

DistriUted Programming

MOD006128 2019-20 Tri 2



SID: 1651542

Contents

[Part 1 (Spinning Car) 2](#_Toc39698449)

[Description of the program 2](#_Toc39698450)

[Use of resources in the next Section 3](#_Toc39698451)

[Part 2 (Offline Car Racing Game) 4](#_Toc39698452)

[GUI Description 5](#_Toc39698453)

[Part 3 (Racing game on a network) 7](#_Toc39698454)

[The language used to communicate with Server 7](#_Toc39698455)

[Other Information 8](#_Toc39698456)

[References 9](#_Toc39698457)

[Appendix (Print-Out of code) 10](#_Toc39698458)

[Part 1 10](#_Toc39698459)

[Part 2 12](#_Toc39698460)

[Part 3 19](#_Toc39698461)

[Server 19](#_Toc39698462)

[Game Client 23](#_Toc39698463)

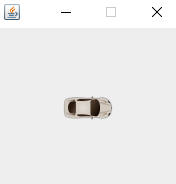
# Part 1 (Spinning Car)



The author created a set of programmatically generated png images named from 0 – 15 (for ease of use in the program) displaying a car from a top perspective were using a small application developed by the author for this task. The resources for this Section of the assignment are in the directory named “Part 1”.

As requested, the images created are of the size 50 pixels by 50 pixels. The use of software to generate most of the photos ensured the models kept the original quality of the first one. The author created the initial image using a tool called “Paint.net”. This approach of generating images and method of naming them was for the sake of efficient development of the program described in the subsection below.

## Description of the program



The program for this task is one that spins a car at a designated rate (100 milliseconds in this case). The program makes use of java features such as a timer, Action listener, JPanel and JForm to display the 16 generated images after a specified interval.

The JPanel is customized to be an action listener in the JForm window. Within the JPanel, there is a timer that ticks every 100 milliseconds. After every timer lap, the index of the sprite image is increased, and the picture named after that index will me displayed. This gives the program the spinning effect.

## Use of resources in the next Section

The images generated in this Section are also resources used in part 2 and 3 of this assignment. Additionally, the author produced an extra set of pictures and used them in the programs in the proceeding sections. The structure of classes were also used in the next part.

# Part 2 (Offline Car Racing Game)

The task for this Section is to create an offline racing game or simulation for two players. The application made is one with a racetrack where two players drive cars around a central grassed area on one personal computer or workstation. The players can accelerate, decelerate, reverse and change the direction of the vehicle. Also, in the event of both cars colliding, the game will come to an end.

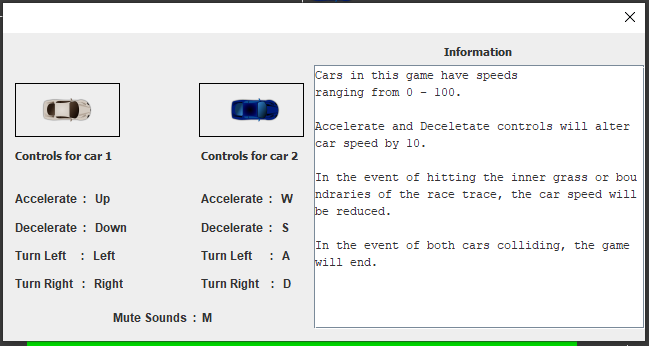


Figure 1 (Game Info window)

**Direction Change**

The image above shows the critical controls of the game. For each player, two keys change the direction of each car (Left or Right turn keys). The turning effect makes use of sprite images from Part 1. The program makes use of the way image files are named (0 to 15) and putting them in an array in that order, turning left displays an image of a previous index and turning right, the immediate next index. A timer runs in the background manipulating the x and y values of the car images and refreshing game graphics.

A few sprite images are below to help better visualize the explanation above.



