Name(s)_____ Period _____ Date _____

Semester Project - Student Written Responses Template

2. Written Responses

Submit one PDF file in which you respond directly to each prompt. Clearly label your responses 2a, 2b, 2c in order. Your response to all prompts combined must not exceed 550 words, exclusive of the Program Code.

Program Purpose and Development

2a. Provide a written response that:

- identifies the purpose of your program
- *and explains what the user should expect to see when they run the program.*

2a.
My project is 8 multiplayer game turn-based card game. Final Project has two project files. One establishes the server and manages the game. Other connects to the server and generate real-time updating GUI for display of the game. When user starts the program, both of the project will display GUI for IP address input followed by a warning if the establishment or connection fails. If the establishment of server was successful, new GUI announces the successful establishment. If the connection was successful, new GUI with game section, message section, and victory condition section is displayed. New GUI will interact with user to transfer and receive data with the server. Game section allows player to perform ability according to the randomly assigned character and end the turn during the player's turn. Message section allows player to chat with others if the player is alive. Last section displays all player's victory condition.

(Must not exceed 150 words)

2b. Describe the incremental and iterative development process of your program, focusing on two distinct points in that process. Describe the difficulties and/or opportunities you encountered and how they were resolved or incorporated. In your description clearly indicate whether the development described was collaborative or independent. At least one of these points must refer to independent program development.

2b.
When I was working with the GUI, I could not find the plugin for GUI visual editing tool that was automatically supported by my IDE. I downloaded both basic and advance tools. However, different from other IDEs for java, Intellij, my IDE, did not support Absolute Layout through visual editing. Therefore, I first designed the structure of GUI and calculated the game sections coordinates for the absolute layout with paper and pencil. Then, manually set every needed element of objects in GUI through coding. However, some of the objects within the panel were still not displayed as intended. Through developing a new project for experiment, I tested how all setting of an object within Absolute Layout influence its display and fixed the displaying problem.
After finishing multi-client connection testing, I tried to implement game system to the program. However, data transferred between server and clients became massive, and the unorganized data types and messages produced unresolvable bugs. I decided to clear out every data transferring code from the project and rebuild it through new organized data transfer system with united data type, which is String. Through new chart of system messages, I resolved most of the bugs I faced before.

(Must not exceed 200 words)

2c. Capture and paste the program code for an object you developed individually on your own. This object must contain a constructor, attributes, and methods, and fulfill a clear purpose in your final project. Explain how your object helped manage the complexity of your program.

2c.

