Question 8.8:

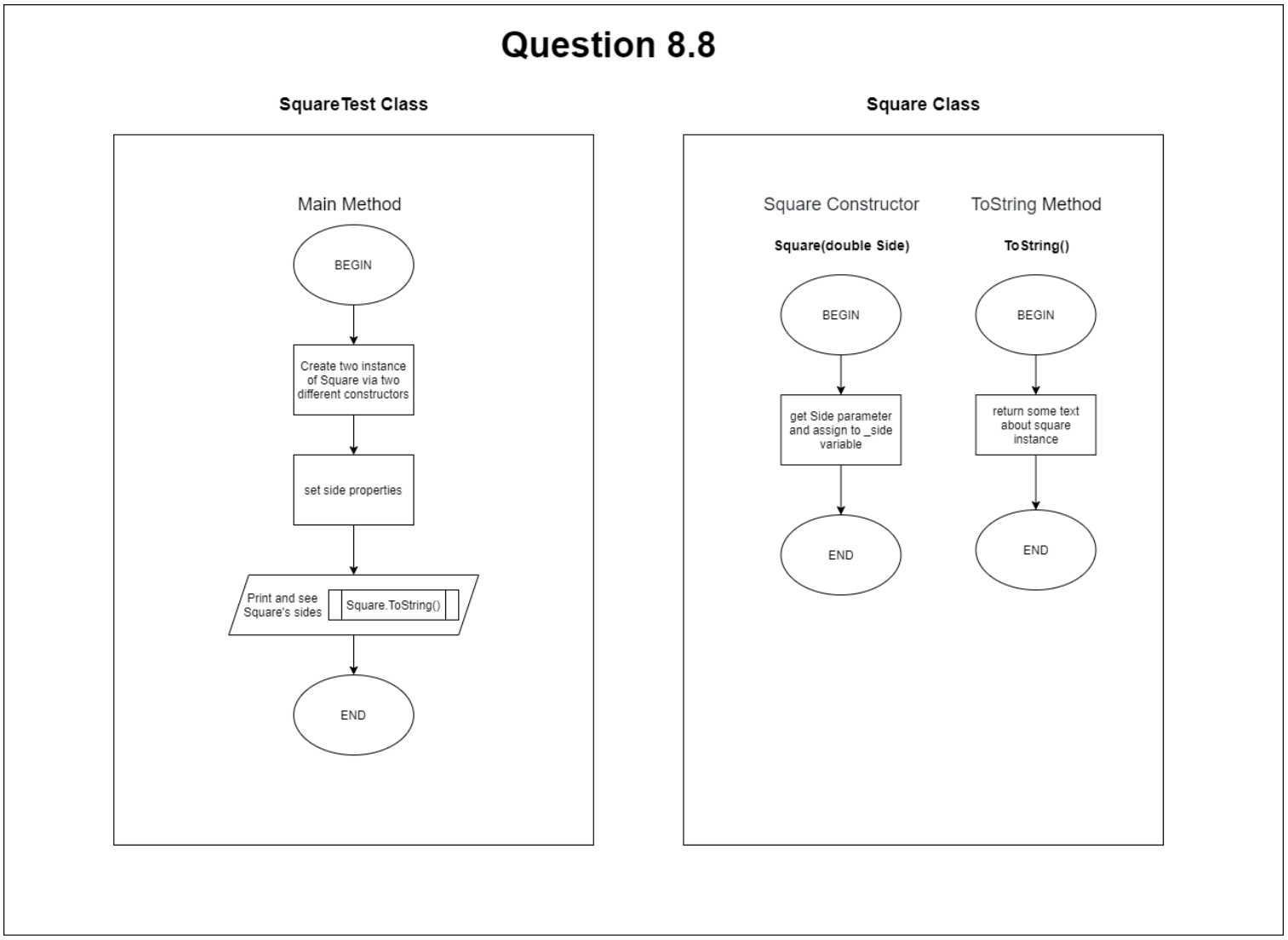
Write a console application that implements a Square shape. Class Square should contain an instance property Side that has get and set accessors for private data. Provide two constructors: one that takes no arguments and another that takes a side length as a value. Write an application class that tests class Square's functionality.

