Course: CSC 220.02
Student: Kullathon “Mos” Sitthisarnwattanachai, **SFSU ID:** 921425216
Teammate: n/a, **SFSU ID:** n/a
Assignment Number: 02
Assignment Due Date & Time: 02-26-2020 at 11:55 PM

PART A - OOP Class Design Guidelines, 15 points

Please choose 3 guidelines and discuss them in-depth. For each guideline, use at least one page for your discussion. It is OK to use code to help demonstrate your points. The code portion, if any, should not take up more than 1/3 of each guideline’s discussion.

- Consistency is one of the principles in Liang’s class design guidelines. It dictates the use of naming conventions and programming styles. Consistent naming convention allows the code to be understood and utilized by others. This is important when a project is developed by a group of people. Having a set of conventions agreed upon and adhered to allows the programmers to quickly identify the purpose of the code and keep the project maintainable. For example, the names of variables and methods in Java are lowerCamelCased, whereas the class names are written in UpperCamelCase. Aside from the casing, the actual names of these components are also important. For example, an integer variable declared as `int x;` does not provide any useful information to the programmers you work with (and after some time, yourself) and any context as to what x actually does. Instead, a more descriptive name could be used in its declaration, such as `int numberOfClients;`. Simple rules like this can allow you and other programmers to keep track of your variables and can save a lot of time when trying to identify an issue. Other styling practices such as tab sizes, indentation, and ordering should also be followed for the sake of consistency between classes. For example, in Java field declarations of a class are often at the top (with constants being

at the top of those), followed by its constructors, then its methods (where the applicable setters and getters are placed first). While these conventions are not enforced by the compiler -- meaning the code will still function if you put your instance methods at the top of the class and name your constants in lowercase -- it is still important to follow such conventions as it allows different parts of the class and their purposes to be quickly identified. Having different classes structured in a similar manner allows for a much more effortless design process and can also prevent common errors from arising.

- Encapsulation is another important principle in Liang's guidelines. It states that the fields for a class should use the private modifier in order to hide and restrict its usage within the class (in other words "encapsulate"). Encapsulation allows you to control the accessors of the fields and makes the class easy to maintain as it will prevent other classes from mutating the fields. There are very few exceptions to this rule, with the major one being the usage of constants. Where fields are meant to be used as a constant (i.e., with the "final" and "static" keyword), one may choose to make the field "public" if the constant would also be useful outside the context of the class. The exception permits such a usage since the "final" keyword will prevent the field from mutated once given a value, and the "static" keyword will signify that the field belongs to the class and does not change. In all other cases, if a field needs to be accessed for any reason, one should implement a "getter" method, which returns the value of the field without allowing access to the field itself. Meaning, that members outside the class will still be unable to mutate the field. Similarly, if the field needs to be changed outside the class, one can implement a "setter" method to allow the value of the field to be changed. Implementing a setter method ensures explicit access to the variable and

prevents any unintended changes to the variable.

For more examples, view the source code for this assignment's file to see how all of the fields are kept private, with the fields being accessed via setters and getters when necessary.

- Clarity is a product of good implementation of the two guidelines mentioned above: consistency and encapsulation. In addition, cohesion and "sensible" class design also ensures the clarity of the class. The purpose of the class' data fields should be clear to the user as to how it is meant to be utilized, if accessible. Values that can be derived from existing fields should not be defined in the class as it can create confusion for the end-user. Furthermore, doing so can decrease the maintainability of the class as each of the extra fields added may also need to be updated when any one of them changes. Similarly, any fields that are made to be accessible via setters and getters should follow the naming conventions of setters and getters in order for the user to clearly understand its purpose. Any setters or getters that include validation or processing in any way should also be clearly documented in their methods in order to let the end-user understand how their data may be processed. And for other methods in the class, they should be defined in such a way that it achieves a single task in order to make them intuitive to use. These methods should also follow a straight toward naming convention, usually with a verb in the front, in order to inform the end-user at a glance about its purpose. Any method that involves fields or parameters should be designed in a way that does not hinder the end-user's ability to implement those classes. Methods that rely on input or value of the data fields should be designed in a way that can be invoked

in any order in the end-user's implementation without interfering with the class' functionality.

PART B - OOP, 60 points

- See attached files.

PART C - Database, 5 EXTRA CREDIT points

- Implemented extra support for internationalization for EC instead. Approved by the professor as alternate EC.