/\*\* A NinjaConn is the object which does all communicating between the

\* application and the local SQLite database. A NinjaConn should only

\* be initialized by a LogIn object, which will disconnect the

\* NinjaConn if the LogIn security check fails. If LogIn succeeds,

\* then the NinjaConn is passed to the main application space, where

\* other objects within the application package can use it to retrieve

\* and update data in the local database.

\*/

protected class NinjaConn {

/\*\* A Connection object from the java.sql library, used to

\* establish a connection to the local SQLite database.

\*/

private Connection conn;

/\*\* A Statement object from the java.sql library, used to pass

\* query and update SQL statements to the local database.

\*/

private Statement stmt;

/\*\* A text String for constructing SQL statements. sqlText is

\* generally constructed and then passed to the Statement object

\* (stmt) in order to execute a query or update in the SQLite

\* database.

\*/

private String sqlText;

/\*\* A ResultSet object from the java.sql library. rSet is used

\* to hold the resulting data from a SQL query executed by the

\* NinjaConn. rSet is then generally returned to the calling

\* object, which can then make use of the data contained in the

\* ResultSet.

\*/

private ResultSet rSet;

/\*\* Constructor which connects to the application database, or

\* generates an error message and returns null if the database

\* cannot be reached. A NinjaConn should only be initialized from

\* the LogIn class, which will disconnect the NinjaConn if the

\* LogIn security check fails.

\* @post conn != null

\* @post stmt != null

\* @post rSet = null

\*/

protected NinjaConn() {..}

/\*\* The quID() operation queries a row specified by the passed

\* value of id, in the table specified by the passed table String.

\* It will then return the resulting ResultSet, which will contain

\* all row information as long as the table was found in the

\* database, and the id was found in the table.

\*/

protected ResultSet quID(int id, String table) {...}

/\*\* The quName() operation queries a row specified by the passed

\* name String, in the table specified by the passed table String.

\* It will then return the resulting ResultSet, which will contain

\* all row information as long as the table was found in the

\* database, and the name was found in the table.

\*/

protected ResultSet quName(String name, String table) {...}

/\*\* The quUsername() operation queries a row specified by the

\* passed username String (uname), in the table specified by the

\* passed table String. It will then return the resulting

\* ResultSet, which will contain all row information as long as

\* the table was found in the database, and the username was found

\* in the table.

\*/

protected ResultSet quUsername(String uname, String table) {...}

/\*\* The quGetAll() operation queries an entire table specified by

\* the passed table String, or generates an error message if the

\* table does not exist or otherwise cannot be queried.

\* quGetAll() will then return a ResultSet containing the data for

\* the entire specified table, as long as the query executed

\* successfully.

\*/

protected ResultSet quGetAll(String table) {...}

/\*\* The updateDBDouble() operation will update a double value

\* (SQLite REAL type) in the local database using the passed value

\* of newVal. The table is specified by the passed table String,

\* the row is specified by the passed id int value, and the column

\* heading is specified by the passed field String. This

\* operation will generate an error message if the SQL update does

\* not succeed. This operation does not change the instance

\* variable associated with the calling object, but relies on

\* those objects to update their own instance variables.

\*/

protected void updateDBDouble(String table, int id, String field, double newVal) {...}

/\*\* The updateDBInt() operation will update an int value (SQLite

\* INTEGER type) in the local database using the passed value

\* of newVal. The table is specified by the passed table String,

\* the row is specified by the passed id int value, and the column

\* heading is specified by the passed field String. This

\* operation will generate an error message if the SQL update does

\* not succeed. This operation does not change the instance

\* variable associated with the calling object, but relies on

\* those objects to update their own instance variables.

\*/

protected void updateDBInt(String table, int id, String field, int newVal) {...}

/\*\* The updateDBString() operation will update a String value

\*(SQLite TEXT type) in the local database using the passed value \* of newVal. The table is specified by the passed table String,

\* the row is specified by the passed id int value, and the column

\* heading is specified by the passed field String. This

\* operation will generate an error message if the SQL update does

\* not succeed. This operation does not change the instance

\* variable associated with the calling object, but relies on

\* those objects to update their own instance variables.

\*/

protected void updateDBString(String table, int id, String field, String newVal) {...}

/\*\* The getDPW() operation is an experimental method which will

\* eventually provide stronger security for

\* password encryption and log-in security. However, at present

\* the security is too strong to function effectively, and this

\* method is deprecated.

\*/

private String getDPW(String name) {...}

/\*\* The hexToByteArr() operation is a private method used in

\* encryption and decryption. This operation takes a hexadecimal

\* String and converts it to an array of bytes. The byte array is

\* then returned to the calling method within the NinjaConn.

\*/

private byte[] hexToByteArr(String hex) {...}

/\*\* The bytesToHex() operation is a private method used in

\* encryption and decryption. This operation takes an array of

\* bytes and converts it to a hexadecimal String. The String is

\* then returned to the calling method within the NinjaConn.

\*/

private String bytesToHex(byte[] bytes) {...}

/\*\* The getAndD() operation takes a passed username, and then

\* gets the associated encrypted log-in password, decrypts it, and

\* returns the plaintext password. This is a private method that

\* should only be called from the checkAccess() operation below

\* for added security. See the checkAccess() operation for

\* information on usage of the decrypted password.

\*/

private String getAndD(String uname) {...}

/\*\* The checkAccess() operation should be called from the LogIn

\* class, and provides an extra layer of encapsulation and

\* security for password decryption. It takes a passed password

\* (pw) and username (uname), and then calls the above getAndD()

\* private method in order to retrieve the decrypted password

\* associated with the passed username. This operation then

\* compares the retrieved value with the passed value, and returns

\* true if they match. If they do not match, the operation

\* retrurns false.

\*/

protected Boolean checkAccess(String pw, String uname) {...}

/\*\* The eAndStore() operation is used to encrypt a passed-in

\* password String, and then store that encrypted password and its

\* associated decryption information in the local database. This

\* operation will return true if the encryption and SQL update

\* process succeeds, and will return false if there are any errors

\* encountered. In the latter case, this operation will also

\* generate an error message.

\*/

protected Boolean eAndStore(String pw, String name) {...}

} /\* end NinjaConn \*/