Solution 8.8:

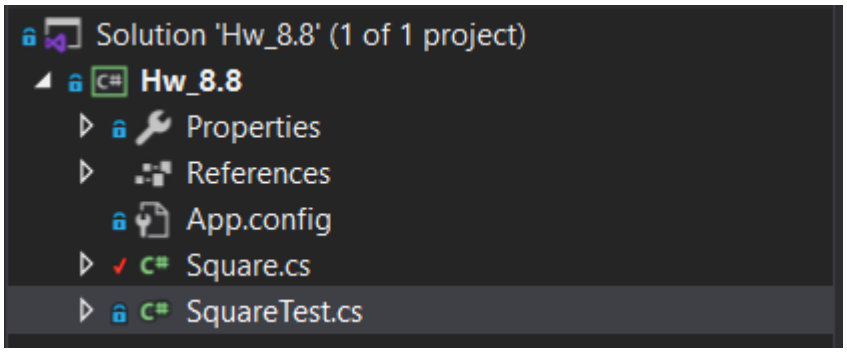
PSEUDOCODE:

- There should be a class named Square
- Square class should have side attribute which will be private and will be accessed via getters and setters
- Square class will have two constructors, one will have side parameter
- SquareTest class will test square instances.
- One instances' side property will be given via parameterized constructor, others via setter
- Both sides will be printed

FLOWCHART:



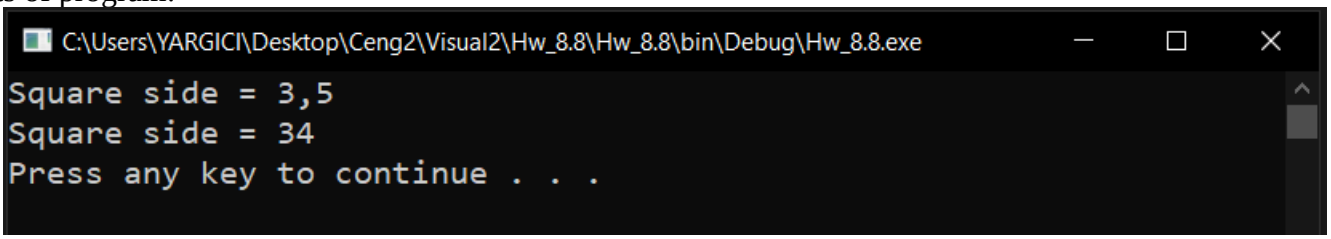
CODE:




```
namespace HW_8._8
{
    6 references
    class Square
    {
        private double _side;

        2 references
        public double Side { get => _side; set => _side = value; }

        1 reference
        public Square()
        {
        }

        1 reference
        public Square(double Side)
        {
            this.Side = Side;
        }

        2 references
        public override string ToString()
        {
            return "Square side = " + _side;
        }
    }
}
```

```
using System;

namespace HW_8._8
{

    class SquareTest
    {

        static void Main(string[] args)
        {
            // test square class without argument
            Square firstSquare = new Square();
            firstSquare.Side = 3.5;
            Console.WriteLine(firstSquare.ToString());

            // test square class side argument
            Square secondSquare = new Square(34);
            Console.WriteLine(secondSquare.ToString());

            // to stop console disappearing in debug mode
            Console.WriteLine("Press any key to continue . . .");
            Console.ReadKey();
        }
    }
}
```

Outputs of program:



Question 12.6:

Extend the program of Fig. 12.30 to include options for changing the size and color of the lines drawn. Create a GUI similar to the one in Fig. 12.35. [Hint: Have variables to keep track of the currently selected size (int) and color (Color object). Set them using the event handlers for the radio buttons. For the color, use the various Color constants (such as Color.Blue). When responding to the mouse moves, simply use the size and color variables to determine the proper size and color.]

