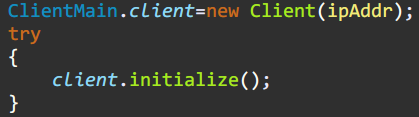
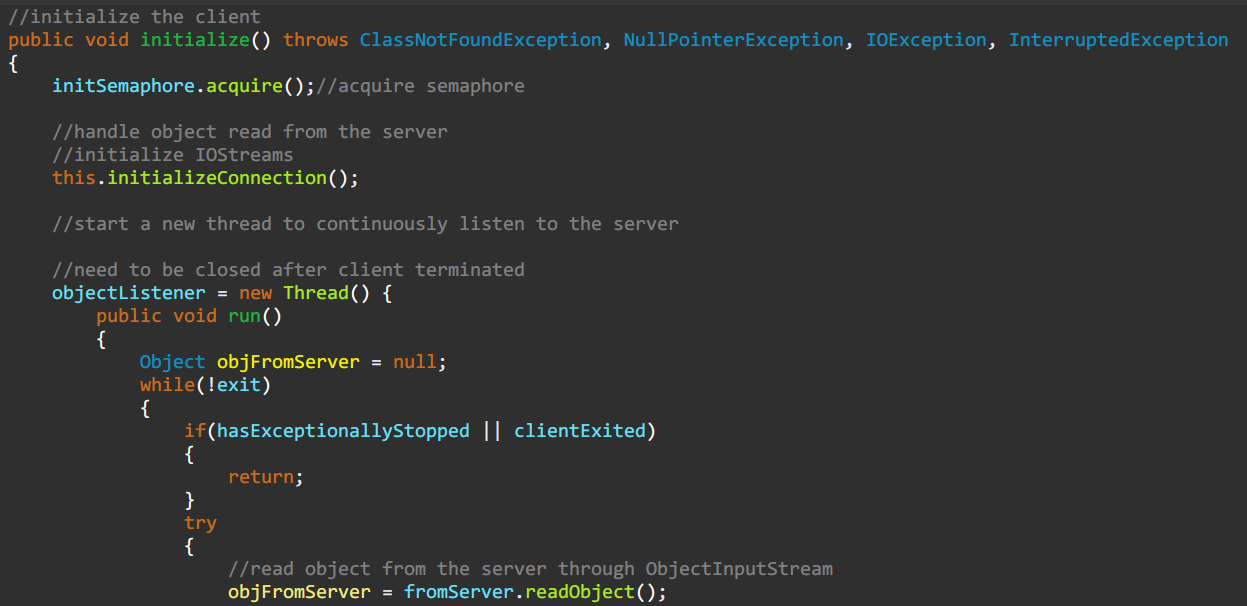
**Object Model**

For the model-controller, it provides a encapsulated method *initialize()* to let the view-controller to initialize an instance of model-controller.

