University of New Brunswick

Faculty of Computer Science

Course: CS2043 – Software Engineering I Deliverable #: _____6_____

Instructor: Natalie Webber Date: _____2017-04-02_____

Project group members:

Student #1: ___3413735_____ / _____Andrew Hampton_____

Student Number / Student Name

Student #2: _____ 3516474_____ / _____ Shane Pelletier _____

Student Number / Student Name

Table of Contents

Contents

Code

BGSetup Parser class

```
import java.util.ArrayList;


public class BGSetupParser {
  public static Ship parseMessage(String message) {
    ArrayList<String> parts = new ArrayList<String>();
    String[] tokens = message.split(" ");


    if (tokens[0].equals("AC")) {
      if (tokens.length != 6) {
        return null;
      } else {
        for (int i = 1; i < tokens.length; i++) {
          parts.add(tokens[i]);
        }
      }
    } else if (tokens[0].equals("CR")) {
      if (tokens.length != 5) {
        return null;
      } else {
        for (int i = 1; i < tokens.length; i++) {
          parts.add(tokens[i]);
        }
      }
    } else if (tokens[0].equals("SB")) {
      if (tokens.length != 4) {
```

```
        return null;
      } else {
        for (int i = 1; i < tokens.length; i++) {
          parts.add(tokens[i]);
        }
      }
    } else if (tokens[0].equals("FR")) {
      if (tokens.length != 3) {
        return null;
      } else {
        for (int i = 1; i < tokens.length; i++) {
          parts.add(tokens[i]);
        }
      }
    } else {
      return null;
    }

    Ship temp = new Ship(tokens[0], parts);
    if (temp.isValid()) {
      return temp;
    } else {
      return null;
    }
  }
}
```

Client class

```
import java.io.*;
import java.net.*;

public class Client
{
  private String host;
  private int port;
  private ServerSocket socket;
  private Socket s;
  private PrintWriter out;
  private BufferedReader in;
  private BufferedReader stdln;
  private SetupGUI setupGUI;

  public Client(String host, int port)
  {
    this.host = host;
    this.port = port;
     s = null;


   try
   {
      s = new Socket(host, port);
   }
   catch (UnknownHostException e)
```

```
 {
  System.err.println("Unknown host: " + host);
  System.exit(-1);
 }
 catch (IOException e)
 {
  System.err.println("Unable to get I/O connection to: "
                + host + " on port: " + port);
  System.exit(-1);
 }

        setupGUI = new SetupGUI();

        openStream();
        messageToServer();

}
public void openStream()
{
 try
 {
  out = new PrintWriter(s.getOutputStream(), true);
  in = new BufferedReader(new InputStreamReader(s.getInputStream()));
  stdln = new BufferedReader(new InputStreamReader(System.in));
 }
 catch(IOException e)
 {
    System.out.println("Problem reading or writing:" + e.getMessage());
 }
```

```
}
public void messageToServer()
{
   boolean done = false;
   try{
   while(!done )
   {
    String line = in.readLine();
    if(line == null)
    {
     done = true;
                    closeStream();
    }
    else
    {
     out.println(line);
    }
   }
 }
 catch(IOException e)
 {
  System.out.println("Problem reading or writing:" + e.getMessage());
 }
}
public void closeStream()
{

 try
 {
```

```
        in.close();
        out.close();
        stdln.close();
        s.close();

    }
    catch(IOException e)
    {
     System.out.println("Problem with closing the reader, writer or socket");
    }



  }
            public void paintComponent()
            {
                        SetupGUI setupGUI = new SetupGUI();
                        setupGUI.createComponents();
            }


}
```

