//서버

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Collections;
using System.Threading;
using System.IO;
using System.Net;
using System.Net.Sockets;
namespace PaintChat_Server
{
    public partial class Form1 : Form
        {
                // 지연 함수
                // 함수에 딜레이를 걸어서 다음 내용이 딜레이가 지난 후에 실행되도록 함
                public static DateTime Delay(int MS)
                {
                        DateTime ThisMoment = DateTime.Now;
                        TimeSpan duration = new TimeSpan(0, 0, 0, 0, MS);
                        DateTime AfterWards = ThisMoment.Add(duration);
                        while (AfterWards >= ThisMoment)
                        {
                                System.Windows.Forms.Application.DoEvents();
                                ThisMoment = DateTime.Now;
                        }
                        return DateTime.Now;
                }
#region Work data
                private const int SEND_HINT_DELAY = 10000;    // 힌트 보낼시 딜레이
                // 작업되어 지는 데이터를 기록하는 구조체
                struct Work
                {
                        public Point current_;
                        public Point previous_;
                        public Color penColor_;
                        public int penWidth_;
                        public int penDashStyle_;
                }
                private List<Work> workList_;                    // 이전 작업 데이터
#endregion
#region Game data
                struct GameData
```

```
        {
            public string question_;                    // 문제
            public string hint_;                        // 힌트
            public bool isPlayingGame_;                 // 게임 진행 여부
            public void reset()
            {
                question_ = "";
                hint_ = "";
                isPlayingGame_ = false;
            }
        }
        private GameData gameData_;                     // 게임에 필요한 데이터
        private bool isPlayingGame() { return gameData_.isPlayingGame_; }
#endregion
#region Network data
        // 네트워크 관련.
        private TcpListener Server;                     // TCP 네트워크 클라이언트에서 연결 수신
        private TcpClient SerClient;                    // TCP 네트워크 서비스에 대한 클라이언트 연결 제공
        private Boolean Start = false;                  // 서버 시작
        private int myPort;                                     // 포트
        private string myName;                          // 별칭(ID)
        private Thread myServer;                        // 스레드
        private Boolean TextChange = false;             // 입력 컨트롤의 데이터입력 체크
        private List<Client> connectClients_;       // 접속 중인 클라이언트들
#endregion
#region Winform data
        // Winform 동작에 필요한 데이터.
        private bool isDrag = false;                    // 현재 드래그 중인지
        private int setTool = 0;                        // 펜 사이즈
        private int setStrip = 0;                       // 펜 스타일
        private int setColor = 0;                       // 펜 색상
        private ColorDialog clg;                        // 펜 색상을 결정하는 다이얼로그
        private bool isRemoveMode_ = false;             // 그리기/지우개 여부
        private Pen currentPen;                         // 현재 활성화 된 펜
        private Graphics currentGraphics_;              // 그림판의 그래픽 접근 변수
        private Point previousPoint;                    // 이전 포지션
        private Point currentPoint_;                    // 현재 포지션
        private Form2 form2_;                           // 문제 내기에 필요한 다이얼로그
#endregion
#region Singleton pattern
        // 싱글톤 패턴을 이용한 전역 접근.
        private static Form1 instance_;
        public static Form1 instance { get { return instance_;  } }
#endregion
        public Form1()
```

```
        {
            InitializeComponent();
                        instance_ = this;
            currentPen = new Pen(Color.Black);
            currentGraphics_ = panel1.CreateGraphics();
                        connectClients_ = new List<Client>();
                        workList_ = new List<Work>();
                        form2_ = new Form2();
                        form2_.Visible = false;
                        문제ToolStripMenuItem.Enabled = false;
        }
        private void 새로만들기NToolStripMenuItem_Click(object sender, EventArgs e)
        {
            panel1.Invalidate();  // 현재 패널에 그려진 그림들을 모두 지움
                        if (Start)
                                NewFile_Send();          // 서버가 가동중인 상태라면, 새로운 파일을 만
들었다고, 접속중인 유저에게 모두 알림
                        workList_.Clear();               // 현재까지 작업된 모든 Work 들을 초기화
        }
        private void 열기OToolStripMenuItem_Click(object sender, EventArgs e)
        {
                        // 다이얼로그를 띄워서 해당 경로의 그림파일을 가져옴
            OpenFileDialog openPanel = new OpenFileDialog();
            openPanel.InitialDirectory = @"C:₩";
            openPanel.Filter = "PNG (*.png)|*.png| All files (*.*)|(*.*)";
            if (openPanel.ShowDialog() == DialogResult.OK)
            {
                Bitmap bitmap = new Bitmap(openPanel.FileName);
                Graphics g = panel1.CreateGraphics();
                g.DrawImage(bitmap, 0, 0);
            }
        }
        private void 저장ToolStripMenuItem_Click(object sender, EventArgs e)
        {
                        // 패널의 그림을 비트맵으로 만듦.
            Bitmap bitmap = new Bitmap(panel1.Width, panel1.Height);
            Graphics g = Graphics.FromImage(bitmap);
            Size size = panel1.Size;
            g.CopyFromScreen(
                PointToScreen(new Point(panel1.Location.X, panel1.Location.Y)),
                new Point(0, 0),
                size);
                        // 다이얼로그를 띄워서 경로를 직접 지정하고, 해당 경로에 현재 그림파일을 저장
함
            SaveFileDialog savePanel = new SaveFileDialog();
            savePanel.InitialDirectory = @"C:₩";
            savePanel.Filter = "PNG (*.png)|*.png| All files (*.*)|(*.*)";
            if (savePanel.ShowDialog() == DialogResult.OK)
```

