1. From Unit 1, we've learned that in OOD encapsulating data is of great significance. So in Unit 2, we have every concrete entity that has concrete meaning wrapped inside a class in the model package.
2. From Unit 1, we've learned inner class can only be accessed through its outer class. In Unit 2, when we want to use some custom onClickListeners inside one class, we choose to use inner class structure.
3. From Unit 1, we've learned that one class's constructor should be defined according to the circumstances that we use it. For example, if we want to have an instance constructed from an instance of another class, we don't want to read all the data out from the latter instance and then call every setter in the former class. So the best way to do it is wrap this functionality inside this class. It can be done either by having a constructor read the latter instance as the input parameter or have one static method to return a newly created instance from given instance of the latter class. In Unit 2, we implemented it in both ways.
4. From Unit 1, we've learned that we should have a clear relationship between classes, and for global parameters' sharing issue, singleton pattern is a good choice to avoid exhausting parameter passing. In Unit 2, we implement singleton pattern for the Client class on the APP (client) side, which should be a unique global instance (only one socket connection for the client side). And this singleton pattern make it very easy and straightforward to share the global variables between Activities/ helper classes
5. From Unit 1, we've learned that implementing interfaces and abstract classes for unifying the access to a family of class is very helpful. In Unit 2, we implement interfaces and abstract classes on both client side and server side.
6. We implement two abstract classes Loader and Sender in the remote package, which unify the constructing and usage of two families of classes. Both the Load and Sender classes extends the AsyncTask class using a template class way. All the loaders/ senders of the entities in the model package, including City, Scene, Schedule, User, extend the corresponding Loader/ Sender abstract class.
7. We implement an interface for all the database accesses on the server side, called DBConnection, to facilitate the accesses to the database. Without an interface, users will find it confusing to have different number of parameters for different senders/ loaders.
8. We also implemented template class when creating the abstract class which extends the AsyncTask class. Given that AsyncTask uses template class method like the way AsyncTask<T1, T2, T3>, we have to also use template class in our custom abstract classes. For instance, the SceneLoader will extend Load<City, Void, Void> to be fully functional (pass in the City instance at runtime).
9. From Unit 1 we've learned about we can adopt multiple inheritance in java by implementing interfaces. In Unit 2 we have 6 different custom adapters for listViews of different usages, all of them extend BaseAdapter and implement ListAdapter interface at the same time.
10. From Unit 1 we've learned that the communication between sockets should be updated periodically, due to the buffer of the sockets may be insufficient to continue. In Unit 2 we have make the socket reset periodically so that they work normally.

11. From the unit 1 we know that the serversocket can listen to one port. Once it accept the socket, it can create a new socket to communicate. In the unit 2, we use it to implement the multithreading of the server.
12. From the unit 1 we know that after readingObject from the socket, the host need to transform the object type to the type we want. In Unit 2, we firstly let App send a certain number before transmitting different types of objects, hence the server can parse them normally.
13. From the unit 1 we know that when sending an object from one host to another, the object type of the both hosts should have the same serialize ID.
14. From the unit 1 we know that socket could enable different objects transmission by reaching a protocol between the sender and receiver. In the unit 2, we send different number before sending different object in the communication between app and server.
15. From the unit 1 we know that socket should be closed to avoid continuing occupying the resource of CPU after used. In the unit 2, we close the socket connection in the server if the app quit.
16. From the unit 1 we know that the close of socket could be implemented in try catch. In the unit 2, we let the operation to close the socket to be happened in the catch exception.
17. From the unit 1 we know that the database normalization can be used to create less redundant tables without information. In the unit 2, we use the database normalization forms through the design of our tables. We store the information of the scene, scheduler, and city separately.
18. From the unit 1 we know that many – to – many relationship could be implemented by a junction table, without destroying the database normalization. In the unit 2, we represent the many – to -many relationship of schedule and scene by using the junction table – schedule_scene.
19. From the unit 1 we know that in a table, we could make a record unique by granting it a unique ID and represent the record by dereferencing the id. In unit 2, we use it through our implementation of databases. For example, we grant each scene a unique ID, and dereferencing the id when we want to get it.
20. From the unit 1 we know that the foreign key can concatenate different tables in databases. In unit 2, we use it through our implementation of databases. For example, the scene table contains the cityID as the foreign key, which reference the id of one record in city table.
21. From Unit 1 we learned to implement custom exception classes and make the software auto-healing. In Unit 2 we applied the self healing mechanism and made the socket connection an auto-healing one, which means if the socket lost its connection, the software will try to connect to the server again when the user tries to log in again.