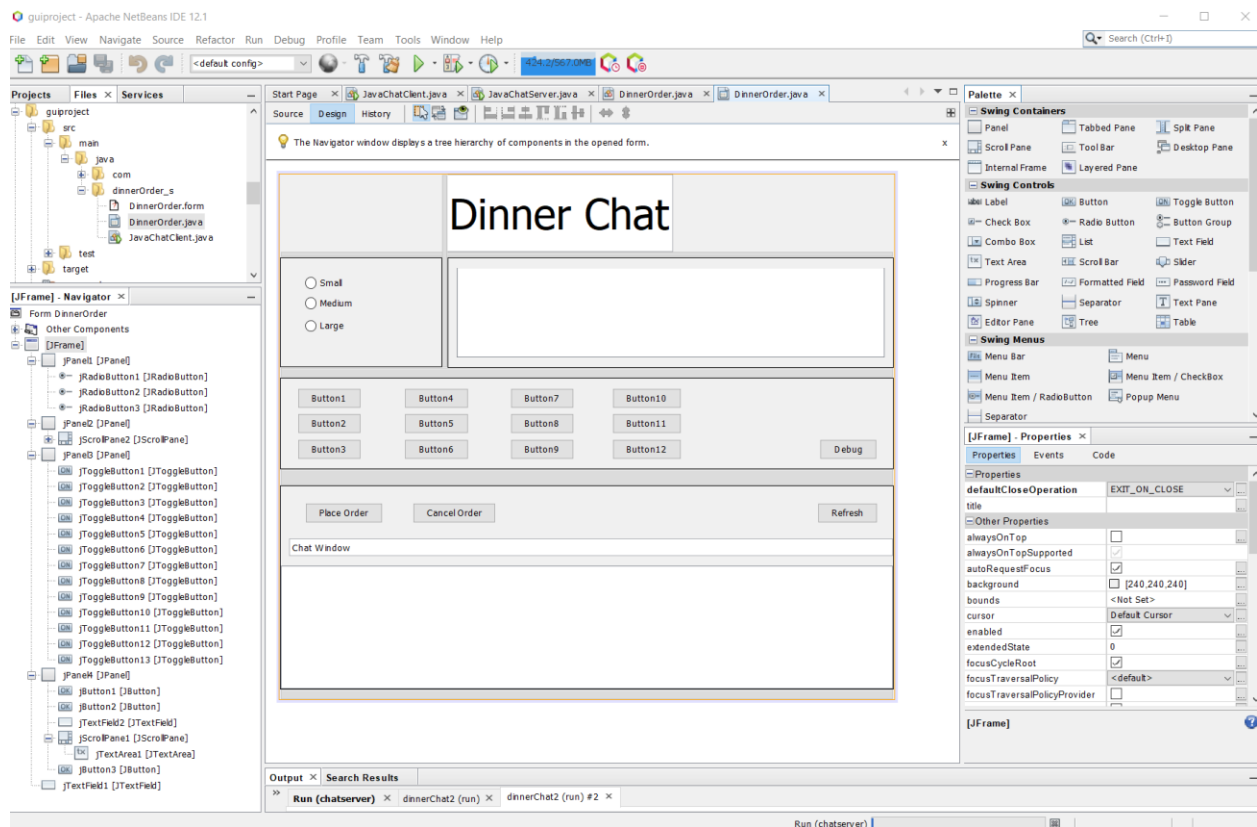


Jamie Shamilian – Dinner Chat Code Review

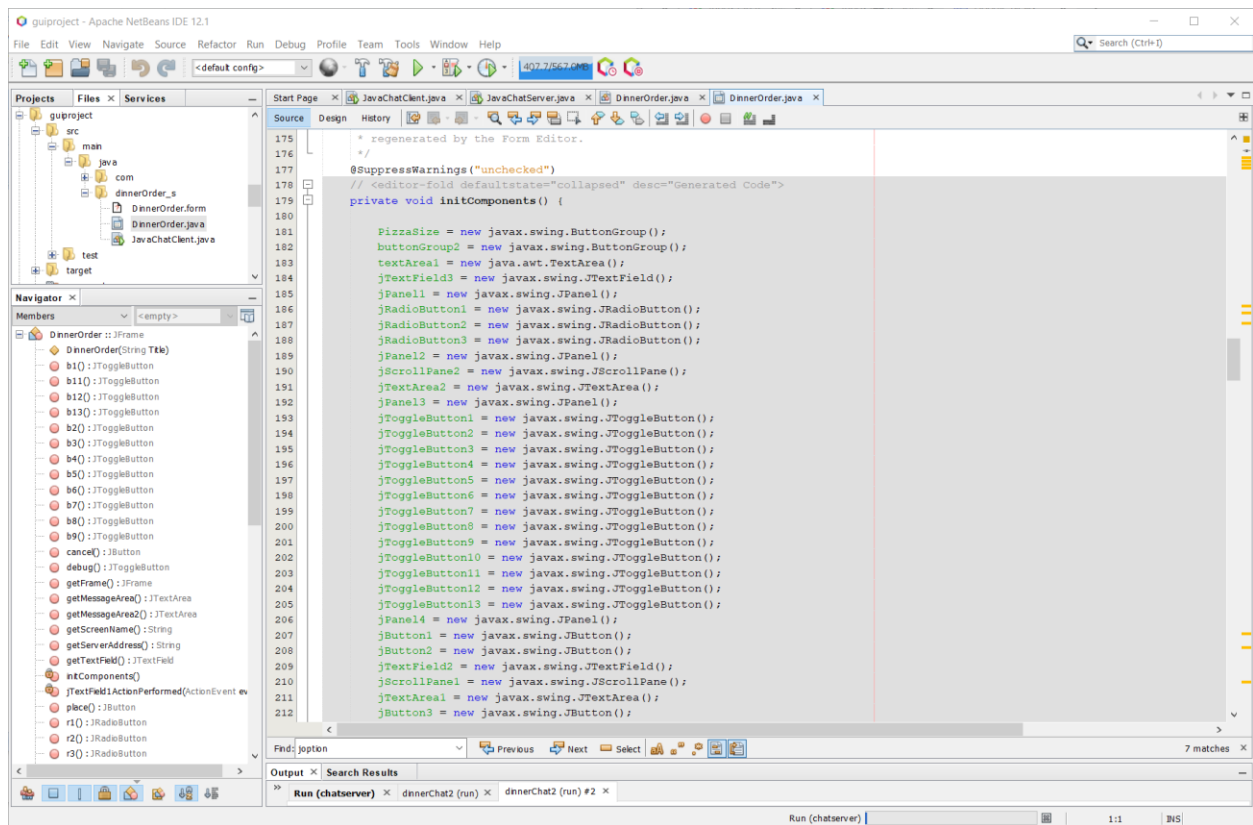
DinnerOrder.java

We use the guibuilder in the Apache Netbeans to create the UI frame.

We select the Design tab and drag components from the right palette to the center design.