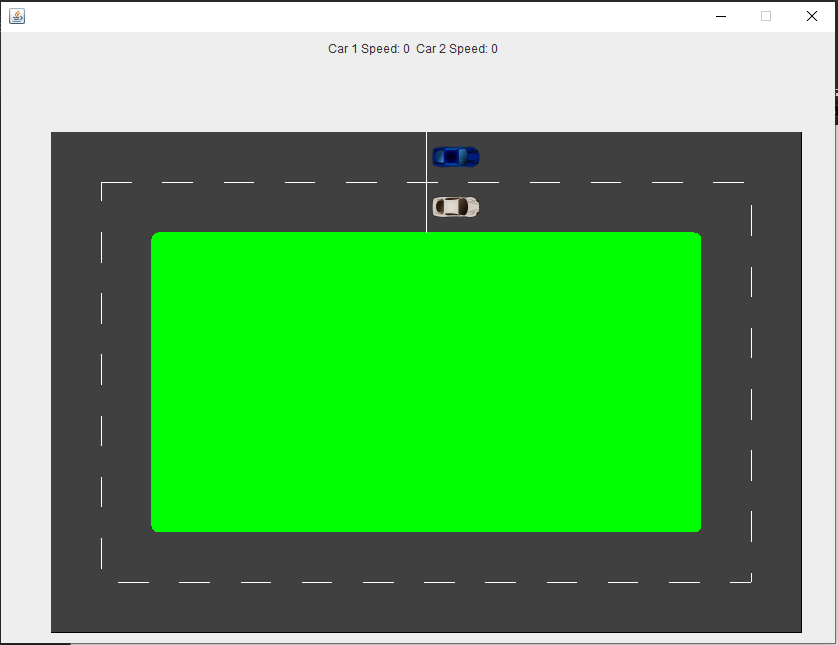
Figure 2 (Visual aid for understanding sprite behavior)

One thing to note about this method is the confusing key control when driving a car downwards (Sprite images 3, 4 and 5). Because of top view perspective, left and right turning the vehicle when at index 3, 4 and 5 in the game are mirrored (Left seems to be right and left, right).

**Acceleration and Deceleration**

Also, two other keys are used to either increase or decrease car speed by 10 (Accelerate or Decelerate Keys). Each car has a maximum speed of 100 and a minimum of 0 (Park). Though the game allows players to reverse their assigned vehicles, reversal speed may be reflected as a negative speed with a limit of -100.

## GUI Description



The racing arena is in a window of 850 pixels by 650 pixels. The author has modified the racetrack to look more appealing visually. In addition to transparent car image backgrounds, the racing road track resembles an actual Road.

**Collision strategies**

The application also employs some collision strategies to make the driving of cars more realistic and aid in the controllability of the vehicles. Cars slow down automatically when approaching sharp turning corners. Cars are also parked (Speed made to be 0) automatically when attempting to leave the racing area or enter the grass area. And the game ends when both cars collide. These events are accompanied by sounds which can be muted by pressing the key ’M’ on the keyboard.

# Part 3 (Racing game on a network)

This Section involves converting the application from Section 2 to a network-based application. In this case, the players control their assigned car using arrow keys. The left key turns the car right, the left, left—the up and down keys, to accelerate and decelerate the vehicle respectively.

The significant difference between this proposed program and that of Section 2 is that this program is network-based. It is accompanied by a server program which must be run first before the racing game is launched. Instructions on how to run the Server and Client are in the “part 3” directory in a text file called “Readme.txt”.

The client application is just a racing game. A player controls one car, and the other vehicle is updated by another player on another instance of the program. On the background, each client has a thread responsible for receiving and sending messages to and from the Server. In contrast, the Server is a single-threaded application that receives signals from the clients, analyses the code and informs concerned clients about the right action to take.

## The language used to communicate with Server

The table below shows the codes transferred between client and Server and their meanings.

|  |  |  |
| --- | --- | --- |
| **Code** | **Purpose** | **When Called** |
| 0 | This code assigns the control of the white car to the client receiving the message. | It is sent to a client when initially connecting to the Server. |
| 1 | This code assigns the control of the blue car to the client receiving the message. |
| >=2 | When a number higher than 1 is sent to a client, it means that all available cars have been assigned taken. | This message is sent when a client tries to join the network the Server has assigned the two available cars. |
| restart | This message informs the clients to restart a game after cars have collided | This message is sent when a car collision has happened and players wish to continue with the game. |
| exit | The exit command is a message sent to all clients instructing them to close. | This message is sent to the Server and other available clients when a client leaves the network. |
| "Car Sprite\_Image\_Index Car\_X\_value Car\_Y\_value CarSpeed" | Messages in this format are sent to clients informing them of the car location and details of the other player’s car. | This message is sent when a player either turns their car or changes their car speed. |

## Other Information

The network created between the Server and Clients makes use of the User Datagram Protocol. The protocol was used because of its speed. Though because this protocol does not guarantee the successful delivery of messages, a client’s display may lag from time to time.

On the event of cars colliding, if a user decides to restart the game, it will be forcefully reset on the other client. But if the other client does not want to continue, closing the client window while the game is going on will instruct the server to close all servers down.

# References

Oracle, a. *Class Graphics.* [on-line] Available at: <<https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html>> [Accessed: May 5, 2020].

Oracle, b. *Class Rectangle.* [on-line] Available at: <<https://docs.oracle.com/javase/7/docs/api/java/awt/Rectangle.html>> [Accessed: May 5, 2020].

