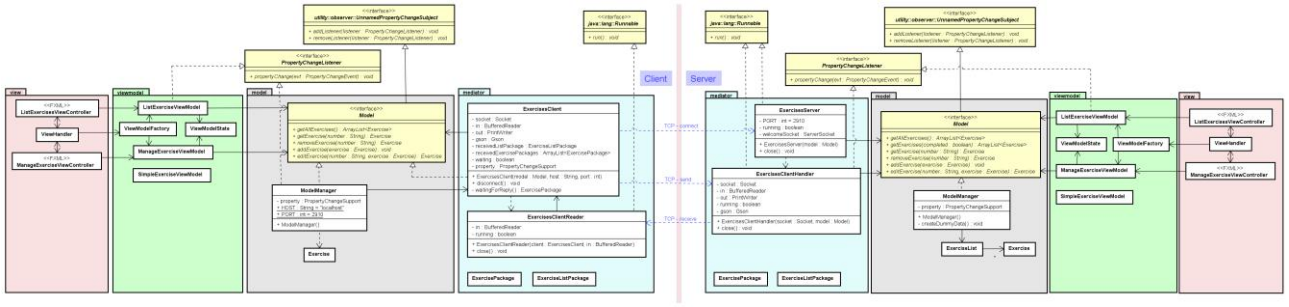
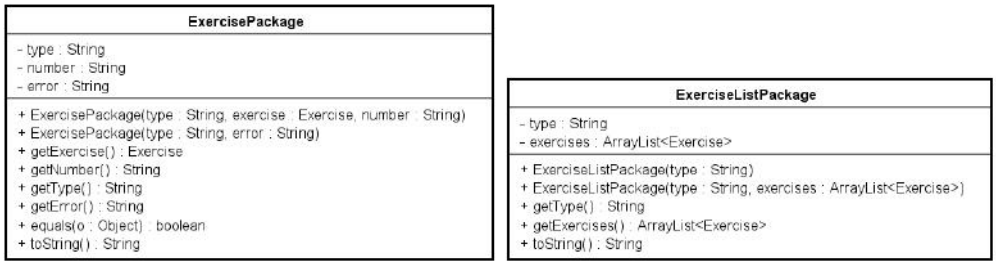
Exercise: MVVM Exercises (in a TCP Client/Server version)