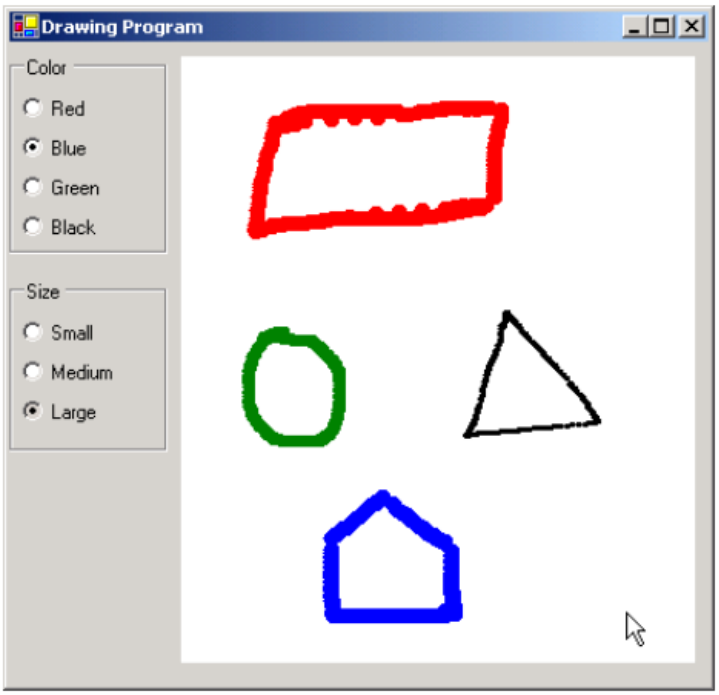


Fig. 12.35 GUI for Exercise 12.6.

Solution 12.6:

PSEUDOCODE:

- There should be two GroupBoxes which will contains colors and sizes as RadioButtons
- On the right side, there should be drawable area which we will draw our shapes
- Program should take the selected size and color RadioButtons and later it should adjust the paint brush

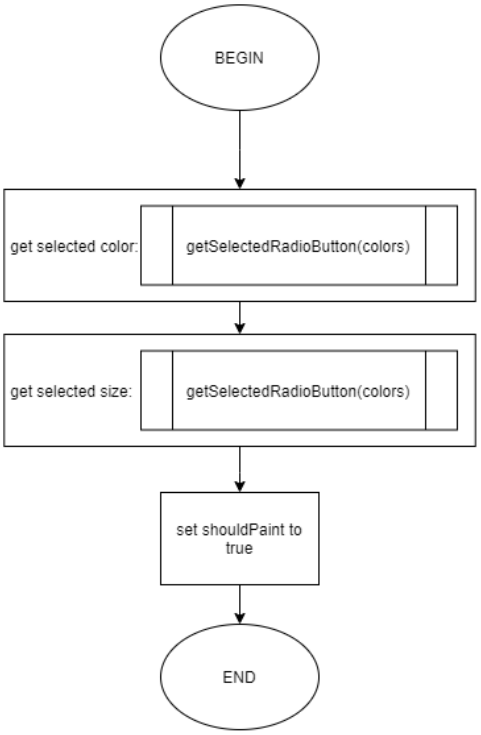
FLOWCHART:

Question 12.6

Painter Class

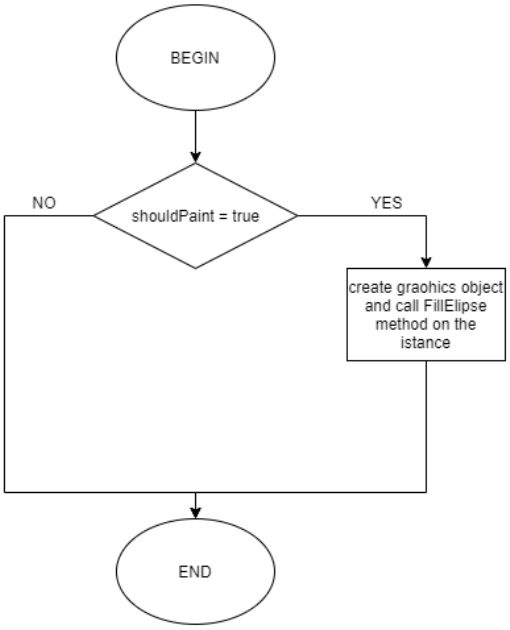
Mouse Down Method

Painter_MouseDown(object sender, MouseEventArgs e)



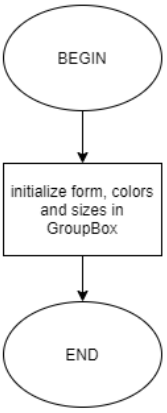
Mouse Move Method

Painter_MouseMove(object sender, MouseEventArgs e)