Game Board class

```
import java.util.ArrayList;
import java.util.Iterator;

public class GameBoard {
  private ArrayList<Ship> playerOneShips;
  private ArrayList<Ship> playerTwoShips;

  public GameBoard() {
            playerOneShips = new ArrayList<Ship>();
            playerTwoShips = new ArrayList<Ship>();
  }

  public Boolean addShip(int playerNumber, Ship ship) {
    System.out.println(ship);
    if (ship == null) {
      return false;
    } else {
      if (playerNumber == 1) {
        return playerOneShips.add(ship);
      } else if (playerNumber == 2) {
        return playerTwoShips.add(ship);
      } else {
        return false;
      }
    }
  }
```

```
public Boolean removeShip(int playerNumber, Ship ship) {
  if (playerNumber == 1) {
    return playerOneShips.remove(ship);
  } else if (playerNumber == 2) {
    return playerTwoShips.remove(ship);
  } else {
    return false;
  }
}

public String attack(int playerNumber, String location) {
  char row = location.charAt(0);
  char col = location.charAt(1);
  char thrdDig = '\0';
  if (location.length() > 2) {
    thrdDig = location.charAt(2);
  }
  if (row != 'A' && row != 'B' && row != 'C' && row != 'D' && row != 'E' &&
      row != 'F' && row != 'G' && row != 'H' && row != 'I' && row != 'J') {
    return "err";
  }

  if (col != '1' && col != '2' && col != '3' && col != '4' && col != '5' &&
      col != '6' && col != '7' && col != '8' && col != '9' && thrdDig != '0') {
    return "err";
  }
  Boolean miss = false;
```

```
for (int i = 0; i < playerOneShips.size(); i++) {
  System.out.println("Player one ships: " + playerOneShips.get(i));
}

for (int i = 0; i < playerTwoShips.size(); i++) {
  System.out.println("Player two ships: " + playerTwoShips.get(i));
}

if (playerNumber == 1) {
  for (Iterator<Ship> iterator = playerOneShips.iterator(); iterator.hasNext(); ) {
    Ship playerOneShip = iterator.next();
    if (playerOneShip.partAt(location)) {
      playerOneShip.removePart(location);
      if (playerOneShip.getSize() == 0) {
        String temp = "" + playerOneShip.toString().charAt(0) +
                      playerOneShip.toString().charAt(1) +
                      playerOneShip.toString().charAt(2);
        iterator.remove();
        if (playerOneShips.isEmpty()) {
          temp = playerOneShip.toString().trim();
          return "sunk " + temp + ",win";
        }
        return "sunk " + temp;
      } else {
        return "hit";
      }
    }
    return "miss";
  }
```

```
    } else if (playerNumber == 2) {
      for (Iterator<Ship> iterator = playerTwoShips.iterator(); iterator.hasNext(); ) {
        Ship playerTwoShip = iterator.next();
        if (playerTwoShip.partAt(location)) {
          playerTwoShip.removePart(location);
          if (playerTwoShip.getSize() == 0) {
            String temp = "" + playerTwoShip.toString().charAt(0) +
                        playerTwoShip.toString().charAt(1) +
                        playerTwoShip.toString().charAt(2);
            iterator.remove();
            if (playerTwoShips.isEmpty()) {
              temp = playerTwoShip.toString().trim();
              return "sunk " + temp + ",win";
            }
            return "sunk " + temp;
          } else {
            return "hit";
          }
        }
        return "miss";
      }
    }
    return "err";
  }
}
```

GameManager class

```
import java.util.ArrayList;


public class GameManager {
  private static Boolean isInSignin;
  private static Boolean isInSetup;
  private static GameBoard board;
  private static Server server;
  private static ArrayList<User> users;
  private static Client client;
  private static MainMenu mainMenu;



  public static void main(String[] args) {
   // Port defaults to 31415 if a port isn't passed in on the command line


  // TODO: main method
   isInSignin = true;
   isInSetup = true;
   board = new GameBoard();
   users = new ArrayList<User>();
                        server = new Server(args.length != 0 ? args[0] : "31415");


  }


  public static String receiveMessage(int playerNumber, String message) {
   if (isInSignin == true) {
     String[] signinMessage = message.split(",");
```

```
  if (signinMessage[0].equals("N")) {
    Boolean found = false;
    for (int i = 0; i < users.size(); i++) {
      if (users.get(i).getName().equals(signinMessage[1])) {
        found = true;
      }
    }
    if (found) {
      return "err,Duplicate User Name";
    } else {
      users.add(new User(signinMessage[1]));
      return "ack";
    }
  } else {
    Boolean found = false;
    for (int i = 0; i < users.size(); i++) {
      if (users.get(i).getName().equals(signinMessage[1])) {
        found = true;
      }
    }
    if (!found) {
      return "err,Unknown User";
    } else {
      return "ack";
    }
  }
}
else if (isInSetup == true) {
  String[] unparsedShips = message.split(",");
```

```
  boolean success = true;

  for (int i = 0; i < unparsedShips.length; i++) {
    System.out.println("unparsedShips[" + i + "] : " + unparsedShips[i]);
    if (!board.addShip(playerNumber,
                BGSetupParser.parseMessage(unparsedShips[i]))) {
                  success = false;
      break;
    }

  }

  if (success) {
     return playerNumber + ":" + "ack," + playerNumber;
  } else {
     return playerNumber + ":" + "err";
  }

 } else {
  String temp = board.attack((playerNumber == 1 ? 2 : 1), message);
  System.out.println("GameManager: " + temp);
  if (temp == "err") {
    return playerNumber + ":" + temp;
  }
  String[] result = temp.split(",");
  if (result.length > 1 && result[1].equals("win")) {
    System.out.println((playerNumber == 1 ? 2 : 1) + ":" + message + ','
                                    + result[0] + ",loss");
```

```
      return (playerNumber == 1 ? 2 : 1) + ":" + message + ','
                                  + result[0] + ",loss";
    }
    System.out.println("GM: " + (playerNumber == 1 ? 2 : 1) + ":" + message + ',' + result[0]);
    return (playerNumber == 1 ? 2 : 1) + ":" + message + ',' + result[0];
   }
  }

  public static void setIsInSetup(Boolean _isInSetup) {
   isInSetup = _isInSetup;
  }

  public static void setIsInSignin(Boolean _isInSignin) {
   isInSignin = _isInSignin;
  }
 }
```