```
    {
        bitmap.Save(savePanel.FileName, System.Drawing.Imaging.ImageFormat.Png);
    }
}
private void 종료XToolStripMenuItem_Click(object sender, EventArgs e)
{
    Application.Exit();
}
private void 그리기ToolStripMenuItem_Click(object sender, EventArgs e)
{
    isRemoveMode_ = false;
}
private void 지우개ToolStripMenuItem_Click(object sender, EventArgs e)
{
    isRemoveMode_ = true;
}
private void 굵기10ToolStripMenuItem_Click(object sender, EventArgs e)
{
    setTool = 1;
}
private void 굵기5ToolStripMenuItem_Click(object sender, EventArgs e)
{
    setTool = 2;
}
private void 굵기3ToolStripMenuItem_Click(object sender, EventArgs e)
{
    setTool = 3;
}
private void 굵기1ToolStripMenuItem_Click(object sender, EventArgs e)
{
    setTool = 0;
}
private void dotToolStripMenuItem_Click(object sender, EventArgs e)
{
    setStrip = 1;
}
private void dashDotToolStripMenuItem_Click(object sender, EventArgs e)
{
    setStrip = 2;
}
private void soildToolStripMenuItem_Click(object sender, EventArgs e)
{
    setStrip = 0;
}
private void 선LToolStripMenuItem_Click(object sender, EventArgs e)
{
    clg = new ColorDialog();
```

```
        clg.ShowDialog();
        setColor = 1;
        // Set the initial color of the dialog to the current text color
    }
    private void panel1_MouseDown(object sender, MouseEventArgs e)
    {
        isDrag = true;
        previousPoint = new Point(e.X, e.Y); // 이전 위치를 기록해둠
    }
    private void panel1_MouseUp(object sender, MouseEventArgs e)
    {
        isDrag = false;
    }
    private void panel1_MouseMove(object sender, MouseEventArgs e)
    {
        if (isDrag == true)  // 드래그 중일시에만 작동
        {
            currentPoint_ = new Point(e.X, e.Y);      // 현재 마우스의 위치값을 저장
            // 설정된 펜 색상을 적용
                            if (setColor == 0)
                currentPen.Color = Color.Black;
            else
                currentPen.Color = clg.Color;

                            // 설정된 펜 굵기를 적용
            if (setTool == 0)
                currentPen.Width = 1;
            else if (setTool == 1)
                currentPen.Width = 10;
            else if (setTool == 2)
                currentPen.Width = 5;
            else if (setTool == 3)
                currentPen.Width = 3;
                            // 설정된 펜 스타일을 적용
            if (setStrip == 0)
                currentPen.DashStyle = System.Drawing.Drawing2D.DashStyle.Solid;
            else if (setStrip == 1)
                currentPen.DashStyle = System.Drawing.Drawing2D.DashStyle.Dot;
            else if (setStrip == 2)
                currentPen.DashStyle = System.Drawing.Drawing2D.DashStyle.DashDot;
                            // 지우개 모드일 시에 색상, 굵기, 스타일을 강제 적용
            if (isRemoveMode_)
            {
                currentPen.Color = Color.White;
                currentPen.Width = 60;
                currentPen.DashStyle = System.Drawing.Drawing2D.DashStyle.Solid;
            }
```

```
                    // 이전 위치에서부터 현재 위치까지 선을 그려줌
                currentGraphics_.DrawLine(currentPen, previousPoint, currentPoint_);

                    // 서버가 작동중 이라면, 접속중인 모든 클라이언트에게 현재 그려진 데이
터를 전송
                    if (Start)
                        DrawData_Send();
                    // 현재 위치를 다시 이전 위치로 저장
                previousPoint = currentPoint_;
            }
        }
        private void panel1_Paint(object sender, PaintEventArgs e)
        {
        }
        private void SendMessage_Click(object sender, EventArgs e)
        {
            if (this.textMessage.Text == "")
            {
                this.textMessage.Focus();
            }
            else
            {
                Msg_send(); // Msg_send()함수 호출
            }
        }
        private void DrawData_Send()
        {
            try
            {
                    // 데이터 전송을 위해서 Point 구조체로 만들어진 위치 데이터를 int형으로
바꾸어줌
                    int currentX = Math.Abs(currentPoint_.X - panel1.Location.X);
                    int currentY = Math.Abs(currentPoint_.Y - panel1.Location.Y);
                    int previousX = Math.Abs(previousPoint.X - panel1.Location.X);
                    int previousY = Math.Abs(previousPoint.Y - panel1.Location.Y);
                    // 현재 적용된 작업을 기록
                    Work work;
                    work.previous_ = new Point(previousX, previousY);
                    work.current_ = new Point(currentX, currentY);
                    work.penColor_ = currentPen.Color;
                    work.penWidth_ = (int)currentPen.Width;
                    work.penDashStyle_ = (int)currentPen.DashStyle;
                    workList_.Add(work);
                    // 실제 보내질 데이터를 가공함
                    string data = "S001&" + currentX + "," + currentY     // currentPoint.
                                  + "&" + previousX + "," + previousY // before
position.
                + "&" + currentPen.Color.R + "," + currentPen.Color.G + "," +
```