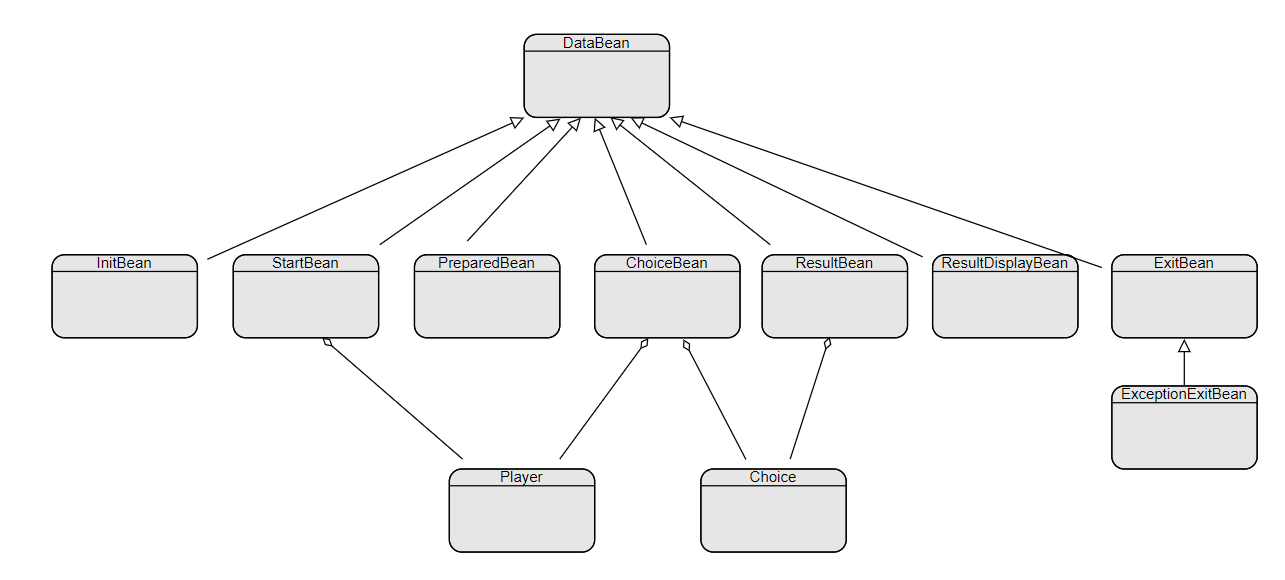
****

*ClientMain (View-Controller)*

****

*Client (Model-Controller)*

Data is encapsulated in the form of JavaBean, including logical Player and Choice that the player has chosen.

****

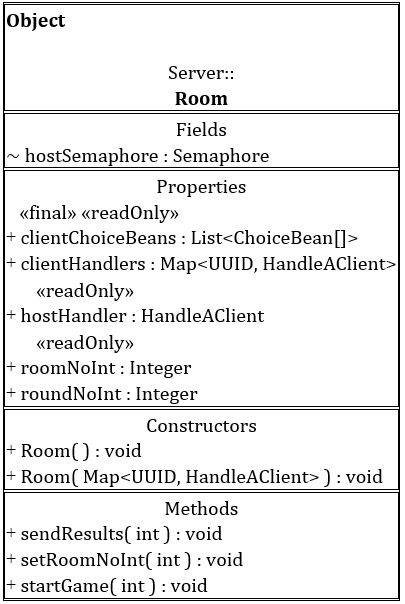
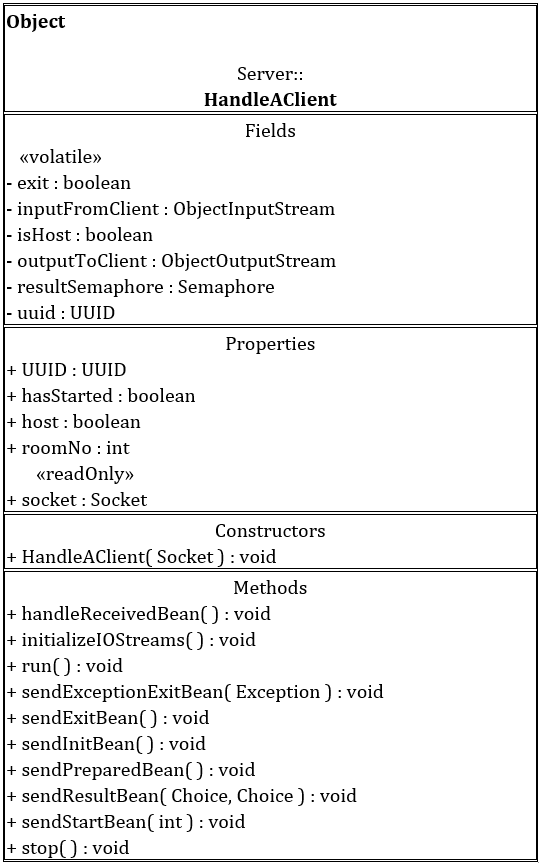
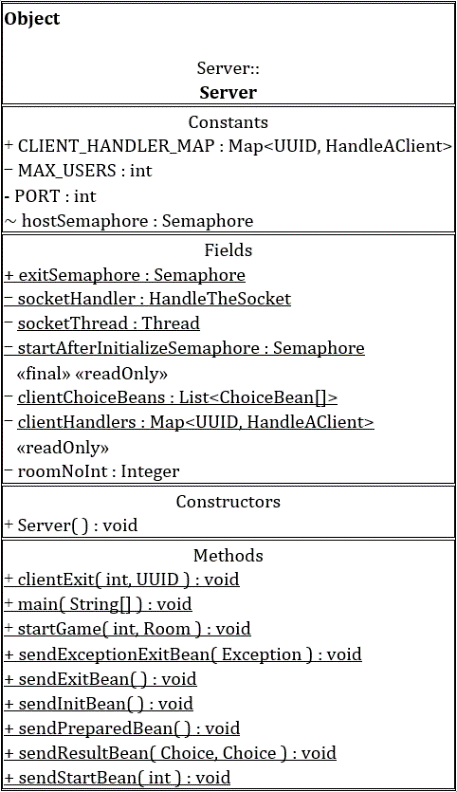
*Model’s Hierarchy*