GamePlayGUI class

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.Toolkit;

public class GamePlayGUI extends JFrame
{
  private static JFrame frame;
  private static Toolkit toolkit;
  private static JButton[][] attackLocation;
  private static JButton[][] shipLocation;
  private static JButton exitButton;
  private static JLabel instructions;
  private static ActionListener listener;

  public GamePlayGUI()
  {
    frame  = new JFrame();
    frame.setSize(800,800);
    frame.setTitle("BattleShip");
    frame.setExtendedState(JFrame.MAXIMIZED_BOTH);
    frame.setLayout(new GridLayout(3, 1));
    frame.setVisible(true);
    toolkit = Toolkit.getDefaultToolkit();

    listener = new ActionListener() {
```

```
    @Override
    public void actionPerformed(ActionEvent e) {
      if (e.getSource() instanceof JButton) {
        String text = ((JButton)e.getSource()).getText();
        JOptionPane.showMessageDialog(null, text);
      }
    }
  };

  shipLocation = new JButton[10][10];
  attackLocation = new JButton[10][10];
  for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 10; j++) {
      attackLocation[i][j] = new JButton((char)(i + 65) + "" + (j + 1));
      shipLocation[i][j] = new JButton((char)(i + 65) + "" + (j + 1));
      attackLocation[i][j].addActionListener(listener);
    }
  }
}

public static void createComponents()
{
  instructions = new JLabel("Please place your Aircraft Carrier", SwingConstants.CENTER);
  instructions.setPreferredSize(new Dimension(120, 20));

  exitButton = new JButton("Exit");

  JPanel attackButtonPanel = new JPanel(new GridLayout(10, 10));
  for (int i = 0; i < 10; i++) {
```

```
    for (int j = 0; j < 10; j++) {
      attackButtonPanel.add(attackLocation[i][j]);
    }
  }

  JPanel shipButtonPanel = new JPanel(new GridLayout(10, 10));
  for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 10; j++) {
      shipButtonPanel.add(shipLocation[i][j]);
    }
  }

  JPanel topPanel = new JPanel();
  topPanel.add(shipButtonPanel);

  JPanel bottomPanel = new JPanel();
  bottomPanel.add(exitButton);
  bottomPanel.add(attackButtonPanel);

  frame.getContentPane().add(topPanel);
  frame.getContentPane().add(instructions);
  frame.getContentPane().add(bottomPanel);
}