Oracle, c. *Class Thread.* [on-line] Available at: <<https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html>> [Accessed: May 5, 2020].

Oracle, d. *Graphics2D.* [on-line] Available at: <<https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics2D.html>> [Accessed: May 5, 2020].

Oracle, e. *Reading from and Writing to a Socket.* [on-line] Available at: <<https://docs.oracle.com/javase/tutorial/networking/sockets/readingWriting.html>> [Accessed: May 5, 2020].

ratanak1010, 2011. *Java-UDP-Chat.* [on-line] Available at: <<https://github.com/ratanak1010/Java-UDP-Chat>> [Accessed: May 4, 2020].

# Appendix (Print-Out of code)

This section contains some screenshots of relevant code.

## Part 1

import javax.swing.\*;  
  
public class ProgramEntry   
{  
  
 public static void main(String[] args)   
 {  
 //Initalize form  
 Form mainForm = new Form();  
   
 //Set form size  
 mainForm.setSize(200, 200);  
 mainForm.setResizable(false);  
   
 //Alow form to close on the pressing of "X"  
 mainForm.setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);  
   
 //Display form  
 mainForm.show();  
 }  
  
}  
import javax.swing.\*;  
  
public class Form extends JFrame  
{  
 //Panel that will be used for all graphical work in the program  
 Palette graphicsPalette;  
   
 public Form()  
 {  
 //initalize graohicsPalette  
 graphicsPalette = new Palette();  
 //fill form with graphics palette  
 graphicsPalette.setSize(WIDTH, HEIGHT);  
 this.add(graphicsPalette);  
 }  
   
}  
import java.awt.Graphics;  
import java.awt.event.\*;  
import javax.swing.\*;  
  
public class Palette extends JPanel implements ActionListener  
{  
   
 Timer timer; //Swing timer that controls car spinning  
 ImageIcon[] car1Images; //Array of sprite images  
   
 int carPhase = 0;//Phase of car(Car at a specific angle)  
   
 public Palette()  
 {  
 //Initalize swing timer  
 timer = new Timer(100, this);  
   
 //Initalize car images  
 car1Images = new ImageIcon[16];  
   
 String path = "car1\\";// Path of images  
   
 //Fill car1Images with images from path directory  
 for(int i = 0; i < car1Images.length; i++)  
 {  
 String currentCar = path + String.valueOf(i) + ".png"; //full image path  
   
 //initalize image object to hold the current image  
 car1Images[i] = new ImageIcon(currentCar);  
 }  
   
 //Start timer  
 timer.start();  
 }  
   
 @Override  
 public void actionPerformed(ActionEvent e)   
 {  
 if(e.getSource() == timer)  
 {  
 //If timer tick  
   
 //Rotate sprite by 22.5 degrees by changing index of current image  
 carPhase++;//Change phase of car  
   
   
 if(carPhase == 16)  
 {  
 //Reset image to original angle  
 carPhase = 0;  
 }  
   
 //Refresh graphics  
 repaint();  
 }  
 }  
   
 @Override  
 public void paintComponent(Graphics g) {  
 super.paintComponent(g);  
   
 //50 is the height and width of images being drawn to the panel  
 int x = (getWidth() - 50)/2;  
 int y = (getHeight() - 50)/2;  
   
 car1Images[carPhase].paintIcon(this, g, x, y);  
 }  
}

## Part 2

import java.awt.\*;  
import java.awt.Stroke;  
import java.awt.event.\*;  
import java.io.File;  
import javax.sound.sampled.AudioSystem;  
import javax.sound.sampled.Clip;  
import javax.swing.\*;  
  
 /\*\*  
 \*  
 \* Author: SID 1651542  
 \*/  
  
public class Palette extends JPanel implements ActionListener, KeyListener  
{  
 Timer timer; //Swing timer that controls car spinning  
 ImageIcon[] car1Images, car2Images; //Array of sprite images  
   
 int[] carPhase;//Phase of car(Car at a specific angle)  
 int[] carPosition; //array holding x and y value of cars  
   
 int[] carLane; //Lane 1 -> inner lane, Lane 2 -> outer lane  
   
 int[] displacementFactor; //Displacement factor of car 1 and 2  
   
 Stroke dashedStroke;  
   
 Label carSpeedLabel;  
   
 Clip[] clips;  
   
 boolean playSound;  
   
 public Palette()  
 {  
 carPhase = new int[] {0, 0};  
 carLane = new int[] {0, 0};  
 carPosition = new int[]{430, 150, 430, 100}; //X, Y values of car 1 and 2  
 displacementFactor = new int[] {0, 0}; //Displacement facor of car 1 and 2  
   
 carSpeedLabel = new Label("Car 1 Speed: 0 Car 2 Speed: 0");  
 add(carSpeedLabel);   
   