**Design Principle**

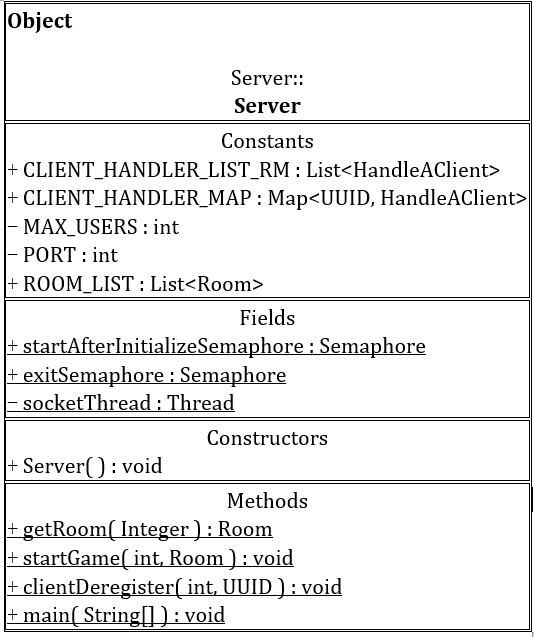
Single Responsibility Principle

The definition of SRP is that: a class should have only one reason to change. We’ve applied SRP to the Server class because it was a large, cumbersome class containing too many methods and attributes. What’s more, the former Server class was designed for handling only 2 players, which is not reasonable.

Therefore, for maintainability and scalability, we’ve separate responsibilities from the former Server class. For example, the duty of sending data to client would be done by HandleAClient class. So, the maintainability increases due to much slimmer Server class acting as a role of a stateless class containing several static methods and some constants. As for the new Room class, it can handle one game of each 2 players, which can be scaled by incrementing the room number. Theoretically, the Server allows multiple clients to connect and play games on, and there is no interference among rooms. These are benefits from applying single responsibility principle.



*After*



*Before*

**Design Pattern**

Mediator Pattern

Mediator pattern is used to reduce communication complexity between multiple objects or classes. For the model-controller, it would be very complicated if every class has a relationship with a client. Therefore, HandleAClient class has become the mediator, transferring messages between the client and the conceptual server (Serer class and Room class). By applying mediator pattern, the cost of communication decreases a lot and the classes are more loosely coupled.

MVC Pattern

As for the model-controller composited by Server, Room, HandleAClient, and Client class, is responsible for sending commands to update the models’ state. The view-controller, that is ClientMain class, can access processed and duplicated data by invoking encapsulated methods from model-controller. But the model used for message passing among classes of model-controller, for instance, InitBean class (used to initialize a client), is not visible to the view-controller.

With Model-view-controller pattern applied, our system is more loosely coupled and we were able to develop different parts simultaneously, which has increased the efficiency of development.

****

Singleton Pattern

The intent of the Singleton pattern is to ensure that a class has only one instance and to provide a global point of access to it. For HandleTheSocket class, who is responsible for handling the socket connection, is considered appropriated to be applied with singleton pattern. By doing so, clients are able to establish connections concurrently without causing any error due to there is only one instance of HandleTheSocket class, which is held by Server class.



**Object Model**

Encapsulation ()

Modularity (MVC)

Hierarchy (DataBean)