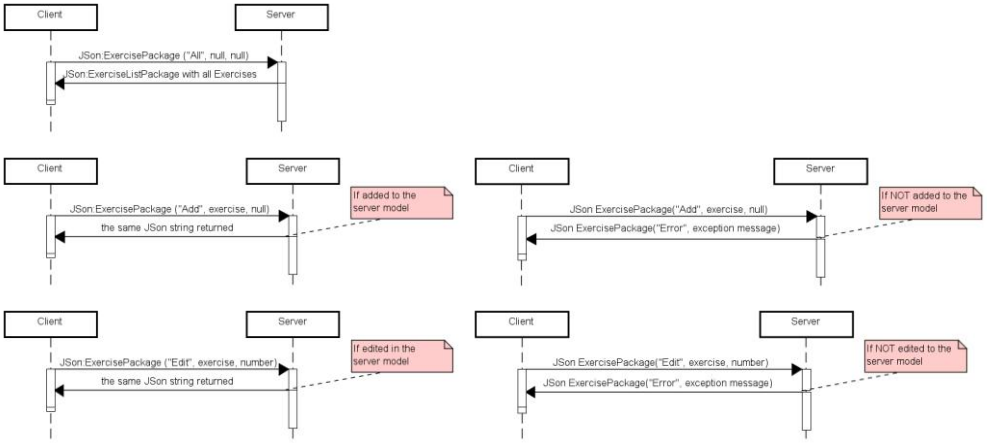


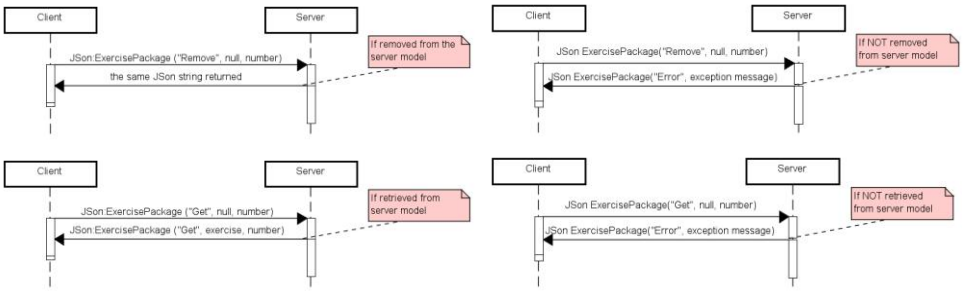
Classes `ExercisePackage` and `ExerciseListPackage` are classes to be used when sending objects (converted to JSon) between client and server.



The communication protocol is the following (see diagrams below): The client sends an ExercisePackage transformed to a JSon string, with type “All”, Add”, “Edit”, “Remove” or “Get”. In case of “Get” also the exercise number, in case of “Add” also an exercise and in case of “Edit” both exercise and exercise number before it has been edited.

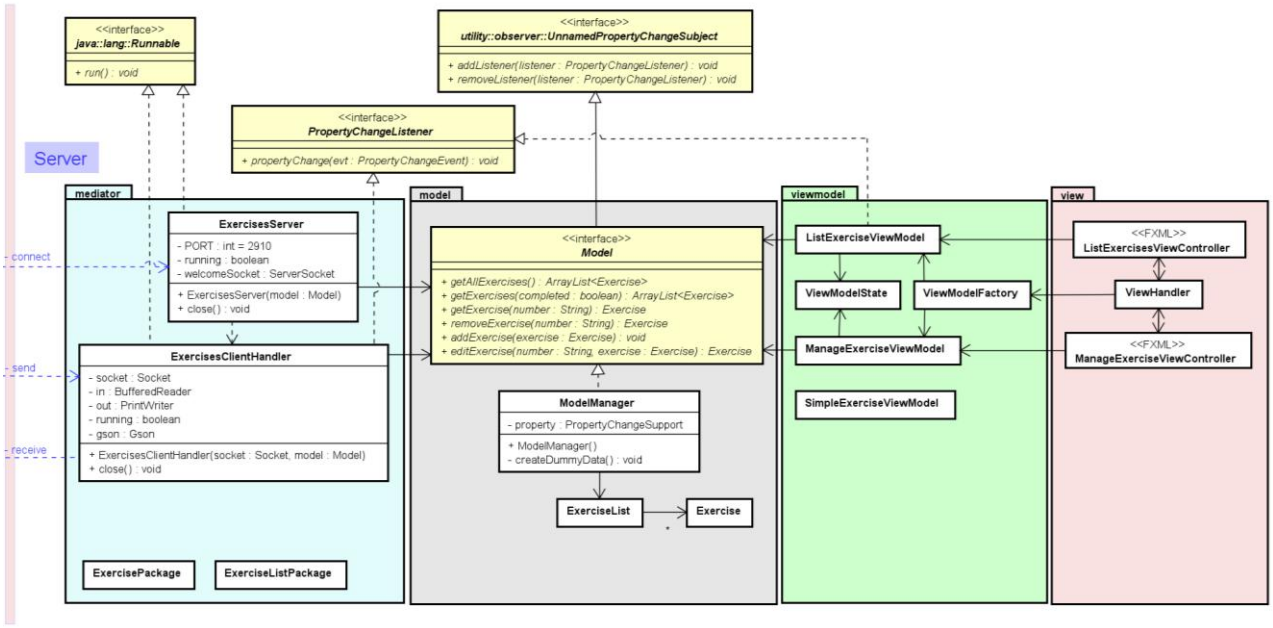
The server replies with a JSon string – if “All” then of type ExerciseListPacket with all exercises, and in all other cases of type ExercisePacket. In case of “Get”, the ExercisePackage contains the exercise retrieved from server model. In case of “Add”, “Edit” and “Remove” with a success, then the same packet as send from client will be send back to the client. In case of an exception from the server model (in “Add”, “Get”, “Edit” and “Remove”), the ExercisePackage contains the type “Error” and the exception message.





