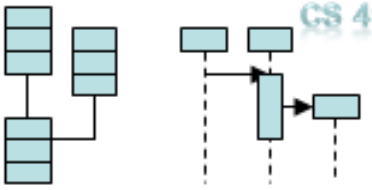

CS 414 Object Oriented Design

Spring 2018
Computer Science Department

Assignment A2

Mocking

DUE: 11:59PM, Wednesday 7 February 2018

20 points

1. Goal

The goal of this assignment is to gain experience using Mockito.

2. Problem

Your good friend and professor, Dr. Chatterbot, is creating an artificial intelligence chatbot. This chatbot accepts strings containing English sentences, and responds with strings that sound as human as possible. He has not yet completed his chatbot code, but he knows he will want to create a chatbot server so various clients can connect and interact with the chatbot.

Your job is to implement the chatbot server while Dr. Chatterbot works on the chatbot code. He wants the server to be robust, so you must write unit tests for it.

Use the attached zip file to work on this assignment. You will find a partial implementation of the `ChatbotServer` class and full source code for the `AIException` class and the `Chatbot` interface. Your tasks are to (1) complete the implementation of `ChatbotServer` and (2) implement a new `TestChatbotServer` using Mockito.

1. `Chatbot.java`

```
package a2;
/**
 *  Single-sentence interface to the Chatbot.  This interface assumes that a
 *  conversation will consist of an alternating sequence of sentences, one
 *  from the user, and one from the bot.
 *
 *  @author Prof. Chatterbot
 */

interface Chatbot {
/**
 * Get a response from the chatbot.
 *
 * @param input The input string.  Must be in English.
 * @return The response from the chatbot, in English.
 *
 * @throws AIException in case the bot goes into an un-recoverable state.
 * If this exception is thrown, the conversation will be automatically
 * restarted from scratch.
 */
 String getResponse(String input) throws AIException;
}
```

2. `AIException.java`

```
package a2;
/**
 *  Exception class to encapsulate exceptions that come from the Chatbot itself.
 *  This is a checked exception.  It will be thrown when the Chatbot AI goes
 *  into a bad state and must be restarted, or when the input is malformed.
 *  This exception requires a message.
 *
 *  @author Prof. Chatterbot
 */
public class AIException extends Exception {
```

```
    public AIException(String message) {
        super(message);
    }
}
```

3. Skeleton of ChatbotServer

```
package a2;
import java.net.ServerSocket;
import java.net.Socket;

/**
 * Allow the Chatbot to be accessible over the network.
 * This class only handles one client at a time.
 * Multiple instances of ChatbotServer will be run on different ports
 * with a port-based load balancer to handle multiple clients.
 *
 * @author <Your Name Here>
 */

public class ChatbotServer {

    /**
     * The instance of the {@link Chatbot}.
     */
    private Chatbot chatbot;

    /**
      * The instance of the {@link ServerSocket}.
      */
    private ServerSocket serversocket;

    /**
     * Constructor for ChatbotServer.
     *
     * @param chatbot The chatbot to use.
     * @param serversocket The pre-configured ServerSocket to use.
     */
    public ChatbotServer(Chatbot chatbot, ServerSocket serversocket) {
        this.chatbot = chatbot;
        this.serversocket = serversocket;
    }

    /**
     * Start the Chatbot server.  Does not return.
     */
    public void startServer() {
        while(true) handleOneClient();
    }

    /**
     * Handle interaction with a single client.  See assignment description.
     */
    public void handleOneClient() {
        // TODO: Your code here.
    }
}
```

The constructor signature for `ChatbotServer` must be what is provided in the skeleton file. The implmentation for `startServer` is also provided. The `startServer()` method starts the chatbot server and waits for clients to connect. After one client disconnects, the server waits for more clients to connect in an infinite loop. Each client is handled in the chatbot server using a method called `handleOneClient()` that you must implement.

The `ChatbotServer` must not throw any exceptions from the constructor, `startServer()`, or `handleOneClient()`. The intention is for the `ChatbotServer` to be instantiated from a main method. However, the main method hasn't been written yet. The main method, if it existed, would create new instances of `Chatbot` and `ServerSocket`, pass them into the constructor of `ChatbotServer`, and then call `startServer()`. You don't need to write the main method, but you do need to write test code that instantiates `ChatbotServer`. The test code will also need to pass new instances of `Chatbot` and `ServerSocket` to the constructor of `ChatbotServer`. For the purposes of this assignment, you do not need to consider multiple simultaneous clients, and thus, no threading is necessary.

The `handleOneClient` method must call `ServerSocket.accept()`, which returns a Socket object for doing the actual network interaction. Then call `getInputStream()` and `getOutputStream()` on the Socket object to get the data streams to use for interaction.

Note that your test code will act as a client of the `ChatbotServer`. The general flow of interactions is as follows:

1. A client connects; the `SocketServer` provides a `Socket` to `ChatbotServer` via the `accept()` method.
2. The client provides a string to the `Socket`.
3. The `ChatbotServer` receives the string, and passes it to the `Chatbot`.
4. The `Chatbot` generates a response and returns it to the `ChatbotServer`.
5. The `ChatbotServer` provides the response to the client via the `Socket`.
6. The client may optionally send more strings and receive more responses, one response per input string.
7. The client disconnects and the SocketServer goes back to waiting for another client.

A note about disconnecting when using Sockets: When the client sends a byte with value -1, that signals the server to disconnect. However, the test code doesn't necessarily need to have a -1 hardcoded in it — `ByteArrayInputStream` automatically sends a -1 when the end of the data is reached. When `BufferedReader.readLine()` receives a -1, it returns `null` to indicate the end of the stream has been reached.

The `ChatbotServer` must gracefully handle exceptions both from the chatbot and from the networking code. If the `Chatbot.getResponse` method throws an `AIException`, the `ChatbotServer` should return the string "Got AIException: <message>" to the client, where "<message>" is the exception message passed to the `AIException` constructor by the `Chatbot` implementation.

If at any time the `ChatbotServer` gets an exception from any of the socket code, then print its stack trace to standard error, and return to `startServer` to wait for a new client connection.

Since we are doing unit testing (and not integration testing), do not use a concrete instance of `Chatbot`, `ServerSocket`, or `Socket`. These must be set up as Mockito mocks within your tests.

Caution: If you're getting unexpected behavior, pay attention to line endings. Some buffer objects buffer until they see a line ending, and sometimes the line endings can change unexpectedly (i.e. DOS vs. Windows vs. Mac) when using Socket.

Important: Make sure your code runs as described above on the linux machines in our department.

---

3. Grading Criteria

Grading will be based on the following criteria:

1. **(10 points)** Your implementations of the `TestChatbotServer` and `ChatbotServer`. For full credit in this part your own tests must pass with your own implementation.
2. **(6 points)** We will execute your `TestChatbotServer` with various faulty implementations of `ChatbotServer` to check if it you can detect those faults. For example, `ChatbotServer` could throw an exception, or it could fail to pass data correctly between the client and the chatbot. In both these cases, your tests must fail.
3. **(4 points)** We will test your `ChatbotServer` with our own TestChatbotServer. Your `ChatbotServer` should pass our test cases.

---

4. Submission

1. Put your name in a comment at the top of `ChatbotServer.java` and `TestChatbotServer.java`.
2. Create a zip file named `a2.zip` containing the two files with no folders.
3. Submit the `a2.zip` in Canvas.

---

5. References

1. Examples of Java networking code
2. https://docs.oracle.com/javase/8/docs/api/java/net/ServerSocket.html
3. https://docs.oracle.com/javase/8/docs/api/java/net/Socket.html