```
currentPen.Color.B + "," + currentPen.Color.A // Pen color.
                        + "&" + (int)currentPen.Width // Pen size.
                        + "&" + (int)currentPen.DashStyle; // Pen style.
                            // 데이터 전송
                            for (int index = 0; index < connectClients_.Count; ++index)
                                    connectClients_[index].writeData(data);
                            // 서버의 rtbText(채팅창)에 id와 메시지 출력
                this.textMessage.Clear();
            }
            catch
            {
                MessageView("데이터를 보내는 동안 오류가 발생하였습니다.");
                this.textMessage.Clear();
            }
        }
                private void NewFile_Send()
                {
                        try
                        {
                                string data = "S002&" + "New";
                                for (int index = 0; index < connectClients_.Count; ++index)
                                        connectClients_[index].writeData(data);
                                // 서버의 rtbText(채팅창)에 id와 메시지 출력
                                this.textMessage.Clear();
                        }
                        catch
                        {
                                MessageView("데이터를 보내는 동안 오류가 발생하였습니다.");
                                this.textMessage.Clear();
                        }
                }
                public void Msg_Send_Clients(string senderId, string value)
                {
                        try
                        {
                                // 아이디 & 메시지 & 날짜시간을 보냄
                                for (int index = 0; index < connectClients_.Count; ++index)
                                {
                                        // 메세지를 보낸 유저를 제외한 다른 접속중인 유저에게 모두 보
냄
                                        if (connectClients_[index].getID() != senderId)
                                                connectClients_[index].writeData(value);
                                }
                        }
                        catch
                        {
                                MessageView("데이터를 보내는 동안 오류가 발생하였습니다.");
```

```
                    this.textMessage.Clear();
                }
            }
        private void Msg_send()
        {
            try
            {
                var dt = Convert.ToString(DateTime.Now);
                // 아이디 & 메시지 & 날짜시간을 보냄
                    for (int index = 0; index < connectClients_.Count; ++index)
                        connectClients_[index].writeData("Server" + "&" +
this.textMessage.Text + "&" + dt);
                    // 서버의 rtbText(채팅창)에 id와 메시지 출력
                    MessageView("Server" + ": " + this.textMessage.Text, Color.Blue);
                this.textMessage.Clear();
            }
            catch
            {
                MessageView("데이터를 보내는 동안 오류가 발생하였습니다.");
                this.textMessage.Clear();
            }
        }
        private void SetButton_Click(object sender, EventArgs e)
        {
            LoginPanel.Visible = false;
        }
        private void SetCancleButton_Click(object sender, EventArgs e)
        {
            LoginPanel.Visible = false;
        }
        private void iDPORTToolStripMenuItem_Click(object sender, EventArgs e)
        {
            LoginPanel.Visible = true;
        }
        private void connectionToolStripMenuItem_Click(object sender, EventArgs e)
        {
            var addr = new IPAddress(0);// IPAddress 클래스의 개체를 초기화, 매개변수가 0->로컬단말
의 아이피 가져옴
            this.myName = this.textID.Text;
            this.myPort = Convert.ToInt32(this.textPort.Text);
            if (!(this.Start))// 서버가 시작되지 않은 경우
            {
                try
                {
                    Server = new TcpListener(addr, this.myPort);                    // ip주소와 포트번호를
인자로 TcpListener의 개체 생성
                    Server.Start();
                        // Server 시작
```

```
                this.Start = true;
                this.textMessage.Enabled = true;
                this.SendMessageButton.Enabled = true;
                this.textMessage.Focus();                                   // 메시지를 쓸 수 있도록 초점을 이동
                this.disconnectToolStripMenuItem.Enabled = true;            // 연결을 끊을 수 있도록 메뉴 활성화
                this.connectionToolStripMenuItem.Enabled = false;           // 연결시도버튼 비활성화
                // Thread클래스의 생성자를 이용하여 개체생성-> ServerStart()메서드로 클라이언트의 수신과 네트워크 스트림의 값을 수신하는 작업을 새로 생성한 스레스에서 수행
                myServer = new Thread(ServerStart);
                myServer.Start();
                this.iDPORTToolStripMenuItem.Enabled = false;
            }
            catch
            {
                MessageView("서버를 실행할 수 없습니다.");
            }
        }
        else
        {
            ServerStop(); //ServerStop() 함수 호출
        }
    }
    private void ServerStart()
    {
        MessageView("서버 실행 : 챗 상대의 접속을 기다립니다...");
        while (Start) // 클라이언트가 접속될 때까지 기다림
        {
            try
            {
                                SerClient = Server.AcceptTcpClient(); // 보류중인 연결요청을 받아들임 ( 비동기 방식이므로, 이 함수는 여기에서 정지함 )

                                // 접속했다면, 새로운 클라이언트 정보를 만들어서 접속중인 클라이언트로 추가함
                                Client newClient = new Client(SerClient);
                                connectClients_.Add(newClient);
                                MessageView("챗 상대 접속..");
                                // 클라이언트가 1명이상일 때 게임시작 가능.
                                if (connectClients_.Count >= 1)
                                        문제ToolStripMenuItem.Enabled = true;
            }
            catch { }
        }
    }
            public void disconnectClient(Client client)
            {
```

