Assignment 3: This Means WAR! (Multiuser game)

Dr. William Kreahling

November 4, 2016

1 Overview

The purpose of this assignment is to give you some experience implementing *concurrent* network applications. After completing this assignment, you will have experience using the Java threading libraries to enable a TCP server to handle more than one client connection simultaneously. Similarly, you will have experience using the same libraries to enable a TCP client to interact with users as well as with a TCP socket.

Your task is to develop two applications: a game server and a game client for multiplayer Battleship. The server will accept incoming TCP connections on a specified port.

2 Communication Protocol

The clients and server applications will communication with each other via TCP. All messages sent to the server begin with a slash The commands that must be supported are:

1. **/join**. The /join command is sent by clients immediately after connecting the the server. The syntax is:

   ```
   /join <username>
   ```

   where username is the user's nick name (multiple users with the same name are not allowed, but should not terminate the TCP connection).

2. **/play**. The /play command is sent by clients to begin a game of Battleship. Play cannot begin if 2 or more users are not "joined". This command is ignored if a game is already in progress.

3. **/attack**. The /attack command is sent by clients during game play to indicate the user to attack, and the location within the board they wish to attack. Players attacking an invalid location or sending this command when a game is not in session or when it is not a client's turn results in the server sending back a private message to the client who sent the command with a informative message.

   ```
   /attack <username> <[0-9]+> <[0-9]+>
   ```

4. **/quit**. The /quit command is sent by clients when they intend to disconnect from the server. This command will cause the server to "clean up" after the client, and to notify all other clients that the user has disconnected, games in progress will continue if there are enough players.

5. **/show**. The /show command is sent by clients when they want to examine the current state of a board.

   ```
   /show <username>
   ```

   This command will cause the server to respond with the game board of the client specified. If the client is the "owner" of the board, then hits, misses, and ships are displayed on the board map. If the client does not own the board, then only hits and missed are displayed. This command is ignored if the game is not in progress, but client's can issue this command when it is not their turn.

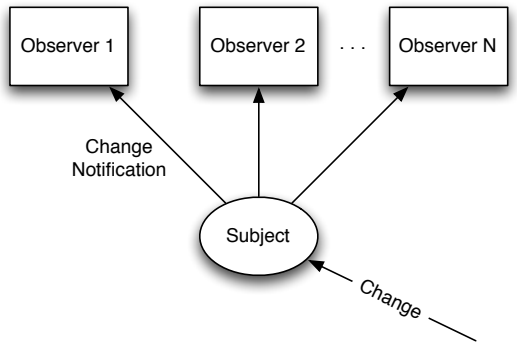

Figure 1: Observer Pattern

3 Design Patterns

In 1977, Christopher Alexander, an architect by trade, developed a language for describing patterns he found in the design of buildings and towns. He said "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice" [1]. These patterns in design are not unique to architecture – such patterns also exist in software systems.

3.1 The Observer Pattern

The intent of the observer pattern is to establish a one-to-many dependency between a set of objects so that when the "subject" changes, the "observers" are notified of the change [2]. Applying such a pattern allows "subjects" and "observers" to be decoupled (i.e., a subject can reference an observer as an observer, and not as its concrete class type).

Figure 1 illustrates the participants in the observer pattern and how data flows between them. First, the Subject has some state in which one more more observers are interested. Whenever a change is made to the state of the subject, the subject notifies all registered observers of the change. Each observer can then independently perform some action in response to the change.

The observer pattern will be used a various points throughout the implementation of both the client and server applications.

4 System Design

To receive full credit on this assignment, your programs must not only function correctly, but should also be designed well. This section describes the **some** structure you should follow to implement your systems.

4.1 Common Classes and Interfaces

The following subsections describe the classes and interfaces in the system that are common to both the client and the server. The implementation of these have been provided for you.

