<u>Deliverable #5: Code Review Strategy</u>

Review your assigned code on the following:

1. Readability & Maintainability: Are each of the functions/methods documented, with details on the function/method's purpose and parameter type/values? Is all the code covered by comments? Are variables appropriately named?
2. Code Design: Are the classes following appropriate design patterns? If not, how can the code be re-designed?
3. Functionality: Is the code's purpose understandable (are you able to follow through the code)? Is there any unnecessary code? Are exceptions/errors handled correctly?
4. Testing: Are there automated tests for the code? Is each test's purpose clear? Are all cases covered?

<u>Deliverable #5: Code Review Summary</u>

<u>Assigned Code:</u>

- Jeremy: database package(DatabaseIO, databaseSession, makeDatabase)
- Violet: gui package, test.accounts package
- Kyle: accounts package
- Bilal: template package, passwords package, main package, database package(databaseAPI and databaseDriver)

<u>Jeremy:</u> DatabaseIO is a lengthy class that encompasses what it stands for, Input and Output for the Database. Both the methods importData and exportData are lengthy as it is required to go through every row and column and properly write the data into an excel format. However there are signs of common operations in which it can be modularized to smaller methods. It may be prudent to break them apart then. The documentation is lacking a little bit, the importData method having no Javadoc and both methods having a few more descriptive comments might work. Of course if refactoring the code to a combination of smaller methods, comments might not be needed. databaseSession is a well implemented class, many of its methods are self-contained and isn't a long series of lines. The only issue is that if one doesn't have a base understanding of databases, or how specifically this is written, one might not be able to make heads or tails of the code with its lack of documentation. More comments is to be desired and

Javadocs would be useful. While a brief one-line description of the method is provided, it should be turned into a Javadoc. Also the internals of the coding, the mechanisms if described with comments would help immensely. makeDatabase is a fairly small class, intialize a connection to the database and to make the database available. With how short these methods are, comments to explain the internals aren't explicitly needed, but again Javadocs would help.

Violet: Code in test.accounts is organized. Each test case is named properly, and it covers all the cases that are need to be tested. No problem for this part. For accessData, login and mainMenu class in Gui package, it would be better to change the first letter of the name to uppercase so it is consistent with other classes. For accessData and login, JavaDocs and documentations are both lacked. And the whole class for accessData is too big, as many methods are contained. Although the naming for methods and variables are good, so understanding the code is not difficult, it would still be better to split them in several classes. In which way the structure would be more clear. For dataUpload, JavaDocs, comments and naming for it are good, the problem is that there are some small warnings around the code. (Those won't affect the working of the code but can be improved, indicated by yellow lines under them.) Same problem is also owned by accessData, DataDownload and ReportGenerator. Design of this code is good. For excelFilter and excelUtils, as they are small, the documentation, naming and design are all good. For GUI, it only requires more documentations to make it perfect. MainMenu is not fully completed, the existing part would require more documentations. DataDownload is not fully completed as well but the existing part is good enough. ReportGenerator's problem is mentioned above, other parts are well-coded.

Kyle: The accounts appears to be stubs. Nothing functional is implemented except for perhaps the factory class. There is no real differentiation with each user account type because they all extend a generic User class that only holds setters and getters. Code is highly maintainable, easy to read and manage. It follows many if not all SOLID design principles and has been thoroughly tested with only weaknesses prevalent in the lack of functionality, which is meant to be added in later features.

Bilal: The main class is fairly small, consisting of the few lines of code required to check for the SQL database file, create one if it doesn't exist, and start up the first GUI panel. There are several commented out lines of code that were used previously for testing. Unless these lines of code are required for explaining unit testing or acceptance testing, they should be deleted before final release of our product. The databaseConnector class consists of a very simple method that connects to the database, it's easy to understand. We should fill in the JavaDocs for the method regardless though. The passwords class is also very easy to follow and understand, since it simply hashes a password and also has a method to compare passwords. JavaDocs are missing as well, need to be added. The Template class is missing JavaDocs and commenting, which should be added, but is still fairly easy to understand due to the small, modular methods. However, I think this class is no longer needed, since we decided to directly work with SQL databases, we need to confirm this and perhaps delete this class if not necessary. The databaseDriver class is another one of those classes that has a name starting with a lowercase character. The code is used for connecting to the SQL database, is easy to understand and follow. However, the documentation is missing for this class and needs to be filled in. The databaseAPI class was missing documentation before, but the problem has now been rectified. The code has small methods that deal with interacting with the SQL database, each of which has an appropriate JavaDoc, making it even easier to understand the purpose of the code within the methods.