public static void main(String[] args)
{
  GamePlayGUI gamePlayGUI = new GamePlayGUI();
  gamePlayGUI.createComponents();
  gamePlayGUI.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
 }
}
```

SetupGUI class

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.Toolkit;
import java.util.ArrayList;

public class SetupGUI extends JFrame
{
 private JFrame frame;
 private Toolkit toolkit;
 private JButton[][] location;
 private JButton exitButton;
 private JLabel instructions;
 private ActionListener listener;
 private String shipToBePlaced;
 private ArrayList<String> parts;
 private ArrayList<ArrayList<String>> ships;

 private ImageIcon backgroundImage;

 public SetupGUI()
 {
  frame  = new JFrame();
  frame.setSize(800,800);
  frame.setTitle("Setup");
```

```
frame.setExtendedState(JFrame.MAXIMIZED_BOTH);
frame.setLayout(new GridLayout(2, 1));
frame.setVisible(true);
toolkit = Toolkit.getDefaultToolkit();
parts = new ArrayList<String>();
ships = new ArrayList<ArrayList<String>>();
shipToBePlaced = "AC";

listener = new ActionListener() {
  @Override
  public void actionPerformed(ActionEvent e) {
    if (e.getSource() instanceof JButton) {
      String text = ((JButton)e.getSource()).getText();
      if (shipToBePlaced.equals("AC")) {
        System.out.print(text + " ");
        parts.add(text);
        if (parts.size() == 5) {
          System.out.println();
          shipToBePlaced = "CR";
          ships.add(parts);
          parts.clear();
        }
      } else if (shipToBePlaced.equals("CR")) {
        parts.add(text);
        System.out.print(text + " ");
        if (parts.size() == 4) {
          System.out.println();
          shipToBePlaced = "SB";
          ships.add(parts);
```

```
            parts.clear();
          }
        } else if (shipToBePlaced.equals("SB")) {
          parts.add(text);
          System.out.print(text + " ");
          if (parts.size() == 3) {
            System.out.println();
            shipToBePlaced = "FR";
            ships.add(parts);
            parts.clear();
          }
        } else if (shipToBePlaced.equals("FR")) {
          parts.add(text);
          System.out.print(text + " ");
          if (parts.size() == 2) {
            System.out.println();
            ships.add(parts);

            // Call server with placed ships.
          }
        }
      }
    }
  };
          createComponents();


}
```

```
public void createComponents()
{
        JPanel backgroundPanel = new JPanel();

        backgroundPanel.setPreferredSize(new Dimension(800, 800));
         Image image = toolkit.getImage("Ocean.png");

        Image scaledImage = image.getScaledInstance(800, 800, Image.SCALE_DEFAULT);
 backgroundImage = new ImageIcon(scaledImage);
 JLabel backgroundLabel = new JLabel(backgroundImage);
         backgroundPanel.add(backgroundLabel);

 instructions = new JLabel("Please place your Aircraft Carrier", SwingConstants.CENTER);
 instructions.setPreferredSize(new Dimension(120, 20));

 exitButton = new JButton("Exit");
        exitButton.addActionListener(new Exit());

        location = new JButton[10][10];
 for (int i = 0; i < 10; i++) {
  for (int j = 0; j < 10; j++) {
    location[i][j] = new JButton((char)(i + 65) + "" + (j + 1));
    location[i][j].addActionListener(listener);
  }


       }

 JPanel buttonPanel = new JPanel(new GridLayout(10, 10));
```

```
  for (int i = 0; i < 10; i++) {
   for (int j = 0; j < 10; j++) {
     buttonPanel.add(location[i][j]);
   }
  }

  JPanel bottomPanel = new JPanel();
  bottomPanel.add(exitButton);
  bottomPanel.add(buttonPanel);
          frame.getContentPane().add(BorderLayout.NORTH, backgroundPanel);
  frame.getContentPane().add(instructions);
  frame.getContentPane().add(bottomPanel);
 }
  private class Exit implements ActionListener
    {
     @Override
     public void actionPerformed(ActionEvent e) {

       System.exit(0);

    }

    }



}
```

