Client Server Multi Thread Communication

The purpose of this project is to establish communication through the network between clients and the server for simple Android game.

The project consist of 2 modules: Server and Client Modules. Due to the game simplicity, we do not save the data in the database, but instead keep all the data in the memory. The game itself is very short (approximately one minute), and is deleted from the memory as it's over.

Server
In the server module we have custom Server class (package com.ubongo.server), which extends Thread. As the server runs it creates new thread and tries to open ServerSocket on specified port. If it succeed, we have server listening all the time on that port.
When client tries to connect and succeed, our server opens new Socket, creates new PrintWriter, and BufferReader for that client. The last step is to create ClientTread custom class and run it. So, for each client we have new socket and another thread.

ClientTread class (package com.ubongo.clientTread), which extends Thread, gets and keeps the client PrintWriter and BufferReader. The main purpose of this class is to listen the particular client and get the input. The request input is packed in RequestPackage class, but comes as a JSON object. Here, we use gson library for creating or parsing object to JSON or vice versa.
The last responsibility of this class is to parse the JSON and get the call id, and based on the id call static methods of another singleton class called GameHolder, by checking arguments of the call and passing necessary ones including already kept client output stream.

GameHolder (package com.ubongo.gameHolder) singleton class deals with our game logic and provides 7 methods. It keeps all the games in HashMap using Game class. Game class itself keeps few settings of the game and ArrayList of Player class objects.
Note, this class keeps the clients output stream, so whenever it's necessary it will use and send some message back. Depending on the called method, this class can send response to individual client, or make broadcast to all the players of the particular game. For sending response we use ResponsePackage custom class and convert it to JSON before sending.

To run the server needs only run the class called Main, which creates Server object and starts it.
Note, the tricky part of our server is that it creates new thread as it runs, and another new thread for each client listening on new port. Basically, it can serve as much clients as free ports it has, that is ranging from 0 to 65535.

Note, all the method calls and internal computation logs are provided in console, so you can follow the processes while making test calls. Due to asynchrony calls, it's a bit pain to follow the processes if the logs are not appropriate. So, we tried to do our best to provide all the logs to make it easier to debug and test.

Client side

Client side does not have more than classes for establishing communication, providing 7 methods and getting responses from that methods.

In Congif classs (package com.ubongo.config) the exact port number should be specified of the running server and the server ip address.
NOTE, even if the server runs on 'localhost' and your device is connected to your computer via cable, you cannot specify server ip address as 'localhost'. As you run Android application using your phone device, it's another JVM, even if it's connected with cable.
In manifest, internet permissions should be added to allow your application use internet.
If your android sdk specified in manifest is greater than 9, you need either change it 9 or add additional 2 lines code in your application, that builds strict mode policy and gets permissions.

ServerManager singleton class (package com.ubongo.serverManager) extends AsyncTask to make asynchrony calls. As it runs, it establishes connection with server by opening new Socket, and initializing output and input streams.
While the connection is established, this class listens its input stream. As new package arrives, it checks the id and calls the appropriate method of ResponseManager singleton class for handling the response package.
Each method provided by this class creates specific RequestPackage for that method and sends to the server as a JSON file.

ResponseManager class (package com.ubongo.responseManager) methods purpose is to get the response package and change the state of the game regarding the method called. This insures that game state change logic is separated from server listener and response package analyzer.

In our main activity (package com.ubongo) we have specified about 25 server calls with different arguments: both success and error calls for testing purposes.

Challenges

It's always a bit tricky to work with threads and more importantly to make the threads to work:)
Testing part a bit harder if you usually use to work with RESTfull Api's and make http calls.
Hard to work again with Java after NodeJS, where everything is dynamic.