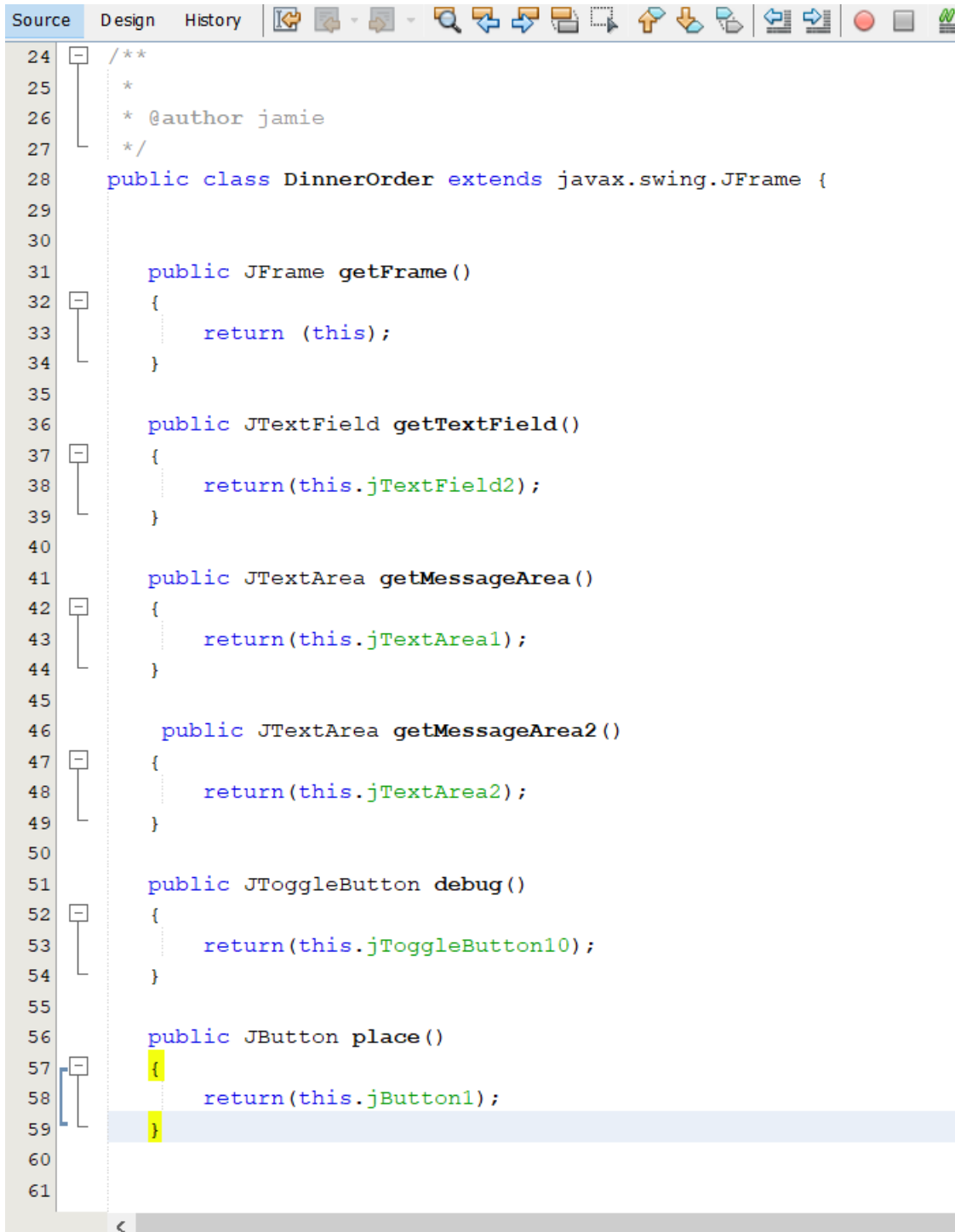


This creates the code to make the above UI we can see when selecting the Source Tab.



We can add our own code to make the private components public.

We can see the public members in the bottom left tile.



The screenshot shows an IDE window with a 'Source' tab selected. The code is for a Java class named `DinnerOrder` that extends `javax.swing.JFrame`. The code includes several getter methods for `JFrame`, `JTextField`, `JTextArea`, `JToggleButton`, and `JButton`. The line numbers 24 through 61 are visible on the left. The code is as follows:

```
24  /**
25   *
26   * @author jamie
27   */
28  public class DinnerOrder extends javax.swing.JFrame {
29
30
31      public JFrame getFrame()
32      {
33          return (this);
34      }
35
36      public JTextField getTextField()
37      {
38          return(this.jTextField2);
39      }
40
41      public JTextArea getMessageArea()
42      {
43          return(this.jTextArea1);
44      }
45
46      public JTextArea getMessageArea2()
47      {
48          return(this.jTextArea2);
49      }
50
51      public JToggleButton debug()
52      {
53          return(this.jToggleButton10);
54      }
55
56      public JButton place()
57      {
58          return(this.jButton1);
59      }
60
61  }
```

Javachatclient

In the java main

We first create the UI with a new DinnerOrder from above.