Note: "Add" "Edit" and "Remove" (if updated to server model) are being broadcasted to all clients (including the one requesting). This part is not shown in the protocol diagrams above. "Get", "All" and the reply if server model throws an exception are only retrieved by the client sending the request.

Part 1: Server application

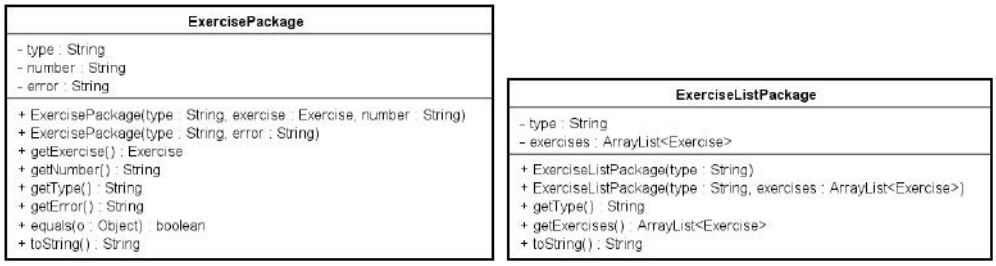


Step 1.1: MVVM

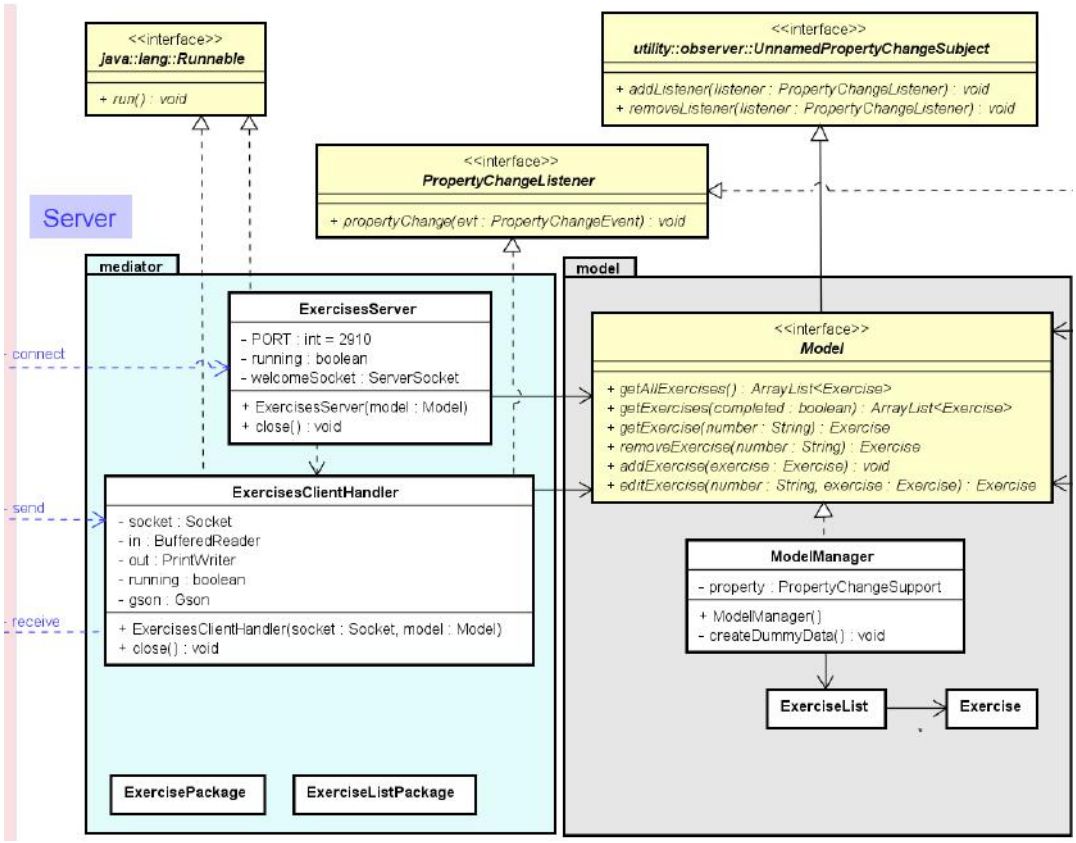
View (red) and ViewModel (green) is exactly as the MVVM single user version you already have. The Model (grey) fires an event when an Exercise has been added, and when an Exercise has been removed.