 //Configure key listner  
 addKeyListener(this);  
   
 try  
 {  
 clips = new Clip[3];  
 clips[0] = AudioSystem.getClip();   
 clips[0].open(AudioSystem.getAudioInputStream(new File("Audio\\swerve.wav")));  
   
 clips[1] = AudioSystem.getClip();  
 clips[1].open(AudioSystem.getAudioInputStream(new File("Audio\\crash.wav")));  
   
 clips[2] = AudioSystem.getClip();  
 clips[2].open(AudioSystem.getAudioInputStream(new File("Audio\\drive.wav")));  
 }  
 catch(Exception e)  
 {  
   
 }  
   
 playSound = true;  
   
 //Initalize car images  
 car1Images = new ImageIcon[16];  
 car2Images = new ImageIcon[16];  
   
 String path = "car1\\";// Path of car1 images  
 String path2 = "car2\\";// Path of car2 images  
   
 //Fill car1Images with images from path directory  
 for(int i = 0; i < car1Images.length; i++)  
 {  
 String currentCar1 = path + String.valueOf(i) + ".png"; //full image path  
 String currentCar2 = path2 + String.valueOf(i) + ".png"; //full image path  
 //initalize image object to hold the current image  
 car1Images[i] = new ImageIcon(currentCar1);// Current sprite image of car 1  
 car2Images[i] = new ImageIcon(currentCar2);//Current sprite image of car 2  
 }  
   
 dashedStroke = new BasicStroke(1, BasicStroke.CAP\_BUTT, BasicStroke.JOIN\_ROUND, 50, new float[]{30.6f, 0, 0}, 0);  
   
 //Initalize swing timer  
 timer = new Timer(100, this);  
 timer.start();  
 }  
   
 void PlaySound(int index)   
 {  
 if(!timer.isRunning()) return;  
 if(!playSound) return;  
   
 try  
 {  
 //Restart clip if reached end  
 int frameLength = clips[index].getFrameLength();  
 if(clips[index].getFramePosition() == frameLength)  
 clips[index].setMicrosecondPosition(1);  
 //Play clip  
 clips[index].start();  
 }  
 catch (Exception exc)  
 {  
 exc.printStackTrace(System.out);  
 }  
 }  
   
 void IncreaseCarSpeed(int car)  
 {  
 if (displacementFactor[car] < 10)   
 displacementFactor[car] += 1;  
   
 UpdateDetails();  
 }  
  
 void ReduceCarSpeed(int car)  
 {  
 if (displacementFactor[car] > -10)  
 displacementFactor[car] -= 1;  
   
 UpdateDetails();  
 }  
  
 void AssertCarLane(int car)  
 {  
 int carX = carPosition[car == 0 ? 0 : 2];  
 int carY = carPosition[car == 0 ? 1 : 3];  
   
 if(carX <= 70 || carX >= 734|| carY <= 118 || carY >= 531)  
 carLane[car] = 2;  
 else  
 carLane[car] = 1;  
 }  
  
 void ControlCarSpeed(int car, int lane)  
 {  
 int xLim0, xLim1, yLim1, yLim0; //Race track corners  
 int carX; int carY; int disFac;// Car x, y values and displacement factor  
   
 //index of sprite image  
 int carDirection = this.carPhase[car];  
   
 if(lane == 1)  
 {  
 xLim0 = 148; xLim1 = 649;  
 yLim1 = 450; yLim0 = 200;  
 }  
 else  
 {  
 xLim0 = 97; xLim1 = 700;  
 yLim1 = 502; yLim0 = 144;  
 }  
  
 carX = carPosition[car == 0 ? 0 : 2];  
 carY = carPosition[car == 0 ? 1 : 3];  
 disFac = displacementFactor[car];  
   
 //Car is close to top right corner  
 boolean nearTopR = carX > xLim1 - disFac \* 5 && carX < xLim1;  
   
 //Car is close to lower right corner  
 boolean nearlowerR = carY > yLim1 - disFac \* 5   
 && carY < yLim1 && carDirection == 4;  
   
 //Car is close to top left corner  
 boolean nearTopL = carY <= yLim0 + disFac \* 5   
 && carY > yLim0 && carDirection == 12;  
   
 //Car is close to lower left corner  
 boolean nearlowerL = carX <= xLim0 + disFac \* 5 && carX > xLim0 ;  
   
 //Reduce car speed if near road corner  
 if((nearTopR || nearlowerR || nearTopL || nearlowerL) && disFac > 4)   
 ReduceCarSpeed(car);  
   