MainMenu class

```
import javax.swing.*;

import java.awt.*;

import java.awt.event.*;

import java.awt.Toolkit;

import javax.swing.JOptionPane;

import java.awt.Graphics;


public class MainMenu extends JFrame

{

 private JPanel panel;

 private JPanel labels;

 private JButton connectButton;

 private JButton exitButton;

 private ImageIcon backgroundImage;


 private JFrame frame;

 private Toolkit toolkit;

 private String connectPopUp;


 private String user;

 private int port;

 private JTextField userIP;

 private JTextField userPort;

 private String userString;

 private String portString;

 private String dummyUser = "localhost";
```

```
private String dummyPort = "31415";
private Client client;
SetupGUI setupGUI;

public MainMenu()
{

  frame  = new JFrame();

  panel = new JPanel(new BorderLayout(5,5));
  labels = new JPanel(new GridLayout(0,1,2,2));
  frame.setSize(800,800);
  frame.setTitle("BattleShip");
  frame.setVisible(true);
  toolkit = Toolkit.getDefaultToolkit();



}

public void createComponents()
{
  // Creates a panel for the background picture
  JPanel backgroundPanel = new JPanel();
  //backgroundPanel.setLayout(new BorderLayout())
  backgroundPanel.setPreferredSize(new Dimension(800, 800));
 Image image = toolkit.getImage("Ocean.png");

  Image scaledImage = image.getScaledInstance(800, 800, Image.SCALE_DEFAULT);
  backgroundImage = new ImageIcon(scaledImage);
```

```
 JLabel backgroundLabel = new JLabel(backgroundImage);

 //Creates the panel for the buttons, it will be placed on top of the backgroundPanel
 JPanel buttonPanel = new JPanel();
 buttonPanel.setPreferredSize(new Dimension(100,100));
 connectButton = new JButton("Connect");
 exitButton = new JButton("Exit");
 exitButton.addActionListener(new Exit());
 connectButton.addActionListener(new ConnectButton());

 buttonPanel.add(connectButton);
 buttonPanel.add(exitButton);

 backgroundPanel.add(backgroundLabel);
 // backgroundPanel.add(connectButton);
 // backgroundPanel.add(exitButton);
 backgroundPanel.add(buttonPanel);
 frame.getContentPane().add(BorderLayout.NORTH, backgroundPanel);
 frame.getContentPane().add(BorderLayout.SOUTH, buttonPanel);
 //frame.getContentPane().add(BorderLayout.CENTER, backgroundPanel);
  LogIn(frame);
 }
private void LogIn(JFrame frame)
{


 labels.add(new JLabel("Enter IP", SwingConstants.RIGHT));
 labels.add(new JLabel("Enter Port", SwingConstants.RIGHT));
 panel.add(labels, BorderLayout.WEST);
```

```
 JPanel controls = new JPanel(new GridLayout(0,1,2,2));
  userIP = new JTextField();
  controls.add(userIP);
  userPort = new JTextField();
 userPort.addActionListener(new ConnectButton());
 controls.add(userPort);
 panel.add(controls, BorderLayout.CENTER);



}
public void checkLogin()
{
        if(userString.equals(dummyUser) && portString.equals(dummyPort))
                {
                        int portNumber = Integer.parseInt(portString);

                        //client = new Client(userString, portNumber);
                        setupGUI =new SetupGUI();
                        frame.setVisible(false);
                }
}

   private class Exit implements ActionListener
   {
     @Override
     public void actionPerformed(ActionEvent e) {

      System.exit(0);
```

```
    }


    }
    private class ConnectButton implements ActionListener
    {
       @Override
       public void actionPerformed(ActionEvent e) {


         connectPopUp = JOptionPane.showInputDialog(frame, panel,"Log in",
JOptionPane.QUESTION_MESSAGE );
          userString = userIP.getText();
          portString = userPort.getText();
                              checkLogin();



         System.out.println(connectPopUp);
       }


    }


 public static void main(String[] args)
 {
    MainMenu mainMenu = new MainMenu();
    mainMenu.createComponents();


 }


}
```