We then create the client with a new JavaChatClient.

This is followed by initializing the JavaChatClient data from the public members of the DinnerOrder UI.

```
/**
 * @param args the command line arguments
 */
public static void main(String[] args) {

    // create the Java UI frame
    DinnerOrder frame = new DinnerOrder("Chat");
    frame.setVisible(true);

    // create the class JavaChatClient
    JavaChatClient client = new JavaChatClient();

    // JFrame frame;
    client.frame = frame.getFrame();
    client.textField = frame.getTextField();
    client.textField1 = frame.getTextField1();
    client.messageArea = frame.getMessageArea();
    client.messageArea2 = frame.getMessageArea2();

    // create a copy of the radiobuttons
    client.r1 = frame.r1();
    client.r2 = frame.r2();
    client.r3 = frame.r3();

    // create an arraylist of the buttons
    client.buttons.add(frame.b1());
    client.buttons.add(frame.b2());
    client.buttons.add(frame.b3());
    client.buttons.add(frame.b4());
    client.buttons.add(frame.b5());
    client.buttons.add(frame.b6());
    client.buttons.add(frame.b7());
    client.buttons.add(frame.b8());
    client.buttons.add(frame.b9());
    client.buttons.add(frame.b11());
    client.buttons.add(frame.b12());
    client.buttons.add(frame.b13());
```

We create the buttons ArrayList and radios ArrayList with all the buttons and all the radio-buttons.

We call the setup function to setup the client and then we run an infinite run loop.

```
// create an arraylist of the radiobuttons
client.radios.add(client.r1);
client.radios.add(client.r2);
client.radios.add(client.r3);

// copy of generic buttons
client.debug = frame.debug();

client.place = frame.place();
client.cancel = frame.cancel();
client.refresh = frame.refresh();

client.frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
client.frame.setVisible(true);

// call setup method above to setup action listener callbacks
client.setup();

// call the run loop above which has an infinite while loop
// inside of try to catch exceptions from run
try {
    client.run();
} catch (Exception e)
{
}

}
```

Client.setup

This method is where we register all of our actionListeners.

For example when the textField is filled out it calls the actionPerformed method and in this case it gets the text from the textField and writes it to the out stream. It then clears the textField. This is the chat function.

```
public void setup()
{
    // Add All the action Listeners

    // read chat window and send chat message to server
    textField.addActionListener(new ActionListener() {
        /**
         * Responds to pressing the enter key in the textfield by sending
         * the contents of the text field to the server. Then clear
         * the text area in preparation for the next message.
         */
        public void actionPerformed(ActionEvent e) {
            out.println(textField.getText());
            textField.setText("");
        }
    });

    // check radio buttons to verify order and then set order placed info a
    place.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {

            Order = new String("No Size Selected ");

            for ( JRadioButton r : radios ) {
                if ( r.isSelected() ) {
                    Order = ( r.getText() + ", " ) ;
                    break;
                }
            }

            if ( Order.startsWith("No") )
            {
                messageArea2.append (Order+"Please select size\n");
                return;
            }
        }
    });
}
```

Here is a simple cancel function inside setup.

We iterate for all radio buttons stored in radios and set the value to false.

We iterate for all buttons stored in buttons and set the value to false.

Finally we send the status to the server.

We then define the refresh button and it calls sendStatus.

```
// reset all the buttons and sendStatus to server via cancel button
cancel.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Order = "Order Canceled";
        messageArea2.append (Order+"\n");

        for ( JRadioButton r : radios ) {
            r.setSelected(false);
        }

        for ( JToggleButton b : buttons ) {
            b.setSelected(false);
        }

        sendStatus();
    }
});

// sendStatus to server via refresh button
refresh.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        sendStatus();
    }
});
```

We define 2 dialogs.