 }  
  
 void MoveCar(int car)  
 {  
 int xDiff = 0, yDiff = 0;  
   
 switch(carPhase[car])  
 {  
 case 0: case 8:  
 xDiff = displacementFactor[car];  
 if(carPhase[car] == 8)  
 xDiff \*= -1; //Move car left  
 break;  
   
 case 1: case 2: case 3: case 5: case 6: case 7:  
 case 9: case 10: case 11: case 13: case 14: case 15:  
 xDiff = displacementFactor[car] / 2;  
 yDiff = displacementFactor[car] / 2;  
   
 //Move car left  
 if (carPhase[car] >= 5 && carPhase[car] <= 11) xDiff \*= -1;  
 //Move car up  
 if (carPhase[car] >= 9 && carPhase[car] <= 15) yDiff \*= -1;  
 break;  
   
 case 4: case 12:  
 yDiff = displacementFactor[car];  
   
 if(carPhase[car] == 12) yDiff \*= -1; //Move car up  
 break;  
 }  
 //Compare car position and speed. Assert if car will crash  
 handleCarCollision(car, xDiff, yDiff);  
 }  
   
 void handleCarCollision(int car, int xDiff, int yDiff)  
 {  
 int xIndex = car == 0 ? 0 : 2; //X index of car   
 int yIndex = car == 0 ? 1 : 3; //Y index of car  
   
 var carX = carPosition[xIndex] + xDiff; //new x position of car  
 var carY = carPosition[yIndex] + yDiff; //new y position of car  
   
 //If car intersects with grass in the centre or attempts leaving race track   
 if(carX > 118 && carX < 682 && carY > 160 && carY < 487  
 || carX < 40 || carX > 760 || carY < 90 || carY > 560)  
 {  
 displacementFactor[car] = 0; //Slow car speed down  
 UpdateDetails();  
 PlaySound(0); //Play swerve sound  
 }  
 else  
 {  
 Rectangle car1 = new Rectangle(carPosition[0],carPosition[1], 30, 20);  
 Rectangle car2 = new Rectangle(carPosition[2],carPosition[3], 30, 20);  
 //if cars crash  
 if(car1.intersects(car2))  
 {  
 PlaySound(1); //Play crash sound  
 timer.stop();  
   
 JOptionPane.showMessageDialog(this,"Game over, cars crashed. Type 'R' on the game UI to restart");  
 }  
 else  
 {  
 //Move car normally  
 carPosition[xIndex] = carX;  
 carPosition[yIndex] = carY;  
 }  
   
 }  
 }  
  
 @Override  
 public void keyPressed(KeyEvent e)   
 {  
 if(!timer.isRunning() && e.getKeyCode() != KeyEvent.VK\_R) return;  
 switch (e.getKeyCode()) {  
 //Key control for car 1  
 case KeyEvent.VK\_UP: IncreaseCarSpeed(0); break;  
 case KeyEvent.VK\_DOWN: ReduceCarSpeed(0); break;  
 case KeyEvent.VK\_LEFT: rotateLeft(0); break;  
 case KeyEvent.VK\_RIGHT: rotateRight(0); break;  
   
 //Car control for car 2  
 case KeyEvent.VK\_W: IncreaseCarSpeed(1); break;  
 case KeyEvent.VK\_S: ReduceCarSpeed(1); break;  
 case KeyEvent.VK\_A: rotateLeft(1); break;  
 case KeyEvent.VK\_D: rotateRight(1); break;  
 default: break;  
   
 case KeyEvent.VK\_M: playSound = !playSound; break;  
   
 case KeyEvent.VK\_R:   
 if(!timer.isRunning())  
 {  
 carPosition = new int[]{430, 150, 430, 100}; //X, Y values of car 1 and 2  
 displacementFactor = new int[] {0, 0}; //Displacement facor of car 1 and 2  
 timer.start();  
 }  
 break;  
 }  
   
 }  
   
 void UpdateDetails()  
 {  
 String carStatus = "Car 1 Speed:"+ displacementFactor[0] \* 10 + " ";  
 carStatus += "Car 2 Speed:" + displacementFactor[1] \* 10;  
 carSpeedLabel.setText(carStatus);  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent e)  
 {  
 //Event handler for timer tick  
 if(e.getSource() != timer) return;  
 for (int i = 0; i < 2; i++)   
 {  
 ControlCarSpeed(i,carLane[i]); //Slow car down if near corner  
 MoveCar(i); //Move car  
 AssertCarLane(i); //Assert what lane car is on  
   
 //Play car drving sound  
 if(displacementFactor[i] > 0) PlaySound(2);  
 }  
 repaint();  
 }  
  