4.1.1 Interface `MessageListener`

MessageListener defines the interface to objects that can *observe* other objects that receive messages. When the subject receives a message, the message is forwarded to all registered observers.

```
package common;
/**
 * This interface represents &quot;observers&quot; of <code>MessageSource</code>s.
 *
 * @author Dr. William Kreahling
 * @version October 2016
 */
public interface MessageListener {
    /**
     * Used to notify observers that the subject has received a message.
     *
     * @param message The message received by the subject
     * @param source  The source from which this message originated (if needed).
     */
    public void messageReceived(String message, MessageSource source);

    /**
     * Used to notify observers that the subject will not receive new messages; observers can
     * deregister themselves.
     *
     * @param source The <code>MessageSource</code> that does not expect more messages.
     */
    public void sourceClosed(MessageSource source);
}
```

4.1.2 Class `MessageSource`

MessageSource is a class that implements the "subject" code necessary to notify "observers" of the receipt of a message. Any class that is capable of receiving a message must extend this class. Upon receipt of a message, that class must invoke the notifyReceipt() method to forward the received message to any registered observers. When a message source is closed (i.e., the listener should expect to receive no more messages from the source), the source should invoke sourceClosed() on the MessageListener.

```
package common;
import java.util.ArrayList;
import java.util.List;

/**
 * This class represents an abstract message source &ndash; a subject in the observer pattern.
 * This class should be extended by classes that want to deliver messages to some set of interested
 * parties.  These &quot;interested parties&quot; are decoupled from the implementation of this
 * class &ndash; those classes must implement the <code>MessageListener</code> interface.
 *
 * @author Dr. William Kreahling
 * @version October 2016
 */
public abstract class MessageSource {
    /** Observers registered to receive notifications about this subject. */
    private List<MessageListener> messageListeners;

    /**
     * Constructs a new <code>MessageSource</code> with no registered observers.
     */
    public MessageSource() {
        this.messageListeners = new ArrayList<MessageListener>();
    }

    /**
     * Adds a new observer to this subject.
```

```
     *
     * @param  listener The new message listener; a new &amp;quot;observer&amp;quot;.
     */
    public void addMessageListener(MessageListener listener) {
        messageListeners.add(listener);
    }

    /**
     * Removes the specified observer from this subject.
     *
     * @param listener The listener to remove.
     */
    public void removeMessageListener(MessageListener listener) {
        messageListeners.remove(listener);
    }

    /**
     * Notifies <b>all</b> registered observers that this message source will generate no new
     * messages.
     */
    protected void closeMessageSource() {
        /*
         * Here we need to iterate over a *copy* of our messageListeners list.  The reason is
         * because if the listener's 'sourceClosed' method removes that listener from this subject,
         * we'd get a ConcurrentModificationException if we were iterating over the original list.
         */
        for (MessageListener listener : new ArrayList<MessageListener>(messageListeners)) {
            try {
                listener.sourceClosed(this);
            } catch (RuntimeException ex) {
                /*
                 * We're doing this on a best-effort basis.  If something goes wrong, we don't want
                 * to stop.  Here, we simply dump the stack and continue.
                 */
                ex.printStackTrace();
            }
        }
        messageListeners.clear();
    }

    /**
     * Notifies <b>all</b> registered listeners that a new message has been received.
     *
     * @param message The message this subject received.
     */
    protected void notifyReceipt(String message) {
        for (MessageListener listener : new ArrayList<MessageListener>(messageListeners)) {
            /*
             * We wrap this in a try/catch block so that just in case one of our observers screws
             * up, we don't want to stop notifying other observers.
             */
            try {
                listener.messageReceived(message, this);
            } catch (RuntimeException ex) {
                /*
                 * We're doing this on a best-effort basis.  If something goes wrong, we don't want
                 * to stop.  Here, we simply dump the stack and continue.
                 */
                ex.printStackTrace();
            }
        }
    }
}
```

4.1.3 Class `ConnectionInterface`

ConnectionInterface is the class responsible for sending messages to and receiving messages from remote hosts. The class extends the MessageSource class, indicating that it can play the role of the "subject" in an instance of the observer pattern. The class also implements the Runnable interface, indicating that it encapsulates the logic associated with a Thread.

4.2 Server Classes and Interfaces

4.2.1 Class `BattleServer`

BattleServer is one of the classes that implement the server-side logic of this client-server application. It is responsible for accepting incoming connections, creating ConnectionInterfaces, and passing the ConnectionInterface off to threads for processing. The class implements the MessageListener interface (i.e., it can "observe" objects that are MessageSources).

4.2.2 Class `BattleShipDriver`

BattleShipDriver contains the main() method for the server. It parses command line options, instantiates a BattleServer, and calls its listen() method. This takes two command line arguments, the port number for the server and the size of the board (if the size is left off, default to size 10 x 10). **You may assume square arrays**.

4.2.3 Class `Game`

Game contains the logic for the game of BattleShip. It has a Grid for each client.

4.2.4 Class `Grid`

Grid is the logic for a single board of Battleship.