Server class

```
import java.net.*;
import java.io.*;

public class Server {
  private ServerSocket socket;
  private Socket playerOneSocket;
  private Socket playerTwoSocket;
  private BufferedReader playerOneIn;
  private BufferedReader playerTwoIn;
  private PrintWriter playerOneOut;
  private PrintWriter playerTwoOut;


  public Server(String port) {
   // TODO: server connection stuff

   try {
    socket = new ServerSocket(Integer.parseInt(port));
   } catch (IOException e) {
    System.err.println("Couldn't listen on: " + port + ". " + e.getMessage());
   }

   System.out.println("Server listening on port " + port);

   try {
    playerOneSocket = socket.accept();
```

```
} catch (IOException e) {
  System.err.println("Accepting player one failed: " + e.getMessage());
}


try {
  playerOneIn = new BufferedReader
            (new InputStreamReader(playerOneSocket.getInputStream()));
  playerOneOut = new PrintWriter
             (playerOneSocket.getOutputStream(), true);

} catch (IOException e) {
  System.err.println("Cannot read or write player one: " + e.getMessage());
}

System.out.println("Player one connected");

try {
  playerTwoSocket = socket.accept();

} catch (IOException e) {
  System.err.println("Accepting player two failed: " + e.getMessage());
}

try {
  playerTwoIn = new BufferedReader
            (new InputStreamReader(playerTwoSocket.getInputStream()));
  playerTwoOut = new PrintWriter
             (playerTwoSocket.getOutputStream(), true);
```

```
                String player2Input;


} catch (IOException e) {
  System.err.println("Cannot read or write player two: " + e.getMessage());

}


System.out.println("Player two connected");


Boolean doneSignin = false;
Boolean doneSetup = false;
Boolean playerOneSuccessfulSignin = false;
Boolean playerTwoSuccessfulSignin = false;
Boolean playerOneSuccessful = false;
Boolean playerTwoSuccessful = false;

while (!playerOneSuccessfulSignin) {
  try {
    String playerOneLine = playerOneIn.readLine();
    System.out.println(playerOneLine);
    if (playerOneLine == null) {
      doneSignin = true;
    } else {
      String message = GameManager.receiveMessage(1, playerOneLine);
      String[] response = message.replaceFirst("(1|2):", "").split(",");
      if (response.length > 1 && (response[1].equals("Unknown User") ||
```

```
        response[1].equals("Duplicate User Name"))) {
      playerOneOut.println(message);
      continue;
     } else {
      playerOneSuccessfulSignin = true;
      playerOneOut.println(message);
     }
   }
  } catch (IOException e) {
   System.err.println("Error reading player input: " + e.getMessage());
  }
}

while (!playerTwoSuccessfulSignin) {
  try {
   String playerTwoLine = playerTwoIn.readLine();
   System.out.println(playerTwoLine);
   if (playerTwoLine == null) {
    doneSignin = true;
   } else {
    String message = GameManager.receiveMessage(2, playerTwoLine);
    String[] response = message.replaceFirst("(1|2):", "").split(",");
    if (response.length > 1 && (response[1].equals("Unknown User") ||
       response[1].equals("Duplicate User Name"))) {
      playerTwoOut.println(message);
      continue;
    } else {
      playerTwoSuccessfulSignin = true;
      playerTwoOut.println(message);
```

```
    }
   }
  } catch (IOException e) {
   System.err.println("Error reading player input: " + e.getMessage());
  }
}

GameManager.setIsInSignin(false);
doneSignin = true;
System.out.println("Hey we're here now");

while (!doneSetup) {
  try {
   if (!playerOneSuccessful) {
    String playerOneLine = playerOneIn.readLine();
    System.out.println(playerOneLine);
    if (playerOneLine == null) {
     doneSetup = true;
    } else {
     String message = GameManager.receiveMessage(1, playerOneLine);
     System.out.println(message);
     String response = message.replaceFirst("(1|2):", "");
     if (message.charAt(0) == '1') {
      if (response.startsWith("ack")) {
       playerOneSuccessful = true;
      }
      playerOneOut.println(response);
     } else {
      playerTwoOut.println(response);
```

```
      }


    }
   }

  if (!playerTwoSuccessful) {
    String playerTwoLine = playerTwoIn.readLine();
    System.out.println(playerTwoLine);
    if (playerTwoLine == null) {
      doneSetup = true;
    } else {
      String message = GameManager.receiveMessage(2, playerTwoLine);
      String response = message.replaceFirst("(1|2):", "");
      System.out.println(response);
      if (message.charAt(0) == '1') {
        playerOneOut.println(response.replaceFirst("(1|2):", ""));
      } else {
        if (response.startsWith("ack")) {
          playerTwoSuccessful = true;
        }
        playerTwoOut.println(response.replaceFirst("(1|2):", ""));
      }
    }
  }
} catch (IOException e) {
  System.err.println("Error reading player input: " + e.getMessage());
}

if (playerOneSuccessful && playerTwoSuccessful) {
```

```
    GameManager.setIsInSetup(false);
    playerOneOut.println("ok");
    doneSetup = true;
  }
}

Boolean doneGame = false;
while (!doneGame) {
  try {
     String playerOneLine = playerOneIn.readLine();
     System.out.println("Player one input: " + playerOneLine);
     if (playerOneLine == null) {
       doneGame = true;
     } else {
       String message = GameManager.receiveMessage(1, playerOneLine);

       System.out.println("Player one messsage: " + message);
       String[] response = message.replaceFirst("(1|2):", "").split(",");
       if (response.length > 1 && response[1].equals("err")) {
         playerOneOut.println(response[1]);
         continue;
       }
       if (response.length > 2) {
         playerOneOut.println(response[1] + ",win");
         playerTwoOut.println(response[0] + ',' + response[1] + ',' + response[2]);
       } else {
         playerOneOut.println(response[1]);
         playerTwoOut.println(response[0] + ',' + response[1]);
       }
```

```
    }


    String playerTwoLine = playerTwoIn.readLine();
    System.out.println(playerTwoLine);
    if (playerTwoLine == null) {
      doneGame = true;
    } else {
      String message = GameManager.receiveMessage(2, playerTwoLine);
      String[] response = message.replaceFirst("(1|2):", "").split(",");
      if (response.length > 1 && response[1].equals("err")) {
        playerTwoOut.println(response[1]);
        continue;
      }
      if (response.length > 2) {
        playerTwoOut.println(response[1] + ",win");
        playerOneOut.println(response[0] + ',' + response[1] + ',' + response[2]);
      } else {
        playerTwoOut.println(response[1]);
        playerOneOut.println(response[0] + ',' + response[1]);
      }
    }

  } catch (IOException e) {
    System.err.println("Error reading player input: " + e.getMessage());
  }
}


        try {
```

```
      playerOneOut.close();
      playerOneIn.close();
      playerOneSocket.close();
    }
     catch (IOException e) {
       System.err.println("Unable to close player one's resources: "
                    + e.getMessage());
     }
     try {
       playerTwoOut.close();
       playerTwoIn.close();
       playerTwoSocket.close();
     }
      catch (IOException e) {
        System.err.println("Unable to close player two's resources: "
                     + e.getMessage());
      }
}


          }
```