 @Override  
 public void keyReleased(KeyEvent e) { }  
  
 @Override  
 public void keyTyped(KeyEvent e) {}  
   
 @Override  
 public void paintComponent(Graphics gr)   
 {  
 Graphics2D g = (Graphics2D) gr.create();  
   
 super.paintComponent(g);  
   
 g.setColor(Color.black); g.drawRect(50, 100, 750, 500); // outer edge  
 g.setColor(Color.darkGray); g.fillRect(50, 100, 750, 500); // outer edge  
   
 g.setColor(Color.darkGray); g.fillRect( 100, 150, 650, 400 ); // Inner lane   
   
 g.setStroke(dashedStroke);   
 g.setColor(Color.white); g.drawRect( 100, 150, 650, 400 ); // mid-lane marker   
 g.setStroke(new BasicStroke());  
   
 g.setColor(Color.green); g.fillRoundRect( 150, 200, 550, 300 , 14, 14); //Inner grass  
   
 g.setColor(Color.white); g.drawLine( 425, 100, 425, 200 ); // start line   
   
 car1Images[carPhase[0]].paintIcon(this, g, carPosition[0], carPosition[1]);  
 car2Images[carPhase[1]].paintIcon(this, g, carPosition[2], carPosition[3]);  
 }  
  
 void rotateLeft(int car)  
 {  
 //Rotate car 22.5 degrees anti-clockwise  
 carPhase[car]--;  
 if(carPhase[car] == -1) carPhase[car] = 15;  
   
 }  
  
 void rotateRight(int car)  
 {  
 //Rotate car 22.5 degrees anti-clockwise  
 carPhase[car]++;  
 if(carPhase[car] == 16) carPhase[car] = 0;  
 }  
  
}

## Part 3

### Server

import java.io.\*;  
import java.net.\*;  
import java.util.\*;  
  
public class Server extends Thread  
{  
 //Network port number  
 public static int port = 7334;  
 //Buffer size  
 private final static int BUFFER = 1024;  
 //Message buffer  
 static byte[] buf = new byte[BUFFER];  
 //Network socket  
 private static DatagramSocket socket;  
 //List of client addresses  
 private static ArrayList<InetAddress> clientAddresses;  
 //List of client ports  
 private static ArrayList<Integer> clientPorts;  
 //List of client ID's  
 private static ArrayList<String> existingClients;  
  
 public static void main(String args[])   
 {  
 try   
 {  
 if(args.length > 0)  
 {  
 port = Integer.getInteger(args[0]);  
 }  
 //Set up server  
 socket = new DatagramSocket(port);  
 //Set up list of client addresses  
 clientAddresses = new ArrayList();  
 //Set up list of client ports  
 clientPorts = new ArrayList();  
 //Set up list of ID's refrencing clients  
 existingClients = new ArrayList();  
 //Run server  
 RunServer();  
 }   
 catch (IOException | InterruptedException ex)   
 {  
 //Print out error message  
 System.out.println(ex.getMessage());  
 }  
  
 }  
   
 private static String[] GetClientMessage() throws IOException, InterruptedException  
 {  
 //Return array in format [Client id, client message]  
 Arrays.fill(buf, (byte)0);  
   
 //Initalize data packet  
 DatagramPacket packet = new DatagramPacket(buf, buf.length);  
   
 //Recieve client message from socket to packet  
 socket.receive(packet);  
   
 //Get client address  
 InetAddress clientAddress = packet.getAddress();  
   
 //Get client Port  
 int clientPort = packet.getPort();  
   
 //Client port  
 String id = clientAddress.toString() + "," + clientPort;  
  
 //Check if client is an old one  
 if (!existingClients.contains(id))   
 {  
 //Add id to id list  
 existingClients.add(id);  
  
 //Add port to client port list  
 clientPorts.add(clientPort);  
  
 //Add address to address list  
 clientAddresses.add(clientAddress);  
   
 //Prompt server admin that client has been added  
 System.out.println(id + " Joined the network");  
   
 int intValue = existingClients.indexOf(id);  
 String message = String.valueOf(intValue);  
   
 if(intValue < 2)  
 {  
 //Send only to specific client  
 SentToClient(id, message, false);  
 }  
 else  
 {  
 //Prompt client to exit  
 SentToClient(id, "Exit", false);  
 }  
 }  
  
 var clientMessage = new String(buf, buf.length).trim();  
  
 if(!clientMessage.isEmpty())  
 {  
 System.out.println("Client: " + clientMessage);  
 return new String[]{id, clientMessage};  
 }  
 else return null;  
 }  
   
 public static void SentToClient(String id, String message, boolean toAll) throws IOException, InterruptedException  
 {  
 if(message.isEmpty()) return;  
   