4.3 Client Classes and Interfaces

4.3.1 Class `BattleClient`

BattleClient is one of the classes that implement the client-side logic of this client-server application. It is responsible for creating a ConnectionInterface, reading input from the user, and sending that input to the server via the ConnectionInterface. The class implements the MessageListener interface (i.e., it can "observe" objects that are MessageSources). The class also extends MessageSource, indicating that it also plays the role of "subject" in an instance of the observer pattern.

4.3.2 Class `BattleDriver`

BattleDriver contains the main() method for the client. It parses command line options, instantiates a BattleClient, reads messages from the keyboards, and sends them to the client. The command line arguments are: hostname, portnumber and user nickname. All of these command line arguments are required.

5 Details

- You may not use forever loops. There is always a good stopping condition for stopping reading/writing to, or listening to, your sockets.
- Clients disconnecting should not kill the server.
- You must correctly Javadoc all files, fields, and methods for full style credit.
- You must have three packages: server, common, and client. the common package contains **only** those files needed by both the server and the client.
- The program **must** run on agora.
- Programs that do not compile will receive a score of zero. There will be no late turn in for this assignment.
- Both the client and server driver should have correct usage messages EVERY time incorrect command line arguments are supplied.
- Connections to the BattleServer while a game is in progress will be refused.
- All code for the game should exist ONLY on the server. The clients do not keep any information about location, board, turn, etc. The winner of the game is the last person with a ship. There should be no draws.
- **All games should include a way for me to create a board with one or two ships, so I can test win conditions.**

6 Example Run Server

```
java server/BattleShipDriver 9999
```

7 Example Run Client one

```
java client.BattleDriver localhost 9999 mario

!!! mario has joined

/play

Not enough players to play the game

!!! toad has joined

/attack toad 0 0

Game not in progress

!!! luigi has joined

/play

The game begins

mario it is your turn

/attack toad 0 0

Shots Fired at toad by mario
```

```
toad it is your turn

Shots Fired at mario by toad

luigi it is your turn

Shots Fired at toad by luigi

mario it is your turn

/show mario
    0   1   2   3   4   5   6   7   8   9
  +---+---+---+---+---+---+---+---+---+---+
0 | X |   |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+
1 |   |   |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+
2 |   |   |   |   |   |   | B |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+
3 |   |   |   |   |   |   | B |   | D |   |
  +---+---+---+---+---+---+---+---+---+---+
4 |   |   |   |   |   |   | B |   | D |   |
  +---+---+---+---+---+---+---+---+---+---+
5 |   |   |   |   | R |   | B | C |   |   |
  +---+---+---+---+---+---+---+---+---+---+
6 |   |   |   |   | R |   |   | C |   |   |
  +---+---+---+---+---+---+---+---+---+---+
7 |   |   | S |   | R |   |   | C |   |   |
  +---+---+---+---+---+---+---+---+---+---+
8 |   |   | S |   |   |   |   | C |   |   |
  +---+---+---+---+---+---+---+---+---+---+
9 |   |   | S |   |   |   |   | C |   |   |
  +---+---+---+---+---+---+---+---+---+---+

/show toad
    0   1   2   3   4   5   6   7   8   9
  +---+---+---+---+---+---+---+---+---+---+
0 | X | @ |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+
1 |   |   |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+
2 |   |   |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+
3 |   |   |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+
4 |   |   |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+
5 |   |   |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+
6 |   |   |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+
7 |   |   |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+
8 |   |   |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+
9 |   |   |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+

/attack toad 1 1

Shots Fired at toad by mario

toad it is your turn

/show toad
    0   1   2   3   4   5   6   7   8   9
  +---+---+---+---+---+---+---+---+---+---+
0 | X | @ |   |   |   |   |   |   |   |   |
```

```
   +---+---+---+---+---+---+---+---+---+---+
1  |   | @ |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+
2  |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+
3  |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+
4  |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+
5  |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+
6  |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+
7  |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+
8  |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+
9  |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+

!!! toad surrendered

mario it is your turn

/quit
```