```
                    // 클라이언트가 접속을 끊었다면, 접속중인 클라이언트 정보를 알리고, 제거함
                    for (int index=0; index<connectClients_.Count; ++index)
                    {
                        if (connectClients_[index].getID() == client.getID())
                        {
                            connectClients_.Remove(client);
                            client.close();
                            break;
                        }
                    }
                    // 클라이언트가 1명도 없다면, 게임을 시작할 수 없다.
                    if (connectClients_.Count < 1)
                    {
                        // 데이터 초기화.
                        ResetData();
                        form2_.clearQustionData();
                    }
                }
                public void MessageView(string strText, Color color = default(Color))
                {
                    if (color == default(Color))
                        color = Color.Black;
                    // 지정된 문구만 컬러 설정.
                    string text = strText + "₩r₩n";
                    int currentLength = this.chatBox.TextLength;      // 현재까지 채팅박스에 기록된
모든 텍스트의 길이
                    this.chatBox.AppendText(text);                                  // 입력한 텍스
트와 엔터키 추가
                    this.chatBox.SelectionStart = currentLength;      // 최종 텍스트의 길이부터 선택
시작
                    this.chatBox.SelectionLength = text.Length;             // 현재 작성하려고 하
는 메세지의 길이만큼 선택을 종료
                    this.chatBox.SelectionColor = color;                    // 선택된 메세지만 색
상을 적용함
                    this.chatBox.Focus();                                                   //  초
점이동
                    this.chatBox.ScrollToCaret();                                   // 컨트롤의 내
용을 현재 캐럿 위치까지 스크롤
                    this.textMessage.Focus();                                               //  초
점이동
                }

        private void ServerStop() // 서버 종료
        {
            this.Start = false;
            this.textMessage.Enabled = false;
            this.textMessage.Clear();
            this.SendMessageButton.Enabled = false;
            this.connectionToolStripMenuItem.Enabled = true;
```

```
            this.disconnectToolStripMenuItem.Enabled = false;
                        // 데이터 초기화.
                        ResetData();
                        // 서버 종료를 모든 클라이언트에 알리고, 클라이언트의 접속을 강제 종료함
                        Msg_Send_Clients("", "S998&서버 종료");
                        for (int index = 0; index < connectClients_.Count; ++index)
                                connectClients_[index].close();
                        connectClients_.Clear();
            if (SerClient != null)
                                SerClient.Close();          // TcpClient 클래스 개체 리소스 해제
            if (Server != null)
                                Server.Stop();              // TcpListen 클래스 개체 리소스 해제
            if (myServer != null)
                                myServer.Abort();           // 외부 스레드 종료
            MessageView("연결이 끊어졌습니다.");
        }
        private void disconnectToolStripMenuItem_Click(object sender, EventArgs e)
        {
            try
            {
                if (this.SerClient.Connected)       // 연결중인 상태
                {
                    var dt = Convert.ToString(DateTime.Now);
                                        for (int index = 0; index < connectClients_.Count; ++index)
                                                connectClients_[index].writeData("Server" + "&" + "채팅
APP가 종료되었습니다." + "&" + dt);
                }
            }
            catch { }
            ServerStop();           // 서버종료
            this.iDPORTToolStripMenuItem.Enabled = true;
            this.connectionToolStripMenuItem.Enabled = true;
        }
        private void textMessage_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (e.KeyChar == (char)13)   // 엔터 키를 누를때
            {
                e.Handled = true;                   // 소리 없앰
                if (this.textMessage.Text == "")
                {
                    this.textMessage.Focus();
                }
                else
                {
                    Msg_send();                     // Msg_send()함수 호출
                }
            }
```

```
        }
        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            ServerStop();          // 서버종료
        }
        private void toolStripButton1_Click(object sender, EventArgs e)
        {
        }
                private void 출제ToolStripMenuItem_Click(object sender, EventArgs e)
                {
                        form2_.Visible = true;
                }
                public void gameStart(string question, string hint)
                {
                        // 게임 데이터 설정
                        gameData_.question_ = question;
                        gameData_.hint_ = hint;
                        gameData_.isPlayingGame_ = true;

                        // 다이얼로그 끄기
                        문제ToolStripMenuItem.Enabled = false;
                        // 서버 및 클라이언트에 메세지 띄우기
                        string message = "[게임 시작]& 문제가 출제되었습니다.";
                        for (int index = 0; index < connectClients_.Count; ++index)
                                connectClients_[index].writeData(message);
                        MessageView(message.Replace("&", ""), Color.Red);
                        // 해당 초 만큼 지난뒤에 힌트 발송
                        message = "Server& 최초 힌트는 " + (SEND_HINT_DELAY/1000) + "초 후에 보여
집니다.";
                        for (int index = 0; index < connectClients_.Count; ++index)
                                connectClients_[index].writeData(message);
                        MessageView(message.Replace("&", " :"), Color.Blue);
                        Delay(SEND_HINT_DELAY);
                        sendHintMessage();
                }
                private void sendHintMessage()
                {
                        // 힌트를 접속중인 모든 유저에게 보낸다
                        string message = "[Hint]& " + gameData_.hint_;
                        for (int index = 0; index < connectClients_.Count; ++index)
                                connectClients_[index].writeData(message);
                        MessageView(message.Replace("&", ""), Color.Red);
                }
                public void checkRightAnswer(string id, string message)
                {
                        // 게임중이 아니라면 검사할 필요가 없다
                        if (isPlayingGame() == false)
```

```
                        return;
                // 정답이라면 게임을 끝냄
                if (gameData_.question_ == message)
                        onGameClear(id);
        }
        private void onGameClear(string id)
        {
                // 게임 플레이 종료
                gameData_.isPlayingGame_ = false;