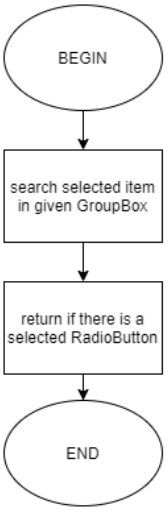
Painter Constructor

Painter()



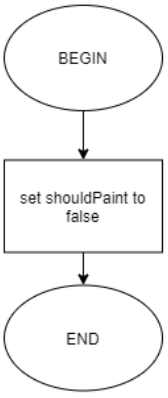
Get Selected RadioButton Method

getSelectedRadioButton(colors)

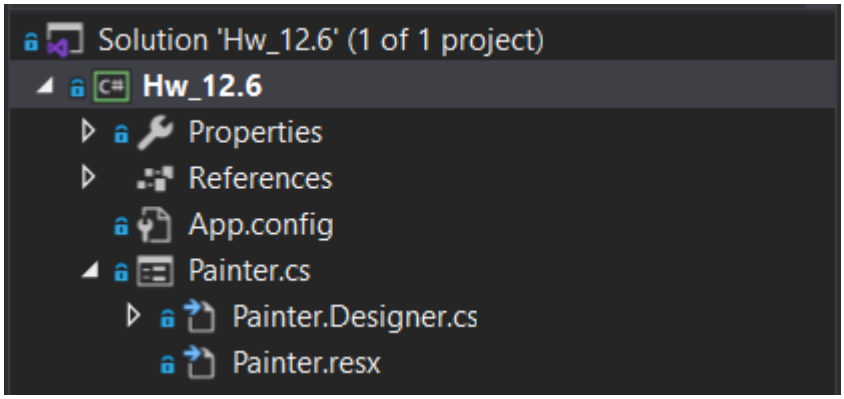


Mouse Up Method

Painter_MouseDown(object sender, MouseEventArgs e)



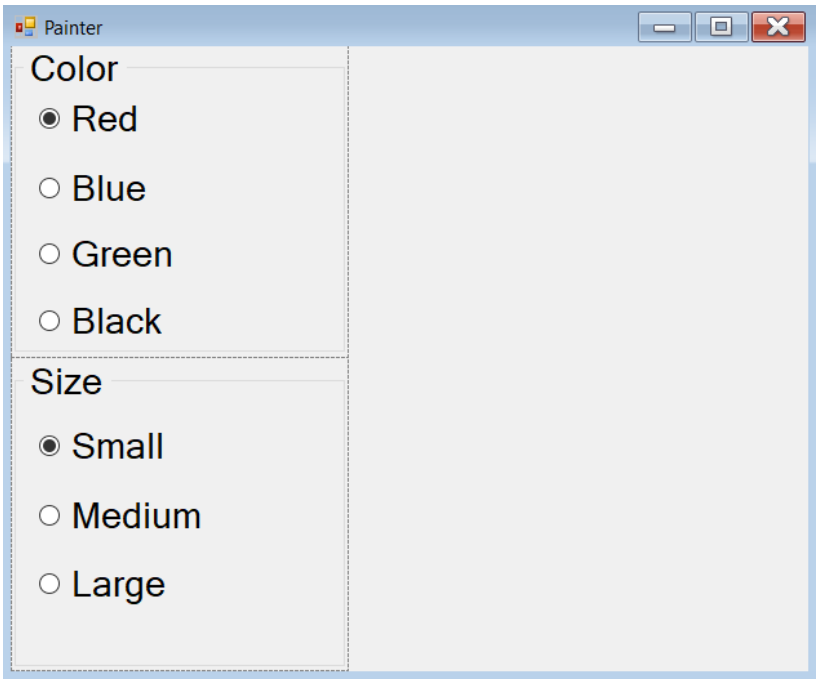
CODE:



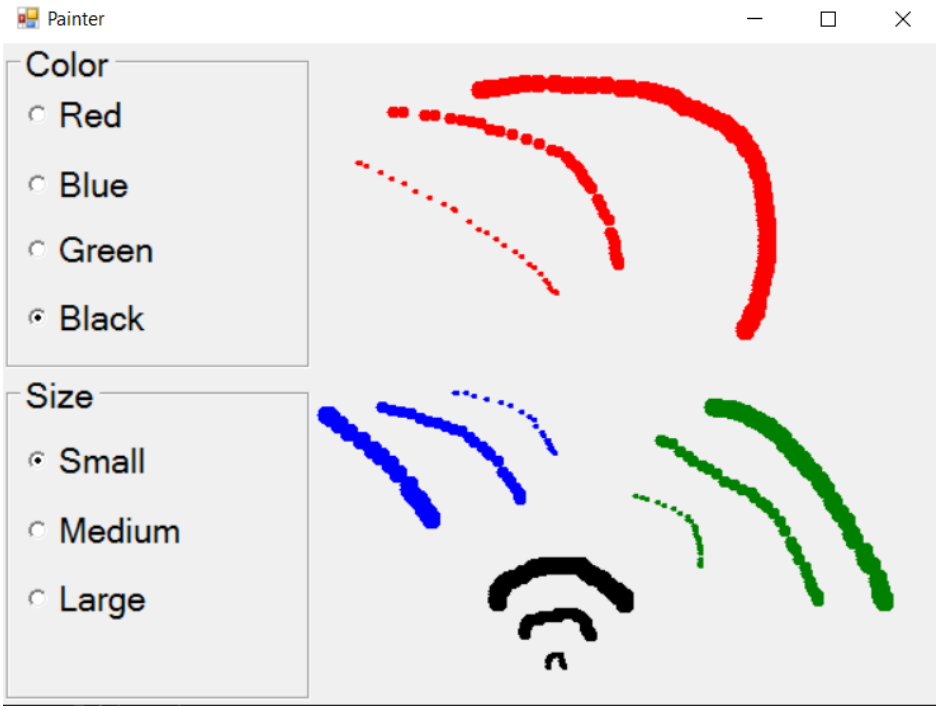

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;

namespace Hw_12._6
{
    3 references
    public partial class Painter : Form
    {
        bool shouldPaint = false; // whether to paint
        GroupBox colors;
        GroupBox sizes;
        Dictionary<String, Color> colorTypes = new Dictionary<String, Color>()
        {
            { "Red", Color.Red },
            { "Blue", Color.Blue },
            { "Green", Color.Green},
            { "Black", Color.Black}
        };
        Dictionary<String, int> sizeTypes = new Dictionary<String, int>()
        {
            { "Small", 4},
            { "Medium", 8},
            { "Large", 12}
        };