8 Example Run Client Two

```
java client.BattleDriver localhost 9999 toad

!!! toad has joined

/show toad

Play not in progress

!!! luigi has joined

The game begins

mario it is your turn

/attack mario 0 0

Move Failed, player turn: mario

Shots Fired at toad by mario

toad it is your turn

/show toad
    0   1   2   3   4   5   6   7   8   9
  +---+---+---+---+---+---+---+---+---+---+
0 | X | R |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+
1 |   | R |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+
2 |   | R |   | C | D |   |   |   | B |   |
  +---+---+---+---+---+---+---+---+---+---+
3 |   |   |   | C | D |   |   |   | B |   |
  +---+---+---+---+---+---+---+---+---+---+
4 |   |   |   | C |   |   |   |   | B |   |
  +---+---+---+---+---+---+---+---+---+---+
5 |   |   |   | C |   |   |   |   | B |   |
  +---+---+---+---+---+---+---+---+---+---+
```

```
6 |   |   |   | C | S | S | S |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+---+
7 |   |   |   |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+---+
8 |   |   |   |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+---+
9 |   |   |   |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+---+

/attack mario 0 0

Shots Fired at mario by toad

luigi it is your turn

Shots Fired at toad by luigi

mario it is your turn

/show toad

    0   1   2   3   4   5   6   7   8   9
  +---+---+---+---+---+---+---+---+---+---+---+
0 | X | @ |   |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+---+
1 |   | R |   |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+---+
2 |   | R |   | C | D |   |   |   | B |   |   |
  +---+---+---+---+---+---+---+---+---+---+---+
3 |   |   |   | C | D |   |   |   | B |   |   |
  +---+---+---+---+---+---+---+---+---+---+---+
4 |   |   |   | C |   |   |   |   | B |   |   |
  +---+---+---+---+---+---+---+---+---+---+---+
5 |   |   |   | C |   |   |   |   | B |   |   |
  +---+---+---+---+---+---+---+---+---+---+---+
6 |   |   |   | C | S | S | S |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+---+
7 |   |   |   |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+---+
8 |   |   |   |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+---+
9 |   |   |   |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+---+

Shots Fired at toad by mario

toad it is your turn

/quit
```

9 Example Run Client Three

```
java client.BattleDriver localhost 9999 luigi

!!! luigi has joined

The game begins

mario it is your turn

/show luigi

    0   1   2   3   4   5   6   7   8   9
  +---+---+---+---+---+---+---+---+---+---+---+
0 |   | C | C | C | C | C |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+---+
```

```
  1 |   |   |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+---+---+
  2 |   |   |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+---+---+
  3 |   | B |   |   |   | D |   |   |   |   |
    +---+---+---+---+---+---+---+---+---+---+
  4 |   | B | S |   |   | D |   |   |   |   |
    +---+---+---+---+---+---+---+---+---+---+
  5 |   | B | S |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+---+---+
  6 |   | B | S |   |   |   |   | R | R | R |
    +---+---+---+---+---+---+---+---+---+---+
  7 |   |   |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+---+---+
  8 |   |   |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+---+---+
  9 |   |   |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+---+---+

Shots Fired at toad by mario

toad it is your turn

Shots Fired at mario by toad

luigi it is your turn

/attack toad 0 1

Shots Fired at toad by luigi

mario it is your turn

Shots Fired at toad by mario

toad it is your turn

!!! toad surrendered

mario it is your turn

!!! mario surrendered

luigi it is your turn

GAME OVER: luigi wins!

/attack mario

Invalid command: /attack mario

/attack mario 0 0

Game not in progress

/quit
```

10 Hand-In Instructions

By November 18, you need to turnin a partially working version of the battleship game. I should be able to view maps for multiple players (both with and without ships), randomly place ships on the board, determine if a move hits a ship. This is a NON-networked version of the game. After you submit this program, you will need to schedule a time (as a team) to come by and demo it. This portion of the program will be assigned a grade. (handin using assignment number 3).

The final multiplayer version is due by 11:59 PM on Friday, December 2nd. Submit only the Java source files. You must submit the assignment using the *handin* command on Agora. (handin using assignment number 4).

Handin works as follows:

```
handin.<course#>.<section#>  <assignment#>  <files>
```

Therefore, to submit this assignment, you must use the following command (assuming each of the files is in your current working directory):

```
handin.465.1  4  *.java
```

11 Notes on Collaboration

You may work is groups of two on this assignment. If you work in groups, you must include both group members' name on the submission. Each group should submit only one copy of the assignment. You must include a text file clearly stating what each team member's role was in this program.

References

[1] Christopher Alexander, Sara Ishikawa, and Murray Silverstein. *A Pattern Language: Towns, Buildings, Construction (Center for Environmental Structure Series)*. Oxford University Press, August 1977.

[2] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.