The first gets a string for the servers IP address.

The second gets a string of the client's name.

```
/**
 * Prompt for and return the address of the server. Dialog
 */
private String getServerAddress() {
    return JOptionPane.showInputDialog(
        frame,
        "Enter IP Address of the Server:",
        "Welcome to Dinner Chat",
        JOptionPane.QUESTION_MESSAGE);
}

/**
 * Prompt for and return the desired screen name. Dialog
 */
private String getName() {
    return JOptionPane.showInputDialog(
        frame,
        "Choose a screen name:",
        "Screen name selection",
        JOptionPane.PLAIN_MESSAGE);
}
```


We define a method to read a menu.json file and present a pull down menu to select the type of dinner we are placing. We create a JSONArray called menulist. We count the number of entries. We create an array of Strings called options and then we call showInputDialog to present the dialog pulldown panel.

```
public String getDinnerType() {

    String[] options = null;
    int menucnt = 0;

    JSONParser parser = new JSONParser();
    Object obj;

    try {

        // load the menu.json file
        obj = parser.parse(new FileReader("Menu.json"));
        JSONObject jsonObject = (JSONObject) obj;
        JSONArray menulist = (JSONArray) jsonObject.get("menulist");

        Iterator<JSONObject> iterator;

        // count the number of menu items
        iterator = menulist.iterator();
        while (iterator.hasNext()) {
            menucnt++;
            //System.out.println(iterator.next());
            iterator.next();
        }

        // setup options for dialog below
        options = new String[menucnt];
        for ( int i = 0; i < menucnt; i++ ) {
            String s = menulist.get(i).toString();
            //System.out.println(s);
            options[i] = s;
        }

    } catch (Exception e) {
        e.printStackTrace();
    }

    String n = (String) JOptionPane.showInputDialog(null, "Which Type of dinner",
        "Pizza", JOptionPane.QUESTION_MESSAGE, null, options, options[0]);

    return(n);
}
```

run is the main loop.

It starts by calling the `getServerAddress` presented earlier to fetch the IP address of the server.

Then we create a socket with the IP address of the server and a port of 9001.

We then create an in stream and an out stream.

We then start the infinite while loop.

We wait to read from the socket (in stream) at the top of the loop.

Depending on the message from the server we perform different actions.

```
private void run() throws IOException {

    // System.out.println(dinnerType);
    // Make connection and initialize streams

    // call getServerAddress above which shows dialog and returns a string of server address
    String serverAddress = getServerAddress();
    // returns a socket from server address and fixed port number 9001
    Socket socket = new Socket(serverAddress, 9001);

    // setup input and output stream of socket to server
    in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    out = new PrintWriter(socket.getOutputStream(), true);

    int lineCnt = 0;
    // Process all messages from server, according to the protocol.
    while (true) {

        //messageArea.append("top input"+lineCnt+"\n");

        // read from socket
        String line = in.readLine();
        lineCnt++;
    }
}
```

For example the first thing the server does after we connect is to send us a "SUBMITNAME" command

And we write our name to the out stream. The server then sends us a NAMEACCEPTED #, where # is the number of the clients so far. If we get a NAMEACCEPTED 1, we call getDinnerType to ask the user the type of dinner as defined earlier. We then set our UI buttons from the json file for that type of dinner.

```
// if server asks from my name
// I call getName and write my name to server
if (line.startsWith("SUBMITNAME")) {
    out.println(getName());
} else if (line.startsWith("NAMEACCEPTED")) {
    // if server says name was accepted
    textField.setEditable(true);

    // if server says name was accepted and
    // I am the first then call getDinnerType to choose f
    // and setButtons from json File
    if ( line.startsWith("NAMEACCEPTED 1 ") )

        if (dinnerType == null ){

            dinnerType = getDinnerType();

            setButtonsFile();
        }
    }
}
```

After a valid connection the server sends a "SYNC: " message to the clients.

