

PROJECT: DATA LOGGER

[A User’s Guide to our Java-SQL project]



**I Introduction**

**II Subject Description**

**III Subject Analysis**

III.1 Major Features

I[II.2 Application Feasability](#_30j0zll)

III.3 Data Description

III.4 Expected Results

III.5 Algorithms Study

[III.6 Scope of the application](#_1fob9te)

**IV Conception**

IV.1 Chosen Algorithm

IV.2 Data Structures

IV.3 Global application flow

IV 4 [Global schema and major features schema](#_3znysh7)

**V Console operations description**

**VI Configuration instructions**

**VII Commented Screenshots**

**VIII Bibliography**

I: INTRODUCTION

This document is here to serve as a user-friendly guide to the Java project that is to be submitted. In layman’s terms, this document would detail the concept, the objective, the features of the program, how to use it correctly, the theories that exist behind the features and the results that the program would give under the correct jurisdiction.

II: SUBJECT DESCRIPTION

This project is built around one concept: To take information from a user pertaining to a specific individual, group of individuals and store the information in a database, with specific operations to create said database, edit entries or stacks of entries in the database, update the entries in the database and delete said entries.

The project is implemented in Java language, on the platform Eclipse ver. Oxygen, along with the extension for Apache Derby to utilize SQL Language markup for storage.

III: SUBJECT ANALYSIS

This project is centered around the idea of database creation and the submission of information via user input, the editorial of said information, its selection and deletion. This is followed up with the documentation and the user guide. The program makes use of the Java programming language to create the code needed, organized in classes and packages for easy organization, and makes use of the SQL language in order to create datasets to store the information in.

The documentation is exported in .pdf form and contains all the technical details of the program contained within and the user guide contains, as indicated by the name, a guide for new and returning program users in order to properly access program features and gain a successful output.

III.1 Major features:

The program makes use of the Java programming language and is executed with the Eclipse Oxygen platform and JDK versions 8.0 and above. In addition to this, it makes use of the extensions for Apache Derby in order to provide the SQL database creation capabilities, the JDK extension kits in order to establish connections between the Java platform, the SQL extension platform and the file system in between.

This project makes use of the Eclipse Oxygen program, along with extensions to Apache Derby in order to access the SQL Database and the Github extension to update the files to Github for submission.

III.2 Application Feasibility:

The program makes use of classes across many packages, which are connected using the property of inheritance and polymorphism that is applicable in the Java language.

There’s also use of the import and package keywords that enables the programmer to make use of the different kinds of packages available in the Java development kit, which include the use of keywords that would access the File System and the SQL Development language by establishing connections between the two of them as well as the keywords required to perform input, output operations in the File System and creation, updating and deletion of schemas and information in those schemas in the SQL language.

All of this could be implemented in a simple Java program, drawing upon connection features in Java packages that can be accessed with simple implementation keywords, and into SQL languages. As it stands, the feasibility of the application is at the ‘easily realisable’ level, would take less than a week to implement.

III.3 Data Description

The Create\_Identity class will create the Database necessary to store the information in. It comes with its own statements to create both the empty database and the first entry into that database. As a special note: the Create\_Identity class must be executed first.

The Select\_Identity class will search the database for particular information given to it and return any and all results that it will come up with. The search quota that it fulfils relates to the chosen user’s name.

The Update\_Identitiy class functions similarly to the Select class; it will search the database for the results pertaining to a particular user name and will then update the entry in the database with the new information provided by the person invoking the query at that particular point in time.

The Delete\_Identity class also functions similarly to the above two classes; the results that the SQL search query pulls up will then be deleted from the table.

The MyFormatter class inherits from the Formatter class and is specifically used in order to format the error messages that the class Logger exports.

The Logger class utilizes the Myformatter class in order to export error messages should the program fail to execute.

III.4 Expected results:

Depending on the choice of query the user inputs, they would receive different results:

-Invoking the Create\_Identity class would result in the creation of an empty database, along with the first entry into the database.

-Invoking the Select\_Identity class would search up a query in the database related to the name column of the database and display all the information attributed to that query.

-Invoking the Update\_Identity class would use a search query in the database to locate the entries in the database and replace the results with an updated version of the information provided by the user. Therefore, the result would be the display of updated information in the table.

-Invoking the Delete\_Identity class would first invoke a search query related to the name column in a database and eradicating the results from the database, which would then result in the display of a database with the searched query missing.

All output statements and results have been isolated to the classes that run them, save for the error messages that execute from a common Logger class across all four SQL-directed classes.

III.5 Algorithms study:

There are different kinds of algorithms considered when designing a program around a simple, yet open-ended concept. Such a program has the same sort of architectural building blocks, yet can be implemented in a multitude of ways.

Listed below are the algorithms, starting with the one employed in the program to those briefly considered for the process.

Algorithm #1: Java-SQL implementation

The algorithm makes use of simple connection features between a Java platform and its SQL extension in order to take and store data. It takes in the information to be logged, modified, selected or deleted from the user and modify the database according to the user’s wishes.