                // 정답자를 서버와 접속중인 모든 유저에게 알림
                string message = "축하합니다. " + id + "님이 정답을 맞추셨습니다.";
                MessageView(message, Color.Blue);
                for (int index = 0; index < connectClients_.Count; ++index)
                        connectClients_[index].writeData("Server&" + message);
                // 게임이 종료되었음을 서버와 접속중인 모든 유저에게 알림
                message = "게임이 종료되었습니다";
                MessageView(message, Color.Blue);
                for (int index = 0; index < connectClients_.Count; ++index)
                        connectClients_[index].writeData("Server&" + message);
                // 게임 데이터 초기화
                ResetData();
                // 클라이언트가 1명이상일 때 게임시작 가능
                if (connectClients_.Count >= 1)
                        문제ToolStripMenuItem.Enabled = true;
        }
        private void ResetData()
        {
                // 게임 플레이 종료
                gameData_.isPlayingGame_ = false;
                // 새로운 파일로 초기화
                panel1.Invalidate();
                NewFile_Send();
                workList_.Clear();
                // 데이터 리셋
                gameData_.reset();
                // 문제메뉴 비활성화
                문제ToolStripMenuItem.Enabled = false;
        }
    }
}


using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
```

```
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace PaintChat_Server
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();
        }
                //문제 데이터를 초기화
                public void clearQustionData()
                {
                        QuetionBox.Clear();
                        HintBox.Clear();
                }
                //게임시작 버튼 이벤트 함수.
                private void GameStartButton_Click(object sender, EventArgs e)
                {
                        this.Visible = false;
                        Form1.instance.gameStart(QuetionBox.Text, HintBox.Text);
                        clearQustionData();
                }
                //문제 내기 취소 버튼 이벤트 함수.
                private void QuestionCancleButton_Click(object sender, EventArgs e)
                {
                        this.Visible = false;
                        clearQustionData();
                }
    }
}

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Collections;
using System.Threading;
using System.IO;
using System.Net;
using System.Net.Sockets;
namespace PaintChat_Server
```

```
{
        // 유저 정보
        // 각 유저의 아이디 및 네트워크 통신 관련 스트림 변수들을 관리
        public class Client
        {
                private string id_;                                     // 클라이언트 아이디
                private TcpClient client_;          // TCP 네트워크 서비스에 대한 클라이언트 연결 제공
                private NetworkStream stream_;   // 네트워크 스트림
                private StreamReader reader_;    // 스트림 읽기
                private StreamWriter writer_;      // 스트림 쓰기
                private bool clientCon_;           // 클라이언트 시작
                private Thread receiveThread_;   // 데이터 받기 쓰레드
                private bool isMaster_;                    // 문제 출제자 여부
                public Client(TcpClient client)
                {
                        id_ = "Empty";
                        client_ = client;
                        stream_ = client_.GetStream();
                        reader_ = new StreamReader(stream_);
                        writer_ = new StreamWriter(stream_);
                        clientCon_ = true;
                        receiveThread_ = new Thread(receive);
                        receiveThread_.Start();
                        isMaster_ = false;
                }
                public string getID() { return id_; }                                   // 클라이언트 ID
                public bool isClientCon() { return clientCon_; }  // 클라이언트 접속 여부
                public void setMaster() { isMaster_ = true; }             // 문제 출제자 여부
                // 전송받은 스트림에서 데이터를 읽어들임.
                public string readData()
                {
                        return reader_.ReadLine();
                }
                // 데이터를 스트림에 작성하고, 보냄
                public void writeData(string value)
                {
                        writer_.WriteLine(value);  // 스트림에 데이터 작성
                        writer_.Flush();                    // 스트림 버퍼를 지우고, 스트림 버퍼에 존재한
데이터를 내부 스트림에 씀(전송)
                }
                // 접속 종료시 호출되는 콜백함수
                public void close()
                {
                        clientCon_ = false;                         // 접속 여부를 비접속으로
                        receiveThread_.Abort();           // 데이터 리시브 스레드 종료.
                        receiveThread_ = null;
                        if (reader_ != null)
```

```
                reader_.Close(); // StreamReader 클래스 개체 리소스 해제
            if (writer_ != null)
                writer_.Close(); // StreamWriter 클래스 개체 리소스 해제
            if (stream_ != null)
                stream_.Close(); // NetworkStream 클래스 개체 리소스 해제
        }
        // 스트림에서 데이터 읽기 가능 여부
        public bool CanRead()
        {
            return stream_.CanRead;
        }
        // 서버 및 클라이언트 모드에서 myReader 스레드 개체에서 실행되는 메서드로 메시지를 받은 데이터를 화면에 출력하는 작업을 수행
        private void receive()
        {
            try
            {
                while (this.clientCon_)// 클라이언트의 연결이 종료될때까지 계속 실행
                {
                    if (CanRead())// 스트림에서 데이터를 읽을 수 있는 경우
                    {
                        var msg = readData();// id&메세지&보낸날짜시간
                        var Smsg = msg.Split('&');// &를 기준으로 메시지 구분
                        if (Smsg[0] == "S001")// 첫 구분자가  "S001"인 경우
                        {
                        }
                        else if (Smsg[0] == "S002")
                        {
                        }
                        else if (Smsg[0] == "S003")
                        {
                        }
                        else if (Smsg[0] == "S999") // 클라이언트 접속 종료.
                        {
                            Form1.instance.MessageView(Smsg[1] + " 접속 종료");
                            Form1.instance.disconnectClient(this);
                        }
                        else if (Smsg[0] == "S998") // 클라이언트 접속.
                        {
                            id_ = Smsg[1];
                            Form1.instance.MessageView("접속  ID  :  "  + id_);     // 접속 클라이언트를 서버 메세지 창에 띄움.
                        }
                        else
                        {
                            if (msg.Length > 0)  // 읽은 메시지가 있는 경우
```