        Graphics graphics;
        Color selectedColor;
        int selectedSize;
```

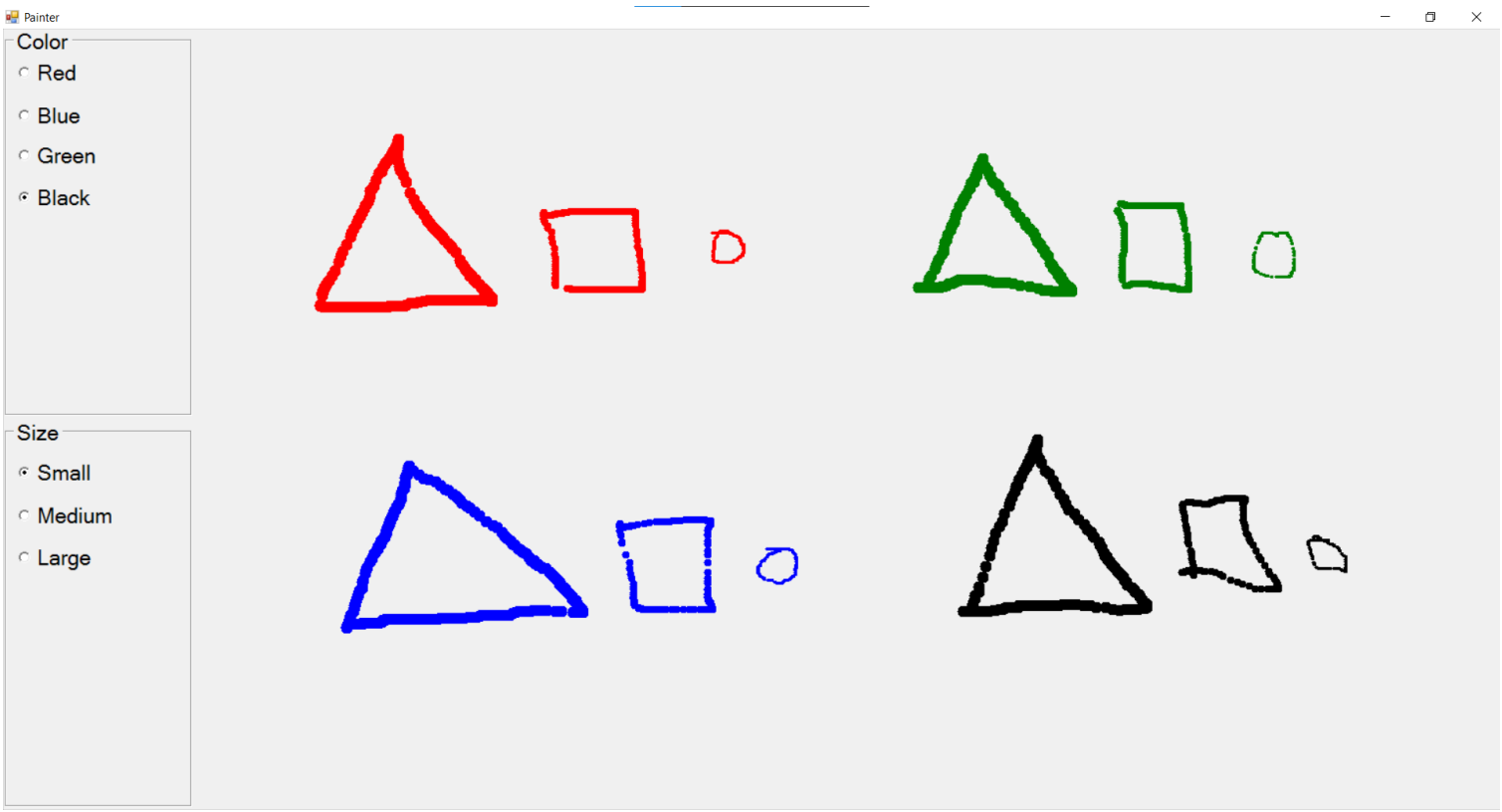
```
        /// creates a form as a drawing surface
        1 reference
        public Painter()
        {
            InitializeComponent();
            colors = gbColors;
            sizes = gbSizes;
        }

        /// The main entry point for the application.
        [STAThread]

        static void Main()
        {
            Application.Run(new Painter());
        }

        // should paint after mouse button has been pressed

        private void Painter_MouseDown(object sender, MouseEventArgs e)
        {

            RadioButton checkedColor = getSelectedRadioButton(colors);
            RadioButton checkedSize = getSelectedRadioButton(sizes);
            selectedColor = colorTypes[checkedColor.Text];
            selectedSize = sizeTypes[checkedSize.Text];
            shouldPaint = true;
        }

        2 references
        private RadioButton getSelectedRadioButton(GroupBox groupBox)
        {

            foreach (RadioButton rdo in groupBox.Controls.OfType<RadioButton>())
            {
                if (rdo.Checked)
                {
                    return rdo;
                }
            }
            throw new Exception("Radio button should be selected");
        }

        // stop painting when mouse button released
        1 reference
        private void Painter_MouseUp(object sender, MouseEventArgs e)
        {
            shouldPaint = false;
        }

        // draw circle whenever mouse button moves (and mouse is down)
        1 reference
        protected void Painter_MouseMove(object sender, MouseEventArgs e)
        {
            if (shouldPaint)
            {
                graphics = CreateGraphics();
                graphics.FillEllipse(new SolidBrush(selectedColor), e.X, e.Y, selectedSize, selectedSize);
            }