All information must be first logged using the Login class and then depending on the operation that the user chooses, one specific class is selected, messages output and the information is either stored in the database, edited, replaced with updated information or deleted in its entirety.

Algorithm #2: The XML formula

The XML algorithm resembles the Java-SQL implementation and features even the same steps up to data submission, with the only difference of using XML as the markup language, instead of SQL. Both algorithms were briefly considered, before going with SQL, because of the familiarity and experienced attached with this language.

An algorithm for this would remain similar to the one currently employed, with the noted difference of using XML markup language.

Algorithm #3: File System

The File System algorithm is one where the third storage option would be to utilize Java File Systems in order to store data. This would be the easier of the three to implement due to the lack of requirements for extensions. It only requires the use of the import keyword to use the package required to access the file systems and creation of a new file in order to store the information. However, this algorithm was not considered, due to the lack of categorization of the information stored in the file, the extra processes and steps required in order to edit particular entries in the file and in deletion of the information for the file.

III.6 Scope of the application

The application works over a global scope, interacting with classes and packages that exist outside the local scope in the Java platform, as well as connecting to different platforms altogether such as interacting with the SQL platform.

IV: CONCEPTION

The program’s conception is based on a single concept: To take the information provided by the user and store it in a database for later use, with the added options of selecting and displaying it, modifying the data by erasing it in the storage and replacing it with new data provided by the user, as well as deleting the data entries in its entirety.

In order to do this, we have made use of both the Java language and the SQL language, the Java language to execute commands we leave in the code to act as in-built responses to user queries, establish a connection to this SQL language and create databases in the SQL language to store the information in. This is done by making use of the Eclipse platform to write the Java code and making use of the Apache Derby extension in order to access SQL scrapbooks to store databases in.

IV.1 Chosen algorithm

The algorithm chosen for the concept here is one employing steps to connect the Java program directly with the database, with use of connectivity classes and keywords.

The SQL operations have been divided into four classes, each satisfying a particular operation objective, to implement and modify data as according to user wishes.

IV.2 Data structures

The Java program as a whole is divided into seven classes, five of which are directly tied to program operations. One class is used to take in information in the form of user input and the other four are used to either log the information in the database associated with the program or manipulate already logged-in information from the

The main SQL operations are divided into four classes which connect to the SQL database via extensions that they inherit from the JDK packages. Each operation pertains to a particular class.

The Create\_Identity class has two main functions; to create and empty database and upload the first bit of information to be stored. This class must be executed only once at the beginning; all other operations must be carried out by the other three classes. As a special note: the Create\_Identity class must be executed before any other operation, as lack of a database would create an error in modifying, updating or deletion of information.

The Select\_Identity class will search the database for particular information given to it and return any and all results that it will come up with. The search quota that it fulfils relates to the chosen user’s name. Once it’s pulled up results relevant to the search quota, it displays them on the screen.

The Update\_Identitiy class functions similarly to the Select class; it will search the database for the results pertaining to a particular user name and will then update the entry in the database with the new information provided by the person invoking the query at that particular point in time. Once it has done so, it will display a system message stating that the database had been updated successfully.

The Delete\_Identity class also functions similarly to the above two classes; searching up a record that would be relevant to the information given by the use, which then ends with the results that the SQL search query pulls up will then be deleted from the table and displaying a system message to indicate when the deletion has completed.

The other two classes are interlinked with each other and act as a functional unit in order to provide error messages in case the main four SQL classes and the other Login class has some issues resolving in output. They also do provide proper system messages for when the classes execute correctly.

The MyFormatter class inherits from the Formatter class already defined in the Java packages in order to be able to act on the Logger class and format messages as according to the situation.

The Logger class acts as an output class and polices the output generated by the classes other than MyFormatter. In cases of error, it outputs the appropriate error messages through the selected class inheriting its features and printing the correct error message. It also prints the appropriate message should the class present its output correctly, therefore for both cases, it covers the system output.

IV.3 Global application flow:

On a global scale, there exists the phenomenon of inheritance. Inheritance from the classes defined by the programmers in their own unique packages, as well as packages that are part of the Java utility and SQL package through the import and package keywords. Through this, classes from outside of the packages currently being implemented can access each other, in order to continue carrying out processes while maintaining a sense of categorization, from a programming standpoint and an aesthetic standpoint. The processes are detailed as below:

All of the classes inherit from the Logger class in order to perform basic output features depending on the situation.

Create\_Identity, Select\_Idenitity, Update\_Identity and Delete\_Identity inherit from the sql and util packages in Java in order to perform database operations on modifying and storing information.

MyFormatter inherits from Logger and Logger inherits from MyFormatter in order to format the output messages that Logger is in charge of, in accordance with the type of situation: specific error messages for specific failure of execution and proper output messages should the program carry out operations successfully.