```
                                                {
                                                    Form1.instance.MessageView(Smsg[0]
+ " : " + Smsg[1]);     // 메세지 내용 서버에 출력.
                                                    Form1.instance.Msg_Send_Clients(id_,
msg);                           // 만약 클라이언트에게 송신된 메세지라면, 다른 클라이언트에게도 똑같이
전송.
                                                    Form1.instance.checkRightAnswer(id_,
Smsg[1]);                       // 게임이 시작되었고, 문제가 출제되었다면, 정답 확인.
                                                }
                                            }
                                        }
                                    }
                                catch { }
                            }
                        }
}
```

//클라

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Collections;
using System.Threading;
using System.IO;
using System.Net;
using System.Net.Sockets;
namespace PaintChat_Client
{
    public partial class Form1 : Form
    {

                // Winform 동작에 필요한 데이터.
                private bool isDrag = false;                    // 현재 드래그 중인지
                private int setTool = 0;                        // 펜 사이즈
                private int setStrip = 0;                       // 펜 스타일
                private int setColor = 0;                       // 펜 색상
                private ColorDialog clg;                        // 펜 색상을 결정하는 다이얼로그
                private bool isRemoveMode_ = false;             // 그리기/지우개 여부
                private Pen currentPen;                                 // 현재 활성화 된 펜
                private Graphics currentGraphics_;              // 그림판의 그래픽 접근 변수
                private Point previousPoint;                    // 이전 포지션
                private Point currentPoint_;                    // 현재 포지션
        // 네트워크 관련.
        private TcpClient client;                       // TCP 네트워크 서비스에 대한 클라이언트 연결
제공
        private NetworkStream myStream;                         // 네트워크 스트림
        private StreamReader myRead;                    // 스트림 읽기
        private StreamWriter myWrite;                   // 스트림 쓰기
        private Boolean ClientCon = false;              // 클라이언트 시작
                                                                // 포트
        private int myPort;
        private string myName;                                  // 별칭
        private Thread myReader;                                // 스레드
        private Boolean TextChange = false;             // 입력 컨트롤의 데이터입력 체크
        public Form1()
        {
            InitializeComponent();
            currentPen = new Pen(Color.Black);
            currentGraphics_ = panel1.CreateGraphics();
        }
```

```
private void panel1_MouseDown(object sender, MouseEventArgs e)
{
}
private void panel1_MouseUp(object sender, MouseEventArgs e)
{
}
private void panel1_MouseMove(object sender, MouseEventArgs e)
{
}
private void panel1_Paint(object sender, PaintEventArgs e)
{
}
private void SendMessage_Click(object sender, EventArgs e)
{
    if (this.textMessage.Text == "")
    {
        this.textMessage.Focus();
    }
    else
    {
        Msg_send(); // Msg_send()함수 호출
    }
}
private void Msg_send()
{
    try
    {
        var dt = Convert.ToString(DateTime.Now);
        // 아이디 & 메시지 & 날짜시간을 보냄
        myWrite.WriteLine(this.myName + "&" + this.textMessage.Text + "&" + dt);
        myWrite.Flush();
        // 서버의 rtbText(채팅창)에 id와 메시지 출력
        MessageView(this.myName + ": " + this.textMessage.Text);
        this.textMessage.Clear();
    }
    catch
    {
        MessageView("데이터를 보내는 동안 오류가 발생하였습니다.");
        this.textMessage.Clear();
    }
}
private void SetButton_Click(object sender, EventArgs e)
{
    LoginPanel.Visible = false;
}
private void SetCancleButton_Click(object sender, EventArgs e)
{
```

```
            LoginPanel.Visible = false;
        }
        private void iDPORTToolStripMenuItem_Click(object sender, EventArgs e)
        {
            LoginPanel.Visible = true;
        }
        private void connectionToolStripMenuItem_Click(object sender, EventArgs e)
        {
            var addr = new IPAddress(0);                                    // IPAddress 클래스의 개체를
초기화, 매개변수가 0->로컬단말의 아이피 가져옴
            this.myName = textID.Text;
            this.myPort = Convert.ToInt32(this.textPort.Text);
            if (!(this.ClientCon))
            {
                ClientConnection();                                         // ClientConnection()
함수 호출
            }
            else
            {
                this.textMessage.Enabled = false;
                this.SendMessageButton.Enabled = false;
                Disconnection();                                            // 함수 호출
            }
        }
        private void ClientConnection()
        {
            try
            {
                                // 접속되었다면, 데이터들을 적용 및 초기화함
                client = new TcpClient(this.textIP.Text, this.myPort);
                MessageView("서버에 접속 했습니다.");
                myStream = client.GetStream();
                myRead = new StreamReader(myStream);
                myWrite = new StreamWriter(myStream);
                this.ClientCon = true;
                this.connectionToolStripMenuItem.Enabled = false;
                this.disconnectToolStripMenuItem.Enabled = true;
                this.textMessage.Enabled = true;
                this.SendMessageButton.Enabled = true;
                this.textMessage.Focus();
                                // 접속이 성공했음을 서버에 알림
                                myWrite.WriteLine("S998&" + this.myName);
                                myWrite.Flush();
                                // 서버로부터 데이터를 전송받을 리시브 스레드를 생성함
                myReader = new Thread(Receive);
                myReader.Start();
            }
            catch
```

```
            {
                this.ClientCon = false;
                MessageView("서버에 접속하지 못 했습니다.    ");
            }
        }
        private void MessageView(string strText, Color color = default(Color))