 System.out.println("Server: " + message);  
  
 byte[] data = (message).getBytes();  
 for (int i=0; i < existingClients.size(); i++)   
 {  
 boolean isClient = existingClients.get(i) == null   
 ? id == null : existingClients.get(i).equals(id);  
   
 if(isClient || toAll)  
 {  
 //If sender ID then reply  
 InetAddress cl = clientAddresses.get(i);  
  
 int cp = clientPorts.get(i);  
  
 var packet = new DatagramPacket(data, data.length, cl, cp);  
  
 socket.send(packet);  
 break;  
 }  
 }  
   
 }  
   
 public static boolean GetAndRespondToClient() throws IOException, InterruptedException  
 {  
 String[] clientMessage = GetClientMessage();  
 if(clientMessage == null) return true;   
 if(clientMessage[1].isEmpty()) return true;  
   
 if (clientMessage[1].equals("exit"))  
 {  
 //Send only to specific client  
 System.out.println("Client Controling car has left the server");  
 System.out.println("Server is now shutting down ");  
 SentToClient(existingClients.get(0), clientMessage[1], false);  
 SentToClient(existingClients.get(1), clientMessage[1], false);  
 return false;  
 }  
 else if (clientMessage[1].equals("restart"))  
 {  
 //Send only to specific client  
 System.out.println("Restarting game");  
 SentToClient(existingClients.get(0), clientMessage[1], false);  
 SentToClient(existingClients.get(1), clientMessage[1], false);  
 return false;  
 }  
 else  
 {  
 //car carphase carX carY displacementFactor  
 String[] args = clientMessage[1].split(" ");  
 if(args.length != 5) return true;  
   
 int i;  
 if(args[0].equals("0")) i = 1;  
 else i = 0;  
 SentToClient(existingClients.get(i), clientMessage[1], false);  
   
 }  
 return true;  
 }  
   
 public static void RunServer() throws IOException, InterruptedException   
 {  
 System.out.println("Server Running");  
   
 while (true)   
 {  
 if(!GetAndRespondToClient()) break;  
  
 }  
  
 }  
  
  
}

### Game Client

import java.net.\*;  
  
 /\*\*  
 \*  
 \* Author: SID 1651542  
 \*/  
  
public final class GameClient  
{  
 static DatagramSocket socket;  
 public static String host = "127.0.0.1"; //Address to be used for communication  
 public static int port = 7334; //Address to be used for communication  
 static MessageReceiver messageReciever; //Class handling recieving messages  
 static Thread messageRecieverThread; //Thread dedicated to the message reciever  
   
 static MessageSender messageSender; //Class handling sending messages to the server  
 static Thread messageSenderThread; //Thread dedicated to the message sender  
   
 public static void main(String[] args) throws Exception  
 {   
 if(args.length > 0)  
 {  
 //Get port and host info if started with parameters  
 try  
 {  
 host = args[0];  
 port = Integer.valueOf(args[1]);  
 }  
 catch(NumberFormatException e)  
 {  
 System.out.println(e.getMessage());  
 }  
 }  
 //Set up socket  
 socket = new DatagramSocket();  
 System.out.println("Socket set up");  
 //Set up class for recieving messages  
 messageReciever = new MessageReceiver(socket);  
 //Set up class for sending messages  
 messageSender = new MessageSender();  
 //Set up thread handling the message reciever  
 messageRecieverThread = new Thread(messageReciever);  
 //Set up thread handling the message sender  
 messageSenderThread = new Thread(messageSender);  
 //Start message reciever  
 messageRecieverThread.start();  
 System.out.println("Recieving thread Initalized");  
 //Start message sender  
 messageSenderThread.start();  
 System.out.println("Sending thread Initalized");  
 }  
}

import java.net.DatagramPacket;  
import java.net.DatagramSocket;  
import java.net.InetAddress;  
  
class MessageSender implements Runnable   
{  
 public static boolean sendMessage(String s) throws Exception   
 {  
 if(s.isEmpty()) return true; //Stop function if message is empty  
   
 System.out.println("Client: " + s); //Display message to be sent  
   
 byte buf[] = s.getBytes(); //Get bytes from string  
  
 //Get address  
 InetAddress address = InetAddress.getByName(GameClient.host);  
  
 //Initalize packet  
 DatagramPacket packet = new DatagramPacket(buf, buf.length, address, GameClient.port);  
  
 //Send packet to destination  
 GameClient.socket.send(packet);  
   
 return !s.equals("exit"); //Keep sending kessages  
 }  
  
 @Override  
 public void run()   
 {  
 boolean connected = false;  
   
