By design, the specifications of the semester project as well as the support Server code is vague. This document is meant to give you an overview of the server architecture and give an explanation of all important interfaces. Implementation details will not be discussed.

Being forced to code against third party, incomplete, buggy code is unfortunately a common experience that I have faced when writing software in for both commercial and industrial systems. While the server is complete in it’s implementation, it has been tested using minimal cases to play simple games. It will be up to you to work with and modify the server if deemed necessarily.

If any team comes across bugs that break the server and kill the game, it is your responsibility to locate the error, fix the bug, and test the fix. If you have convincing evidence of a bug in the server but cannot find the cause, we can meet to fix the error. Recognizing that a bug exists and fixing the problem privately is unfair and subject to strong penalties. I cannot anticipate how many game breaking bugs there are (hopefully, none!).

Server Architecture

The Bridge game server has been modeled as a system composed of 4 major components:

- **Client/Server RMI Interface (CSI)** - Provides communication between the Bridge game client and server. Specifically, defines all remote methods that the client can invoke. The CSI also manages all Bridge tables. This module sits in the RMI registry to give clients access to the server.

- **Bridge Table (BT)** - Offers a controlled environment for clients to play Bridge. The BT also keeps record of all Bridge Games played on a table, for clients to get game histories.

- **Bridge Game Instance (BG)** - the component that controls individual games of Bridge. Holds all of the game logic and controls required for clients to play a game.

- **Player Database** - Maintains a data structure (specifically, a Hashtable) that keeps track of active player instances. This module is responsible for ensuring all clients are assigned unique id numbers.

This architecture tends itself well to a component-centric design. Note that the CSI is composed of a structure that keeps track of all BT, and the BT is composed of multiple BG’s. This creates a nice compositional hierarchy. Because all components of the server need to query the Player Database, this is left as a static structure in memory.

Server Processing

To get a better idea about how the Server works, here is a high-level flow of what happens as Clients log in.

1. If the client connects successfully, a playerID is created and saved to the database.
2. The client will join an existing table, or create a new table. If a new table is created, the CSI instantiates a new **BridgeTable**. If the client joins an existing table, the interface will lookup the table and then alert the table that a client is sitting at some position.
3. When the table detects that all seats are full, the table will create a new **BridgeTblGame** and set that game as the current active one.
4. Control of each game is controlled by the BG module. When the module determines a game is over, the BG will alert the BT. The BT will save a record of the game, and then tell all players to leave the table.
5. After telling all players to leave the table, the server keeps the table open, available for players to join for a new game.
Server Interfaces

The functionality of each server module is represented by a java Interface. These interfaces are heavily commented. The most interesting and useful interfaces for Bridge client designers are the BridgeServerInterface and BridgePlayer interfaces. All other interfaces are used internally in the server, and clients do not need to be known of their existence. Each interface is explained below:

- **BridgePlayer** - Defines all the methods that the server will call to communicate with a bridge clients. All clients need to implement this interface.
- **BridgeServerInterface** - Defines the communication interface between client and server. Clients will communicate to the server through invoking methods defined by this interface. Almost completely defines the CSI module (it is missing the data structure to manage all BridgeTables, a flaw in the design of this interface).
- **BridgeTable** - The interface that fully defines the BT server module.
- **BridgeTblGame** - The interface that fully defines the BG server module.
- **BridgeHand** - An interface describing the operations that can be performed on a hand in Bridge.

Also of note is an auxillery object **Card**. The Card class is used extensively in the Server design as internal representation of the cards that are used during play. Feel free to use the Card class and its static helper methods as you see fit.