        {
                        if (color == default(Color))
                                color = Color.Black;
                        // 지정된 문구만 컬러 설정.
                        string text = strText + "₩r₩n";
                        int currentLength = this.chatBox.TextLength;      // 현재까지 채팅박스에 기록된
모든 텍스트의 길이
                        this.chatBox.AppendText(text);                                  // 입력한 텍스
트와 엔터키추가
                        this.chatBox.SelectionStart = currentLength;      // 최종 텍스트의 길이부터 선택
시작
                        this.chatBox.SelectionLength = text.Length;               // 현재 작성하려고 하
는 메세지의 길이만큼 선택을 종료
                        this.chatBox.SelectionColor = color;                      // 선택된 메세지만 색
상을 적용함

                        this.chatBox.Focus();                                                   //  초
점이동
            this.chatBox.ScrollToCaret();                                 // 컨트롤의 내용을 현재 캐럿
위치까지 스크롤
            this.textMessage.Focus();                                             // 초점이동
        }

        private void Disconnection()
        {
            this.ClientCon = false;
            this.textMessage.Enabled = false;
            this.textMessage.Clear();
            this.SendMessageButton.Enabled = false;
            this.connectionToolStripMenuItem.Enabled = true;
            this.disconnectToolStripMenuItem.Enabled = false;
            try
            {
                                myWrite.WriteLine("S999&" + this.myName);
                                myWrite.Flush();
                if (!(myRead == null))
                {
                    myRead.Close(); // StreamReader 클래스 개체 리소스 해제
                }
                if (!(myWrite == null))
                {
                    myWrite.Close(); // StreamWriter 클래스 개체 리소스 해제
                }
```

```
                if (!(myStream == null))
                {
                    myStream.Close(); // NetworkStream 클래스 개체 리소스 해제
                }
                if (!(client == null))
                {
                    client.Close(); // TcpClient 클래스 개체 리소스 해제
                }
                if (!(myReader == null))
                {
                    myReader.Abort(); // 외부 스레드 종료
                }
                MessageView("연결이 끊어졌습니다.    ");
            }
            catch
            {
                return;
            }
        }
        // 서버 및 클라이언트 모드에서 myReader 스레드 개체에서 실행되는 메서드로 메시지를 받은 데이터를 화면에 출력하는 작업을 수행
        private void Receive()
        {
            try
            {
                while (this.ClientCon) // 클라이언트의 연결이 종료될때까지 계속 실행
                {
                    if (myStream.CanRead) // 스트림에서 데이터를 읽을 수 있는 경우
                    {
                        var msg = myRead.ReadLine();
                        var Smsg = msg.Split('&'); // &를 기준으로 메시지 구분
                        if (Smsg[0] == "S001") // 첫 구분자가  "S001"인 경우
                        {
                                                                    // currentPoint
                                                                    var cp = Smsg[1].Split(',');
                                                                    currentPoint_.X     =     panel1.Location.X     +
Convert.ToInt32((cp[0]));
                                                                    currentPoint_.Y     =     panel1.Location.Y     +
Convert.ToInt32((cp[1]));
                                                                    // previousPoint
                                                                    var pp = Smsg[2].Split(',');
                                                                    previousPoint.X     =     panel1.Location.X     +
Convert.ToInt32((pp[0]));
                                                                    previousPoint.Y     =     panel1.Location.Y     +
Convert.ToInt32((pp[1]));

                                                                    // Pen color
                                                                    var pc = Smsg[3].Split(',');
```

```
                                                currentPen.Color                                =
Color.FromArgb(Convert.ToInt32(pc[3]), // A

                                                        Convert.ToInt32(pc[0]),         // R
                                                        Convert.ToInt32(pc[1]),         // G
                                                        Convert.ToInt32(pc[2])); // B

                                                // Pen size
                                                var ps = Smsg[4];
                                                currentPen.Width = Convert.ToInt32((ps));
                                                // Pen style
                                                var pst = Smsg[5];
                                                setStrip = Convert.ToInt32((pst));
                                                if (setStrip == 0)
                                                        currentPen.DashStyle                    =
System.Drawing.Drawing2D.DashStyle.Solid;

                                                else if (setStrip == 1)
                                                        currentPen.DashStyle                    =

System.Drawing.Drawing2D.DashStyle.Dot;

                                                else if (setStrip == 2)
                                                        currentPen.DashStyle                    =

System.Drawing.Drawing2D.DashStyle.DashDot;

                                                // 전송받아 적용된 데이터를 이용하여 그림을 그
려줌

                                                currentGraphics_.DrawLine(currentPen,
previousPoint, currentPoint_);
                        }
                        else if (Smsg[0] == "S002")     // 새로 만들기
                        {

                                                var newFile = Smsg[1];
                                                if (newFile.Equals("New"))