 //Wait for client to succesfully send packet  
 while(!connected)  
 {  
 try   
 {  
 //Send greeting message  
 sendMessage("Hello");  
 //Set connection status to true  
 connected = true;  
  
 } catch (Exception e) {}  
 }  
  
 System.out.println("Ready to revieve from network");  
   
 while (true)   
 {  
 try   
 {  
 boolean cont = sendMessage(StaticData.GetClientMessage()); //Send message to server  
 //Close socket if sent message is exit  
 if(!cont)break;  
 //Pause process for a second  
 }   
 catch(Exception e) {System.err.println(e); }  
 }  
  
 }  
  
}  
import java.io.IOException;  
import java.net.DatagramPacket;  
import java.net.DatagramSocket;  
  
class MessageReceiver implements Runnable   
{  
 static byte buffer[]; //Message buffer  
  
 MessageReceiver(DatagramSocket s)   
 {  
 //Set up buffer  
 buffer = new byte[1024];  
 }  
   
 public static String Recieve()  
 {  
 try   
 {  
 //Initalize packet  
 DatagramPacket packet = new DatagramPacket(buffer, buffer.length);  
 //Recieve data from network into packet  
 GameClient.socket.receive(packet);  
 //Convert pack data to string  
 String serverMessage = new String(packet.getData(), 0, packet.getLength());  
 //Return recieved message  
 return serverMessage;  
  
 } catch (IOException e) {}  
   
 return null;  
 }  
   
 private static boolean RespondToServer(String serverMessage) throws Exception  
 {  
   
 switch (serverMessage) {  
 case "0":  
 if(StaticData.GetCarIndex() != 0)  
 StaticData.SetCar(0);  
 System.err.println("Allocated white car");  
 StaticData.waitForRecieve = true;  
 StaticData.GetForm();  
 break;  
 case "1":  
 if(StaticData.GetCarIndex() != 1)  
 StaticData.SetCar(1);  
 System.err.println("Allocated blue car");  
 StaticData.GetForm();  
 break;  
 case "restart":  
 StaticData.GetForm().Restart();  
 break;  
 case "exit":  
 //Close Application. Server full(Already controling two cars)  
 System.out.println("Server down");  
 System.out.println("Server closing");  
 StaticData.GetForm().dispose();  
 System.exit(0);  
 return false; //Stop running  
 default:  
 //Split server message  
 String[] args = serverMessage.split(" ");  
 //Use array created as details for new car details  
 if(args != null && args.length == 5)  
 StaticData.GetForm().MakeChanges(args);  
 break;  
 }  
   
 return true; //Continue running   
 }  
   
 public static boolean RecieveAndRespond() throws Exception  
 {  
 //Recieve message from server  
 String serverMessage = Recieve();   
 //Display server message  
 if(!serverMessage.isEmpty())  
 System.out.println("Server: " + serverMessage);  
 //return continue value  
 return RespondToServer(serverMessage);  
 }  
  
 @Override  
 public void run() {  
  
 try   
 {  
 //Recieve messages from server and respond indefinitly  
 while (RecieveAndRespond()) {}  
 }   
 catch (Exception ex)   
 {  
 //Display error message  
 System.err.println("Messages can no longer recieve messages");  
 }  
 }  
}

import java.awt.event.\*;  
import javax.swing.\*;  
  
/\*\*  
 \*  
 \* Author: SID 1651542  
 \*/  
public class Form extends JFrame   
{  
 //Panel that will be used for all graphical work in the program  
 Palette graphicsPalette;  
 public Form()  
 {  
 graphicsPalette = new Palette();  
   
 //initalize graohicsPalette  
 graphicsPalette.setFocusable(true);  
   
 //fill form with graphics palette  
 graphicsPalette.setSize(WIDTH, HEIGHT);  
 add(graphicsPalette);  
 setVisible(true);  
   
 //Set form size 850 pixels by 650 pixels  
 setSize(850, 650);  
 setResizable(false);  
   
 //Alow form to close on the pressing of "X"  
 setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);  
 setVisible(true);  
   
 addWindowListener(new WindowAdapter()   
 {  
 @Override  
 public void windowClosing(WindowEvent e)   
 {  
 try {  
 //Close other clients  
 MessageSender.sendMessage("exit");  
 } catch (Exception ex) {}  
 }  
 });  
 }  
   
 public void Restart()  
 {  
 graphicsPalette.Restart();  
 }  
   
 public void MakeChanges(String[] str) throws Exception  
 {  
 //Change other car details  
 graphicsPalette.MakeChanges(str);  
 }  
}