**CSVContents**

This class takes a location and stores the contents of a .csv file in an ArrayList<String[]> format. The stored contents includes every row of the .csv file. Note that each row gets stored one by one, so if an error occurs, the instance variable allRows may be only partially complete. The ArrayList<String[]> follows the data pattern: each row of the .csv file will be split into a String[] where each cell is a separate index; the String[] is then placed in the ArrayList.

**DataContents**

Takes the entire content of an ArrayList<String[]> and stores it according to an appropriate format.The appropriate format is according to the table that is being interacted with. The appropriateformat will match the column ordering of the table and will shear the first row if the first row iscomprised of a header and not actual data. Note, for every record that has an empty column entry, nullwill be stored at that column entry's index.

**DataTypeFormatter**

Logic to convert a record into a properly formatted String[]. Properly formatted means that each value of the record will be formatted to fit its respective data type. For example, in Oracle a date value of 9999-12-31 in the record would be formatted to TO\_DATE('9999-12-31', 'YYYY-MM-DD').

**SQLCommands**

Formats strings for the basic SQL commands: create, select, modify/update, and delete for record(s) in a table.

**UserDataFormatter**

This class acts as a logic container for formatting incoming user data to fit a table in a database.

GraphicalUserInterface

displays the interface that the user will be working with. Will have a select document for the .csv location, a table selector for the table to do things with the record, a column selector that will tell which column in the .csv corresponds to what data, a task number input dialogue, whether they want to delete, update, or insert records.

**InsertUserRecords**

Inserts records into a table given a String[] array of SQL queries.

**SQLFormattedRecords**

Takes the entire content of a String[][] and stores it according to an appropriate SQL format. The appropriate format is according to the database type being used. The appropriate SQL format will match the syntax of a hypothetical insertion of the data. For example, an entered date 9999-12-31 may be converted to TO\_DATE('9999-12-31', 'YYYY-MM-DD') depending on the database type.

**TableCollection**

Structure to contain all of the TableInformation objects in an accessed database. A database is passed in and then all of the tables in the database are created as TableInformation objects. The TableInformation objects are then stored in an ArrayList<TableInformation> as an instance variable.

**TableInformation**

Contains information on the overview of a table in a database. The contained information includes an identification, the name of the table, the column names and their respective data types in correct ordering with respect to the table in the database. Note that the default for the TableInformation's identification instance variable is the table name passed in. The TableInformation identification is able to be changed but the TableInformation's table name is declared final.

UserInteractor

handles the interactive parts of the GUI,

such as bringing up the window for selecting the file location,

bringing up the selectors for the table to insert into, etc.

**UserPreferences**

Dummy object to store all of the user preferences so as to allow for easier data acquisition by other classes. The user's selections will be stored here and then passed on to different classes that will then call getter and setter methods. When UserPreferences object is created, it has the following default values: headerStatus = false.