                                                        panel1.Invalidate();


                        }
                        else if (Smsg[0] == "S003")
                        {
                        }
                                        else if (Smsg[0] == "S998")     // 서버 종료
                                        {
                                                MessageView(Smsg[1]);
                                                Disconnection();
                                        }
                        else
                        {
                            if (msg.Length > 0)  // 읽은 메시지가 있는 경우
                            {
                                                        string computeMessage = Smsg[0] +
" : " + Smsg[1];

                                                        // 특정 메세지는 색상 변경을 적용함
                                                        Color color = Color.Black;
```

```
                                                    if (Smsg[0].Equals("Server"))
                                                    {
                                                        color = Color.Blue;
                                                    }
                                                    else if (Smsg[0].Equals("[게임 시작]")
||

Smsg[0].Equals("[Hint]"))

                                                    {
                                                        computeMessage = Smsg[0]
+ Smsg[1];

                                                        color = Color.Red;
                                                    }
                                                    MessageView(computeMessage, color);
                            }
                        }
                    }
                }
                catch { }
            }
            private void disconnectToolStripMenuItem_Click(object sender, EventArgs e)
            {
                try
                {
                    if (this.client.Connected) // 연결중인 상태
                    {
                        var dt = Convert.ToString(DateTime.Now);
                        myWrite.WriteLine(this.myName + "&" + "채팅 APP가 종료되었습니다." + "&" + dt);
                        myWrite.Flush();
                    }
                }
                catch { }
                Disconnection();//클라이언트종료
                this.iDPORTToolStripMenuItem.Enabled = true;
                this.connectionToolStripMenuItem.Enabled = true;
            }
            private void textMessage_KeyPress(object sender, KeyPressEventArgs e)
            {
                if (e.KeyChar == (char)13) // 엔터 키를 누를때
                {
                    e.Handled = true; // 소리 없앰
                    if (this.textMessage.Text == "")
                    {
                        this.textMessage.Focus();
                    }
                    else
                    {
```

```
                Msg_send(); // Msg_send()함수 호출
            }
        }
    }
    private void Form1_FormClosing(object sender, FormClosingEventArgs e)
    {
        Disconnection(); // 클라이언트종료
    }
    private void toolStripButton1_Click(object sender, EventArgs e)
    {
    }

    private void 변경ToolStripMenuItem_Click(object sender, EventArgs e)
    {
        clg = new ColorDialog();
        clg.ShowDialog();
        setColor = 1;
        // Set the initial color of the dialog to the current text color
    }
    private void asdfasdfToolStripMenuItem_Click(object sender, EventArgs e)
    {

    }
    private void 그리기ToolStripMenuItem_Click(object sender, EventArgs e)
    {
        isRemoveMode_ = false;
    }
    private void 지우기ToolStripMenuItem_Click_1(object sender, EventArgs e)
    {
        isRemoveMode_ = true;
    }
    private void 새로만들기ToolStripMenuItem_Click(object sender, EventArgs e)
    {
        panel1.Invalidate();
    }
    private void 열기OToolStripMenuItem_Click_1(object sender, EventArgs e)
    {
        OpenFileDialog openPanel = new OpenFileDialog();
        openPanel.InitialDirectory = @"C:₩";
        openPanel.Filter = "PNG (*.png)|*.png| All files (*.*)|(*.*)";
        if (openPanel.ShowDialog() == DialogResult.OK)
        {
            Bitmap bitmap = new Bitmap(openPanel.FileName);
            Graphics g = panel1.CreateGraphics();
            g.DrawImage(bitmap, 0, 0);
        }
    }
```

```
private void 저장SToolStripMenuItem_Click(object sender, EventArgs e)
{
    Bitmap bitmap = new Bitmap(panel1.Width, panel1.Height);
    Graphics g = Graphics.FromImage(bitmap);
    Size size = panel1.Size;
    g.CopyFromScreen(
        PointToScreen(new Point(panel1.Location.X, panel1.Location.Y)),
        new Point(0, 0),
        size);
    SaveFileDialog savePanel = new SaveFileDialog();
    savePanel.InitialDirectory = @"C:₩";
    savePanel.Filter = "PNG (*.png)|*.png| All files (*.*)|(*.*)";
    if (savePanel.ShowDialog() == DialogResult.OK)
    {
        bitmap.Save(savePanel.FileName, System.Drawing.Imaging.ImageFormat.Png);
    }
}
private void toolStripMenuItem2_Click_1(object sender, EventArgs e)
{
    setTool = 1;
}
private void toolStripMenuItem3_Click_1(object sender, EventArgs e)
{
    setTool = 2;
}
private void toolStripMenuItem4_Click_1(object sender, EventArgs e)
{
    setTool = 3;
}
private void toolStripMenuItem5_Click_1(object sender, EventArgs e)
{
    setTool = 0;
}
private void dotToolStripMenuItem_Click_1(object sender, EventArgs e)
{
    setStrip = 1;
}
private void dashDotToolStripMenuItem_Click_1(object sender, EventArgs e)
{
    setStrip = 2;
}
private void soildToolStripMenuItem_Click_1(object sender, EventArgs e)
{
    setStrip = 0;
}
private void 변경ToolStripMenuItem_Click_1(object sender, EventArgs e)
{
```

```
            clg = new ColorDialog();
            clg.ShowDialog();
            setColor = 1;
            // Set the initial color of the dialog to the current text color
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            menuStrip2.Visible = false;
        }
    }
}
```