Step 1.2: Mediator

Include classes `ExercisePackage` and `ExerciseListPackage` for the communication protocol.



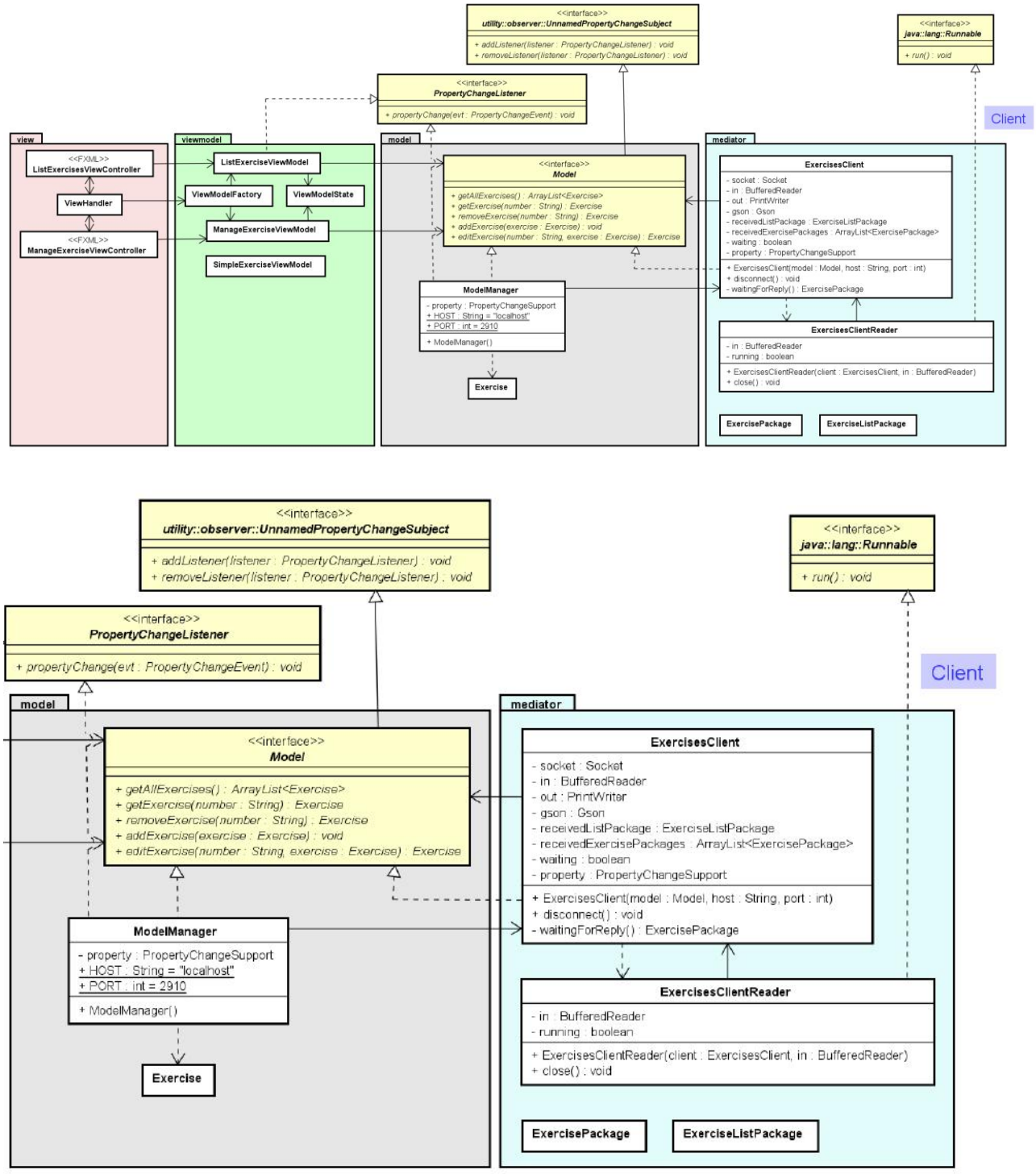
Server mediator and model looks like this:



Classes `ExerciseServer` and `ExercisesClientHandler` represents 1) waiting for a client connection and 2) handling the client communication, respectively. Some remarks:

- They are both Runnable to be used in Threads.
  - In the `run` method for `ExercisesServer` it waits for a connection and creates a thread for the `ExercisesClientHandler`.
  - In the `run` method for `ExercisesClientHandler` it checks the type of the package received, calls the correct method in the model and sends the result back to the client. This is a long method including try-catch blocks, proper calls to model and converting between JSon and `ExercisePackage` and in case of type "All" to `ExerciseListPackage`. In the catch blocks you send the error messages to the client and in case of removing, then also if the Model return `null` indicating that no element has been removed.
- Class `ExercisesClientHandler` is a Listener for the Model and in the `PropertyChange` method it is sending an event as an `ExercisePackage` converted to a JSon string to the client (and because there is one thread connected to each client then this is a broadcast to all clients)
- Class `MyApplication` (not shown in the diagram), creates and starts a server thread. Override the `void` method `stop()` to call `close` for the server (to end the server when the last window closes).
- Make sure to use the `Log` class to log/print when there is any information we would like to see.

Part 2: Client application



Step 2.1: MVVM

View (red) and ViewModel (green) are identical to the ones from the single user application (or the Server). To make it easier to see which window is from Server and which are from clients, you may change the FXML files, e.g. changing the UserData (title of the two windows, either on client or server side).

Model is quite different, because the client model is empty. It is the same `Model` interface, but this time the `ModelManager` only has an association to an `ExerciseClient` from the mediator package. No

`ExerciseList` instance variable (this class do not exist in the client application). *Therefore, you should wait implementing ModelManager until you have implemented the mediator package*

- `ModelManager` is also a listener for the `ExerciseClient` and receiving an event when server broadcasts to clients (e.g. when another client removes an exercise). In the `PropertyChange` method it just fires an event (to be received in the view model)

Step 2.2: Mediator

Classes `ExercisePackage` and `ExerciseListPackage` are the same classes as on the server.

Class `ExerciseClient` is responsible for sending only while `ExerciseClientReader` only receives messages. We don't know if what we receive is a reply for a request (e.g. get exercise with number 1.2 when trying to edit or remove this) or it is a broadcast (e.g. another client just edited another exercise).

Class `ExerciseClientReader` has an input stream (`BufferedReader`) connected to client and in an infinite loop in the `run` method it reads a line from server (a JSon string) and simply call a `receive` method in class `ExerciseClient`.

Class `ExerciseClient`:

- Create the `ExerciseClientReader`, create a Thread and start the thread (in the constructor)
- It has among other variables two instance variables `receivedExercisePackages` and `receivedListPackage` to be used to see if it a reply (having values) or a broadcast (being `null`).
- It implements the `Model` interface and all methods from this interface create the package, convert to Json string, sends to server and wait for the reply. The way it waits is like a monitor class for threads in a while-loop with a `wait()`. The condition for wait is while `receivedExercisePackages` is empty (or in case of "All", `receivedListPackage` is null). These methods need to check if the `error` of the package is not `null` in which case you throw an exception with this error message.
- In the `receive` method (to be called from `ExerciseClientReader`) you check if the type is not "All" which will be treated differently. Could be done like this:
  `if (!gson.fromJson(replyString, Map.class).get("type").equals("All"))`
  Further, if it is waiting for a reply (use the `boolean waiting` before wait and after the wait loop) then add the package in the `receivedExercisePackages` list and call `notify()`. If it is not waiting (it is a broadcast) then fire an event (to be seen by the `ModelManager`). In case of "All" then convert the package, update `receivedListPackage` and call `notify()`.

Final step: Run

Run the server and 2 clients. Test that you can add, edit and remove in one client (or the server) to get updates for the list view for all 3 running applications (clients and server)