IV.4 Global schema and major features schema:

//UML

V: CONSOLE OPERATIONS DESCRIPTION

Create\_Identity Class:

The class first starts with an implementation of the Logger Class, and getting it prepped for outputting messages.

Then, the database connection constants are defined as final and String. This remains the same for all the classes. Then the variables of the class are set, that are used to define the columns of the database.

Next comes the input messages. They would have to be specific for every kind of field in the table.

The CREATE SQL and INSERT SQL statements are executed in.

All following executions are in the first try loop, in order to catch errors.

A connection is defined by first registering the JDBC and establishing a connection. The connection is validated by inputting the DB\_USER and DB\_PASSWORD

Then the table fields are defined. All the user information will be categorized according to this information, and any changes would have to be made in this section.

The empty table is then created.

And the entries are input into the table via setting it in preparedStatement and then inserting it in the INSERT SQL statement.

The INSERT SQL statement must be modified according to the amount of columns defined.

And the execution.

This try loop execution ends and the catch loop implementation follows, in order to set up error messages should the program fail.

The finally block is executed at the end. Should there be no errors, the finally block would be used to close the dbConnection. The catch loops that follow is for if there is a failure in closing the connection.

Select Class:

The class first starts with an implementation of the Logger Class, and getting it prepped for outputting messages.

Then, the database connection constants are defined as final and String. This remains the same for all the classes.

The SELECT SQL statement is placed here with to field to be searched for following the WHERE keyword.

All following executions are in the first try loop, in order to catch errors.

A connection is defined by first registering the JDBC and establishing a connection. The connection is validated by inputting the DB\_USER and DB\_PASSWORD

Here the SQL statement is executed. If there is the option of returning multiple results, then they are executed through a while loop. And the execution.

This try loop execution ends and the catch loop implementation follows, in order to set up error messages should the program fail.

The finally block is executed at the end. Should there be no errors, the finally block would be used to close the dbConnection. The catch loops that follow is for if there is a failure in closing the connection.

Update Class:

The class first starts with an implementation of the Logger Class, and getting it prepped for outputting messages.

Then, the database connection constants are defined as final and String. This remains the same for all the classes. Then the variables of the class are set, that are used to define the columns of the database.

Next comes the input messages. They would have to be specific for every kind of field in the table.

The UPDATE SQL statement is placed here with to field to be searched for following the WHERE keyword.

All following executions are in the first try loop, in order to catch errors.

A connection is defined by first registering the JDBC and establishing a connection. The connection is validated by inputting the DB\_USER and DB\_PASSWORD.

The entries are input into the table via first loading it in the preparedStatement and then the UPDATE SQL statement.

Then the appropriate message is displayed, if there is no errors and the try loop closes.

This try loop execution ends and the catch loop implementation follows, in order to set up error messages should the program fail.

The finally block is executed at the end. Should there be no errors, the finally block would be used to close the dbConnection. The catch loops that follow is for if there is a failure in closing the connection.

Delete Class:

The class first starts with an implementation of the Logger Class, and getting it prepped for outputting messages.

Then, the database connection constants are defined as final and String. This remains the same for all the classes. Then the variables of the class are set, that are used to define the columns of the database.

Next comes the input messages and the place to take in input. They would have to be specific for the type of keyword they conduct the search, in this case we go by uid

Followed by the DELETE SQL statement, with the search criteria following the WHERE statement to easily identify the record.

The first try loop starts at this point.

All following executions are in the first try loop, in order to catch errors.

A connection is defined by first registering the JDBC and establishing a connection. The connection is validated by inputting the DB\_USER and DB\_PASSWORD.

The input is loaded into the table via first loading it in the preparedStatement and then the DELETE SQL statement.

If the class proceeds without any errors, then the appropriate output messages are prepared.

This try loop execution ends and the catch loop implementation follows, in order to set up error messages should the program fail.

The finally block is executed at the end. Should there be no errors, the finally block would be used to close the dbConnection. The catch loops that follow is for if there is a failure in closing the connection.

VI CONFIGURATION INSTRUCTIONS

Should the user be wont to make changes to the program according to their own wishes, this is the guide in order to make changes.

If the user wants to insert more columns in the database, in order to increase the number of specifications attributed to a user record, then the user would need to first make changes in Create\_Identity.

This would create an empty database with the correct amount of columns(specifications) and then insert a first entry accordingly. The user would also have to make changes to the amount of entries in the Update\_Identity and Delete\_Identity classes.

With the Select\_Identity class, changes would be made depending on whether the user wants to search for more than one specification. Currently, the user only searches by uid, but if need be the user can adjust the SQL statement and inputs.

It should be noted that the MyFormatter and Logger classes should not have their code changed in any way to affect the outputs; if the user wishes to change the error messages or even the normal system output, then all they would have to do is change the output assigned for the class concerned and not change the code in MyFormatter or Logger.

VIII BIBLIOGRAPHY

References:

<https://stackoverflow.com>

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

<https://www.lucidchart.com/?re=1>