Ship class

```
import java.util.ArrayList;


public class Ship {
 private String type;
 private ArrayList<String> parts;


 public Ship(String type, ArrayList<String> parts) {
  this.type = type;
  this.parts = parts;
 }


 public Boolean removePart(String location) {
  return parts.remove(location);
 }


 public Boolean isValid() {
  if (parts.size() == 0) {
   return false;
  }


  if (!type.equals("AC") && !type.equals("CR") && !type.equals("SB") &&
      !type.equals("FR")) {
   return false;
  }


  if (type.equals("AC") && parts.size() != 5) {
   return false;
```

```
}

if (type.equals("CR") && parts.size() != 4) {
  return false;
}

if (type.equals("SB") && parts.size() != 3) {
  return false;
}

if (type.equals("FR") && parts.size() != 2) {
  return false;
}

for (int i = 0; i < parts.size(); i++) {
  char row = parts.get(i).charAt(0);

  if (row != 'A' && row != 'B' && row != 'C' && row != 'D' && row != 'E' &&
      row != 'F' && row != 'G' && row != 'H' && row != 'I' && row != 'J') {
    return false;
  }

  char col = parts.get(i).charAt(1);

  if (col != '1' && col != '2' && col != '3' && col != '4' && col != '5' &&
      col != '6' && col != '7' && col != '8' && col != '9') {
    return false;
  }
```

```
    if (parts.get(i).length() > 2) {
      if (parts.get(i).charAt(2) == '1') {
        return false;
      }
    }
  }

  for (int i = 0; i < parts.size(); i++) {
    if (parts.get(0).charAt(0) != parts.get(i).charAt(0) && parts.get(0).charAt(1) != parts.get(i).charAt(1)) {
      return false;
    }
  }

  return true;
}

public Boolean partAt(String location) {
  return parts.contains(location);
}

public int getSize() {
  return parts.size();
}

public String toString() {
  String temp = type + " ";
  for (int i = 0; i < parts.size(); i++) {
    temp += parts.get(i) + " ";
  }
```

```
    return temp;
  }
}
```

User class

```
public class User {
  private String name;
  private int wins;
  private int losses;

  public User(String name) {
    this.name = name;
  }

  public void win() {
    wins++;
  }

  public void lose() {
    losses++;
  }

  public int getScore() {
    return wins * 10 + losses * -10;
  }

  public String getName() {
    return name;
  }
```

```
}
```