Exercise #4 – Exception Handling

ISTE-330 Database Connectivity and Access

Assignment Purpose: Practice implementing exception handling.

*Do all steps in **Java** for **MySQL** and use the **Travel** database from Exercise 3. (100% pts)*

On submitting your assignment, make sure that you provide all Java files (or the entire .Net solution structure). Submissions lacking necessary files or which cannot be compiled will not be accepted or graded!

Description

In class we discussed the importance of trapping all exceptions to prevent any structural information from being exposed. This exercise will require you to handle problems such as malformed queries or changes to database structure.

Create a new data layer object class named "**DLException**" — a custom exception class.

1. Provide a **constructor** that accepts a **single parameter** of type **Exception**
   - Adapt it to extract all (important) possible information from received **SQLException** that needed to be logged
   - Could also be used to automatically initiate **saving to log** the most common exceptions.
2. Provide another **constructor** that accepts a **parameter** of type Exception and **additional string values**.
   - IT should do practically all things that the constructor above does
   - Use it to provide **additional logging info** about the specific exceptions and their cause (other information passed from caller/thrower such as action, location, user info, etc.)
   - For example, if needed multiple String values could be passed in a form of two parallel arraylists, or a single 2D arraylist, or a map, or in any other acceptable form.
   - For SQLException you need to handle all database-related info, including reason(s) for the exception, including **description** of the exception, **SQLState** code, and vendor **error code**.
   - For other exceptions, pass **any available additional information**.
3. Provide a method named "**log**" that **writes** out any available information about the exception (from the exception itself and from the additional string values and any other info you can collect) to a **text log file**, together with a **timestamp** and all other necessary data.
   - It should be called each time when an exception occurs to log all available specific exception info.
4. Set the DLException's new initial **message** to some innocuous string indicating failure (e.g. "Unable to complete operation. Please contact the administrator.") which will be presented to the user.
   - You may provide some details but don't be too revealing.

Modify all your **classes** from Exercise 3 to **catch all possible database-related** (and other) **exceptions,** including **SQLExeption**, in **all** classes and methods **properly**. When SQLException or other exceptions are caught, process them, and always **throw** a new **DLException**. Please note that the DLException's new "user" message should be, when caught, the only one presented to the user (don't reveal other info).

Now, make and **intentional mistake** to test it, e.g. in the database, change the name of a column in the equipment table in order to force an error, and run your code to **test it**.