Code:

```
package com.company;

import javax.swing.*;
import java.io.DataInputStream;
import java.io.IOException;
import java.io.PrintStream;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.UnknownHostException;

/**Manage Thread of Each Client Connection*/
class ServerDataTransferManagement extends Thread {     // how server manage data



    /**Objects for Each Client*/
    private ServerDataTransferManagement[] dataTransferManagements;
    private Socket clientSocket;
    private DataInputStream dataInputStream;
    private PrintStream dataOutputStream;

    /**Variables for Each Client*/
    private static int connectedClientNumber =0;
    private int maxClientsCount;
    private static int receivingFromPlayerIndex = -1;


    /**Initialize variables for each Thread*/
    public ServerDataTransferManagement(Socket clientSocket, ServerDataTransferManagement[]
dataTransferManagements) {
        this.clientSocket = clientSocket;
        this.dataTransferManagements = dataTransferManagements;
        maxClientsCount = dataTransferManagements.length;
    }


    @Override
    public void run() {
        /**Find Index of this thread in Thread Array*/
        for (int i = 0; i < maxClientsCount; i++)
{                                                                   //For each index
            if(dataTransferManagements[i] ==
this){                                                                  //Check if the
Thread in index is the current Thread
                receivingFromPlayerIndex =i;
//Update the current Thread Index
            }
        }
        ServerDataTransferManagement[] dataTransferManagements =
this.dataTransferManagements;                           //Update Thread Array for each run


        try {
            dataInputStream = new DataInputStream(clientSocket.getInputStream());
//Create stream for input from client
            dataOutputStream = new PrintStream(clientSocket.getOutputStream());
//Create stream for output from client
            /**Check for Client Connection made*/
            while (true)
{
    //Continuously Check
                if (dataInputStream.readLine().trim().equals("(Entered)"))
```

```
{                                           //Check if connected Client is actually
responding
                    connectedClientNumber++;
//Increase the number of connected client

                    /**Initial Outputs containing player information for each Client*/
                    dataOutputStream.println("(Name)" +
PlayableCharacters.getCharacter(Game.characterIndexForPlayers.get(receivingFromPlayerIndex)
).getName());
                    dataOutputStream.println("(Position)" +
PlayableCharacters.getCharacter(Game.characterIndexForPlayers.get(receivingFromPlayerIndex)
).getPosition());
                    dataOutputStream.println("(Ability)" +
PlayableCharacters.getCharacter(Game.characterIndexForPlayers.get(receivingFromPlayerIndex)
).getAbility());
                    dataOutputStream.println("(VictoryCondition)" +
PlayableCharacters.getCharacter(Game.characterIndexForPlayers.get(receivingFromPlayerIndex)
).getVictoryCondition());
                    dataOutputStream.println("(ServerOutput)" + "you are Player " +
receivingFromPlayerIndex);
                    dataOutputStream.println("(PlayerIndex)" + receivingFromPlayerIndex);

dataOutputStream.println("(CharacterIndex)"+Game.characterIndexForPlayers.get(receivingFrom
PlayerIndex));
                    dataOutputStream.println("(Round)"+Game.round);

                    /**Notification for all clients as each Client enters*/
                    for (int i = 0; i < maxClientsCount; i++) {
                        /**A Client is accepted to the server*/
                        if (dataTransferManagements[i] != null && i !=
receivingFromPlayerIndex) {

dataTransferManagements[i].dataOutputStream.println("(ServerOutput)"+"*** Player " +
receivingFromPlayerIndex
                                    + " entered the Game ***");
                        }
                        /**All player has entered the server*/
                        if (connectedClientNumber == maxClientsCount) {

dataTransferManagements[i].dataOutputStream.println("(GameStart)" + "Game Started");

dataTransferManagements[i].dataOutputStream.println("(CurrentPlayer)" +
Game.currentPlayerIndex);
                        }
                    }
                    break;
                }
            }
            /**Check for data Client sent*/
            while (true) {
                String line = dataInputStream.readLine();               //received data
from Client

                /**Identify the Client Index in the Array*/
                for (int i = 0; i < maxClientsCount; i++) {
                    if(dataTransferManagements[i] == this){
                        receivingFromPlayerIndex =i;

                    }
                }
                /**Check if Client is Requesting Public Message to all players*/
                if(line.startsWith("(PublicMessage)")) {
                    String message = line.substring(new
String("(PublicMessage)").length());
                    for (int i = 0; i < maxClientsCount; i++) {
                        if (dataTransferManagements[i] != null &&
dataTransferManagements[i]!= this) {

dataTransferManagements[i].dataOutputStream.println("(PublicMessage)"+"Player "+
receivingFromPlayerIndex +" to all: "+message);
                        }
```

```
                    }
                }
                /**Check if Client is Requesting Private Message to an player*/
                else if (line.startsWith("(PrivateMessage)")){
                    int playerIndexToSend=Integer.valueOf(line.substring(new
String("(PrivateMessage)").length(),new String("(PrivateMessage)").length()+1));
                    String message = line.substring(new
String("(PrivateMessage)").length()+1);

dataTransferManagements[playerIndexToSend].dataOutputStream.println("(PrivateMessage)"+"Pla
yer "+ receivingFromPlayerIndex +" to "+playerIndexToSend+": "+message);

                }