The clients then send their current status with includes the type and button settings.

```
} else if (line.startsWith("SYNC:")) {
    // if server ask me to send SYNC I call sendStatus
    messageArea2.append (line.substring(6) + " just joined the Chat\n");
    sendStatus();
}
```

If we get a "SIZE:" message we set the radio buttons.

If we get a "STUFF:" message we set the buttons.

If we get a "TYPE" message we set the dinnerType.

If we get a "MESSAGE:" message we set the messageArea with the text.

```
    } else if (line.startsWith("SIZE: ")) {
        // if server sends Size message set/ reset  radio buttons

        // check which radio button was sent by server
        for ( JRadioButton r : radios ) {
            if ( line.startsWith(r.getText(),8) ) {
                r0 = r;
                break;
            }
        }

        // set / reset radio if it was sent with a +
        if (line.startsWith("+",7))
            r0.setSelected(true);
        else
            r0.setSelected(false);

    } else if (line.startsWith("STUFF: ")) {
        // if server sends STUFF message set / reset buttons

        for ( JToggleButton b : buttons ) {
            // String s = b.getText().substring(0,4);
            String s = b.getText();
            if ( line.startsWith(s ,8) )
            {
                b0 = b;
                break;
            }
        }

        // set / reset button if sent with a +
        if (line.startsWith("+",7))
            b0.setSelected(true);
        else
            b0.setSelected(false);
    }
```

JavaChatServer

We start by creating a listen tcp socket at port 9001.

We then run an infinite loop to accept the socket from the listen and call a new “Handler” thread.

```
public static void main(String[] args) {  
  
    System.out.println("The chat server is running.");  
  
    // create a listen socket and loop forever accepting new clients  
    try {  
        ServerSocket listener = new ServerSocket(PORT);  
  
        while (true) {  
            new Handler( listener.accept()).start();  
            numClients++;  
        }  
  
        } catch (Exception e) {  
            System.out.println("IOException occurred");  
        }  
    }  
}
```

The Handler thread

Sets the name of the client, socket, in stream, outstream.

```
/**  
 * A handler thread class.  Handlers are spawned from the listening  
 * loop and are responsible for a dealing with a single client  
 * and broadcasting its messages.  
 */  
private static class Handler extends Thread {  
    private String name;  
    private Socket socket;  
    private BufferedReader in;  
    private PrintWriter out;
```

The run routine of the Handler thread.

Starts by sending the "SUBMITNAME" request. To the out stream.

And reads the in stream for the name.

It then writes the name into the names hash if it does not already exist.

Otherwise it loops and sends SUBMITNAME again.....

```
while (true) {
    out.println("SUBMITNAME");
    name = in.readLine();
    if (name == null) {
        return;
    }
    // make sure only one thread acts at a time to the names hash set
    synchronized (names) {
        if (!names.contains(name)) {
            names.add(name);
            break;
        }
    }
}
```

We then send "NAMEACCEPTED #" to the out stream.

We then send a "SYNC:" to the other out streams of the other clients.(writers)

We add the outstream to the writers.

```
// send nameaccepted to the client and number of client
// ( 1 indicates first client and forces client to choose dinnerType)
out.println("NAMEACCEPTED "+(writers.size()+1)+" ");

// Send SYNC Message to clients to force sync
for (PrintWriter writer : writers) {

    System.out.println("Sync writers "+(writers.size()+1));
    writer.println("SYNC: "+name);
    writer.flush();
    // break;
}

writers.add(out);
```

We then start our infinite loop for this thread/client.

```
while (true) {  
    String input = in.readLine();  
    if (input == null) {  
        return;  
    }  
}
```

If input starts with \$ then we write TYPE dinnerType to all out streams (writers)

If input starts with + then we write STUFF: + text to all out streams (writers)

If input starts with – then we write STUFF: - text to all out streams (writers)

If input starts with ^ then we write SIZE: +/- text to all out streams (writers)

Otherwise we send MESSAGE name text to all out streams (writers)

If the socket closes then we remove the name from names and

We remove the out from writers and exit the thread.