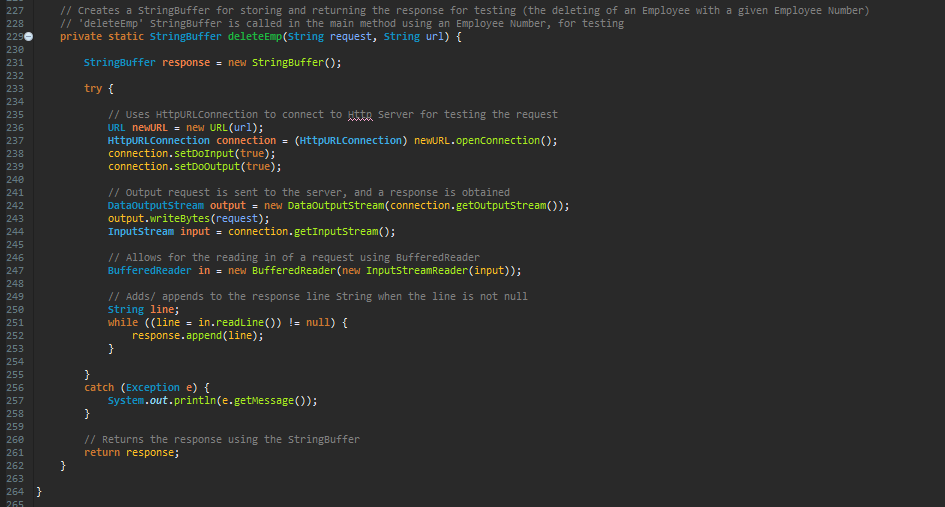




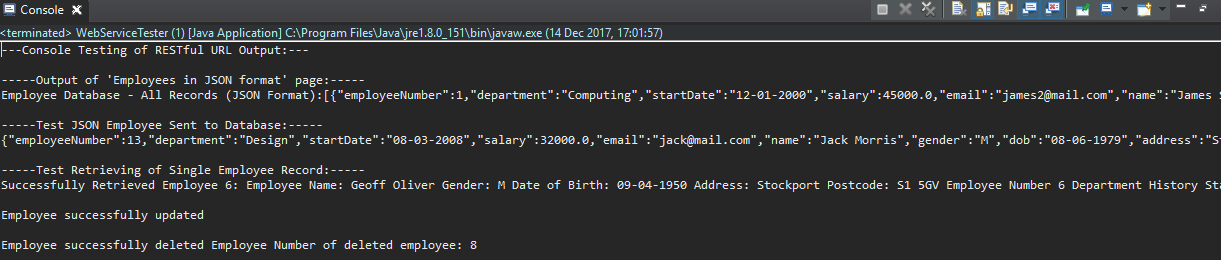


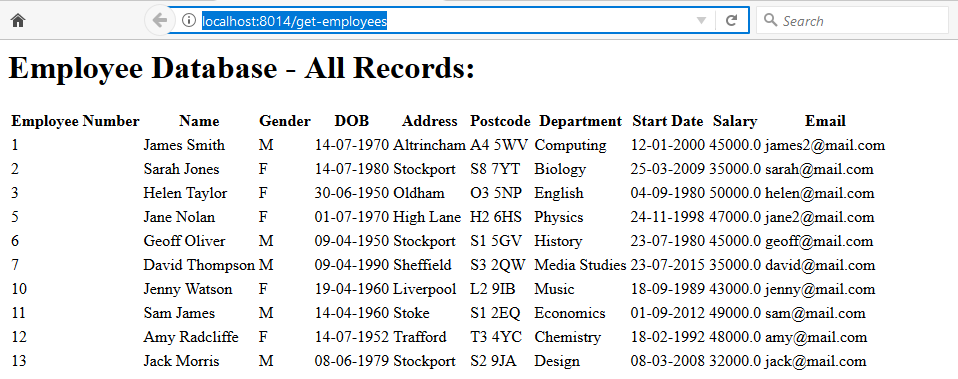






The WebServiceTester class includes code for testing the functionality of the server-side handler CRUD methods at the console. The class tests methods for the getting of all employees in JSON format, the posting of a JSON employee object, the retrieval of a single employee object, the updating of an 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)**

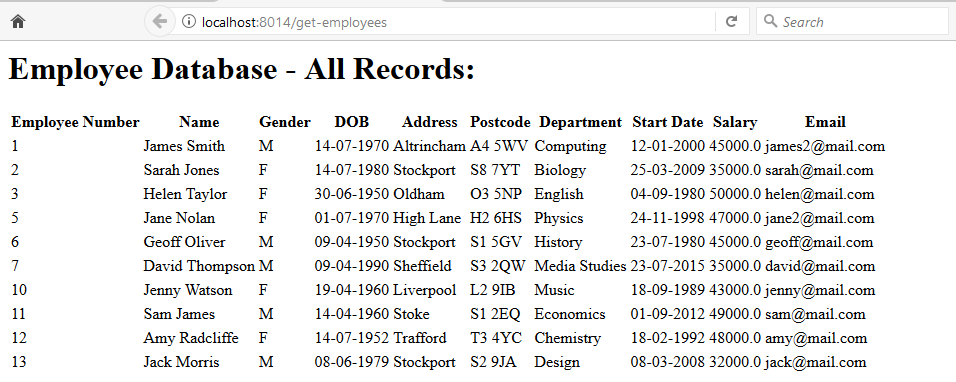
**Evidence of the New Employee Record (13, Jack Morris) in the Database:**

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

**Existing:**

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

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



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.