To implement synchronization in our server, we used synchronized methods to ensure that only one thread can access and modify the sketch at a time.

- We marked the getSketch() method as synchronized to ensure that only one thread can access the sketch at a time. This helps prevent race conditions where multiple threads try to read and modify the sketch simultaneously, leading to inconsistent or incorrect results. By synchronizing the method, other threads must wait until the current thread finishes accessing the sketch before they can access it themselves.

- Similarly, we synchronized methods that modify the sketch (addShape(), removeShape(), moveShape(), and recolorShape()). This ensures that only one thread can modify the sketch at a time. Other threads wanting to modify the sketch must wait until the current thread finishes its modifications.

- Additionally, we used synchronization when accessing the nextID variable to ensure that each thread gets a unique ID when adding a shape to the sketch. By synchronizing the getNextIDAndIncrement() method, we guarantee that each thread gets the next available ID in a thread-safe manner.

- Finally, to ensure thread safety when accessing the list of communicators, we synchronized the addCommunicator() and removeCommunicator() methods. This guarantees that only one thread can modify the list of communicators at a time.

On the editor side, we marked methods as synchronized to clarify that their state modification can only happen synchronously for proper behavior since the underlying sketch methods being called are synchronous.

While we didn’t encounter any multi-client/synchronization issues and believe our code should be mostly immune to them, the server could possibly be overloaded when receiving too many messages. Also, the lack of an explicit message queue might cause messages to be processed/broadcast in a non-consecutive manner.