                /**Check if Client is Requesting Server to return some type of data*/
                else if(line.startsWith("(Server)")){
                    line=line.substring(new String("(Server)").length());
                    /**Check if Client is Requesting Server to Kill certain Player and is
the killing allowed for the Client*/
                    if(line.startsWith("(ToKill)")){
                        int playerIndexToKill = Integer.valueOf(line.substring(new
String("(ToKill)").length(),new String("(ToKill)").length()+1));
                        System.out.println(receivingFromPlayerIndex +" is requesting To
Kill towards Player"+playerIndexToKill);
                        /**Check who sent the request and announce the result if the
request is accepted*/
                        switch
(Game.characterIndexForPlayers.get(receivingFromPlayerIndex)){
                            case 3:

if(Game.characterIndexForPlayers.get(playerIndexToKill)==4){
                                    PlayableCharacters.getCharacter(4).setAlive(false);

System.out.println(PlayableCharacters.getCharacter(4).getName()+" is
alive:"+PlayableCharacters.getCharacter(4).alive);
                                    for(int i=0;i<maxClientsCount; i++) {

dataTransferManagements[i].dataOutputStream.println("(ServerOutput)"+"Player"+playerIndexTo
Kill+" is Dead");
                                    }

dataTransferManagements[playerIndexToKill].dataOutputStream.println("(Dead)");
                                } else
if(Game.characterIndexForPlayers.get(playerIndexToKill)==7){
                                    PlayableCharacters.getCharacter(7).setAlive(false);

System.out.println(PlayableCharacters.getCharacter(7).getName()+" is
alive:"+PlayableCharacters.getCharacter(7).alive);
                                    for(int i=0;i<maxClientsCount; i++) {

dataTransferManagements[i].dataOutputStream.println("(ServerOutput)"+"Player"+playerIndexTo
Kill+" is Dead");

if(PlayableCharacters.getCharacter(7).getPosition().equals(Position.positions[8])){

dataTransferManagements[i].dataOutputStream.println("(ServerOutput)"+"The King is dead");
                                        }
                                    }

dataTransferManagements[playerIndexToKill].dataOutputStream.println("(Dead)");

if(PlayableCharacters.getCharacter(7).getPosition().equals(Position.positions[8])) {
                                        dataTransferManagements[(int) (Math.random() *
8)].dataOutputStream.println("(ServerOutput)" + receivingFromPlayerIndex + " killed the
King");
                                    }
                                }
                                break;
                            case 5:

if(Game.characterIndexForPlayers.get(playerIndexToKill)==2){
                                    PlayableCharacters.getCharacter(2).setAlive(false);
```

```
System.out.println(PlayableCharacters.getCharacter(2).getName()+" is
alive:"+PlayableCharacters.getCharacter(2).alive);
                                } else if
(Game.characterIndexForPlayers.get(playerIndexToKill)==3){
                                    PlayableCharacters.getCharacter(3).setAlive(false);

System.out.println(PlayableCharacters.getCharacter(3).getName()+" is
alive:"+PlayableCharacters.getCharacter(3).alive);
                                    for(int i=0;i<maxClientsCount; i++) {

dataTransferManagements[i].dataOutputStream.println("(ServerOutput)"+"Player"+playerIndexTo
Kill+" is Dead");
                                    }

dataTransferManagements[playerIndexToKill].dataOutputStream.println("(Dead)");
                                }

                                break;
                            case 7:

if(Game.characterIndexForPlayers.get(playerIndexToKill)==0){
                                    PlayableCharacters.getCharacter(0).setAlive(false);

System.out.println(PlayableCharacters.getCharacter(0).getName()+" is
alive:"+PlayableCharacters.getCharacter(0).alive);
                                    for(int i=0;i<maxClientsCount; i++) {

dataTransferManagements[i].dataOutputStream.println("(ServerOutput)"+"Player"+playerIndexTo
Kill+" is Dead");

dataTransferManagements[i].dataOutputStream.println("(ServerOutput)"+"The King is dead");

                                    }

dataTransferManagements[playerIndexToKill].dataOutputStream.println("(Dead)");
                                    dataTransferManagements[(int)
(Math.random()*8)].dataOutputStream.println("(ServerOutput)"+ receivingFromPlayerIndex +"
killed the King");
                                    PlayableCharacters.getCharacter(7).setPosition(8);
                                } else if
(Game.characterIndexForPlayers.get(playerIndexToKill)==1){
                                    PlayableCharacters.getCharacter(1).setAlive(false);

System.out.println(PlayableCharacters.getCharacter(1).getName()+" is
alive:"+PlayableCharacters.getCharacter(1).alive);
                                    for(int i=0;i<maxClientsCount; i++) {

dataTransferManagements[i].dataOutputStream.println("(ServerOutput)"+"Player"+playerIndexTo
Kill+" is Dead");
                                    }

dataTransferManagements[playerIndexToKill].dataOutputStream.println("(Dead)");
                                } else if
(Game.characterIndexForPlayers.get(playerIndexToKill)==6){
                                    PlayableCharacters.getCharacter(6).setAlive(false);

System.out.println(PlayableCharacters.getCharacter(6).getName()+" is
alive:"+PlayableCharacters.getCharacter(6).alive);
                                    for(int i=0;i<maxClientsCount; i++) {

dataTransferManagements[i].dataOutputStream.println("(ServerOutput)"+"Player"+playerIndexTo
Kill+" is Dead");
                                    }

dataTransferManagements[playerIndexToKill].dataOutputStream.println("(Dead)");
                                }
                                break;
                            default:
                                break;
                        }
```

```
