Assignment 5 (0.5 points) - Deadline Aug 21 23:59 hrs

Learning goals

- How to use the Derby database engine in embedded mode.
- How to use the JDBC API to
  - Connect to the database.
  - Send queries to the database.
  - Retrieve and process the results received from the database.

Persistent Employee Database

In this assignment, you will have to implement the *assignment5.EmployeeDB* interface. This interface is exactly the same as given out in assignment 3. In order to implement the interface you will use Derby database engine in embedded mode[1]. You will have to implement the stubbed out methods in *assignment5.DerbyEmployeeDB* class like the *SimpleEmployeeDB* class in assignment 3. We have also provided you with the test class *assignment5.DerbyEmployeeDBTest* to implement the test cases. If you wrote test cases in assignment 3, you can just copy the test cases you wrote. You are more than welcome to add more. *However, its essential for this assignment to implement test cases to test the interfaces defined in EmployeeDB.* Remember this is not a communication (HTTP) assignment.

Implement the assignment5.EmployeeDB interface

- You should use the Derby database engine (using JDBC) to implement the EmployeeDB interface. The `assignment5.DerbyEmployeeDB` class loads the Derby database engine and creates a database called `employeeDB` (if it does not exist). Implement the following stubbed out methods in *assignment5.DerbyEmployeeDB* class using JDBC:
  a. `addEmployee`: Adds an employee record in the employee database (employeeDB). The method throws an exception if an employee record is being added for a department that does not exist in the employee database. The list of departments which are handled by the database is passed when the database is instantiated in the constructor.

[1]Derby is an Open source relational database engine which can be run inside the JVM in which the application executes. http://db.apache.org/derby/. The Derby jar files (http://mirrors.dotsrc.org/apache//db/derby/db-derby-10.11.1.1/db-derby-10.11.1.1-bin.zip) have been provided in the lib directory of the assignment handout. Make sure you add the lib directory in your classpath and you have a database :-).

b. `listEmployeesInDept`: Returns the list of employee records who belong to the departments passed as argument to this method in the employee database (employeeDB).

c. `incrementSalaryOfDepartment`: Increments the salaries of the employees in the departments passed as argument by the amounts passed as argument. Ensure all/nothing semantics[2]. Possible causes of failure can be :

   (a) One or many departments passed as argument do not exist in the employee database (employeeDB).

   (b) The amount by which the salary is to be incremented is negative.

   This method throws exceptions to signal which of the above errors happened.

d. `cleanupDB`: Resets the state of the employee database to an empty state (newly created employee database).

- *Remember you will have to create the database schema (table layout/s) to store the employee records in the database (employeeDB). You are free to architect that as you wish. We have not implemented that for you.*

Implement the test cases

- Implement the test cases for `assignment5.EmployeeDB` interface in the `assignment5.DerbyEmployeeDBTest` class. You can reuse the test cases created in assignment 3 (if you have created any) and create more if you want.

OPTIONAL - Plug in DerbyEmployeeDB class instead of SimpleEmployeeDB in assignment 3

In order to play around more if you want to, you can plug in the DerbyEmployeeDB class that you implemented above, into your code for assignment 3 so that the EmployeeDBHTTPHandler invokes methods on DerbyEmployeeDB class instead of SimpleEmployeeDB class. The whole setup should just work, thanks to interfaces :-). You can also play around with JVM tuning parameters like heap size, garbage collection and also run a profiler to see if tuning the JVM affects the performance of your entire application. The way to stress-test the application would be to write intensive JUnit test cases (creating, updating and reading lots of records). This way you can potentially stress test and tune an application much more than functional testing can ever achieve.

[2]Either all records are incremented or none are incremented in the employee database (employeeDB).