                        /**Check if any player has gained victory*/
                        if(PlayableCharacters.checkForVictoriousPlayer()!=-1){
                            for (int i = 0; i < maxClientsCount; i++) {
                                /**Announce to all Clients that certain player has gained
victory*/
                                if (dataTransferManagements[i] != null) {

dataTransferManagements[i].dataOutputStream.println("(Victory)"+"Player
"+PlayableCharacters.checkForVictoriousPlayer()+" won the Game");
                                }
                            }
                        }
                    }
                    /**Check if Client is Requesting Server to return some information
about other Players*/
                    else if(line.startsWith("(ToFind)")){
                        line = line.substring(new String("(ToFind)").length());
                        /**Check is Client is asking Which player has certain Character*/
                        if(line.startsWith("(PlayerIndexOfCharacter)")){
                            System.out.println("Player "+ receivingFromPlayerIndex +"is
requesting PlayerIndexOfCharacter");
                            int characterIndex = Integer.valueOf(line.substring(new
String("(PlayerIndexOfCharacter)").length(),new
String("(PlayerIndexOfCharacter)").length()+1));
                            int playerIndexOfCharacter =-1;
                            /**Returns the Information to the Client*/
                            for(int i =0; i<8; i++){
                                if(Game.characterIndexForPlayers.get(i)==characterIndex){
                                    playerIndexOfCharacter = i;
                                }
                            }

dataTransferManagements[receivingFromPlayerIndex].dataOutputStream.println("(ServerOutput)"
+"Player "+playerIndexOfCharacter+" is
"+PlayableCharacters.getCharacter(characterIndex).getName());
                        }
                        /**Check is Client is asking Position of certain Player*/
                        else if(line.startsWith("(PositionOfPlayer)")){
                            int playerIndex = Integer.valueOf(line.substring(new
String("(PositionOfPlayer)").length()));
                            /**Returns the Information to the Client*/
                            String positionOfPlayer =
PlayableCharacters.getCharacter(Game.characterIndexForPlayers.get(playerIndex)).getPosition
();

dataTransferManagements[receivingFromPlayerIndex].dataOutputStream.println("(ServerOutput)"
+positionOfPlayer+" is the position of Player "+playerIndex);
                        }

                    }
                }
                /**Check if Client is Requesting Ending its turn*/
                else if(line.startsWith("(EndTurn)")){
                    /**Check if all player has finished their turn during the round*/
                    if(Game.currentPlayerIndex!=7){
                        /**Update Current Player Information for all Client*/
                        Game.currentPlayerIndex++;

                        for (int i = 0; i < maxClientsCount; i++) {
                            if (dataTransferManagements[i] != null) {

dataTransferManagements[i].dataOutputStream.println("(CurrentPlayer)"+Game.currentPlayerInd
ex);



                            }
                        }

                    } else {
                        /**Update Current Player and Current Round Information for all
Client*/
```

```
                        Game.currentPlayerIndex=0;
                        Game.round++;

                        for (int i = 0; i < maxClientsCount; i++) {
                            if (dataTransferManagements[i] != null) {

dataTransferManagements[i].dataOutputStream.println("(CurrentPlayer)"+Game.currentPlayerInd
ex);


dataTransferManagements[i].dataOutputStream.println("(Round)"+Game.round);

                            }
                        }

                    }
                    /**Check if any player has gained victory*/
                    if(PlayableCharacters.checkForVictoriousPlayer()!=-1){
                        /**Announce to all Clients that certain player has gained victory*/
                        for (int i = 0; i < maxClientsCount; i++) {
                            if (dataTransferManagements[i] != null ) {

dataTransferManagements[i].dataOutputStream.println("(Victory)"+"Player
"+PlayableCharacters.checkForVictoriousPlayer()+" won the Game");
                            }
                        }
                    }
                }


            }

        } catch (IOException e) {}              //Catch IO Exception
    }

}
```

Explanation:
This object resolved the complexity of my program through managing all message transferred between server and client. By keeping track of the client index of each message received and categorizing the message according to the request type. After categorizing the message, this object checks for specified request type or, if there is not specified request type, check if request is valid from the received client. If the request is valid it changes the data of the game according to the instruction, reply to the client and, if necessary, check for any victory.
Checking for valid client is made through correlating various of data, such as player index of client, character index of client, if player is alive, ability condition of character.
Reply to the client include information about main request type to the client, if necessary, minor request type to the client, and request to the client.
Checking for victory is done without any data received from the client but checks for data manipulation in the game class and playable character class to identify the victorious player.
Through all these processes it reduces the complexity of project and allow the project concentrate on the user interaction and connection with clients, rather than communication.

(Must not exceed 200 words)

Submit all written responses and program code in a single zip file via Moodle.