        } // end Painter_MouseMove

    } // end class Painter
}
```

Design:



Outputs of program:




Painter
Color
Red
Blue
Green
Black
Size
Small
Medium
Large

Question 13.8:

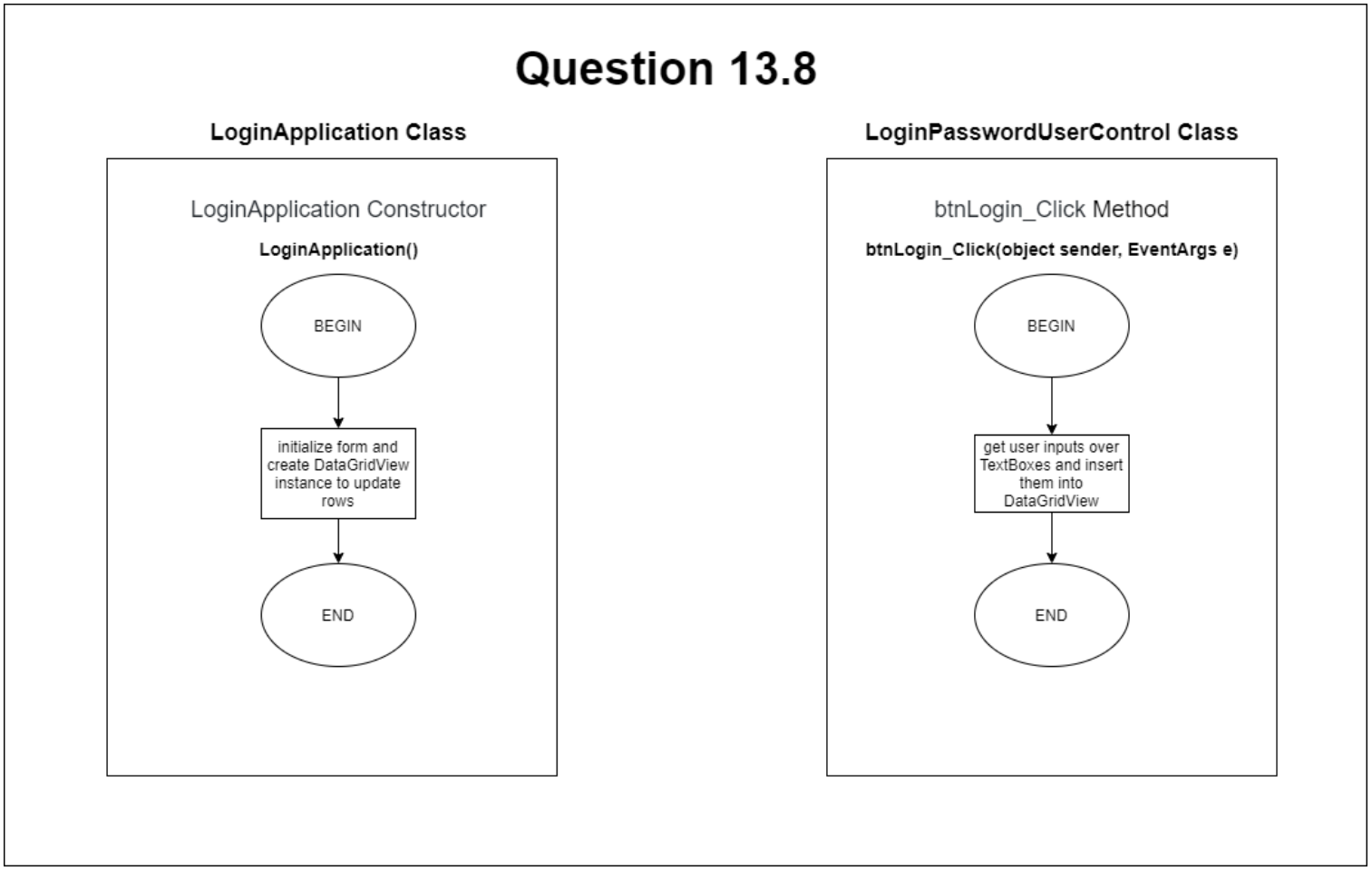
Create a UserControl called LoginPasswordUserControl. The LoginPasswordUserControl contains a Label (loginLabel) that displays String "Login:", a TextBox (loginTextBox) where the user inputs a login name, a Label (passwordLabel) that displays the String "Password:" and finally, a TextBox (passwordTextBox) where a user inputs a password (don't forget to set property PasswordChar to "*" in the TextBox's Properties window). LoginPasswordUserControl must provide public read-only properties Login and Password that allow an application to retrieve the user input from loginTextBox and passwordTextBox. The UserControl must be exported to an application that displays the values input by the user in LoginPasswordUserControl.

Solution 13.8:

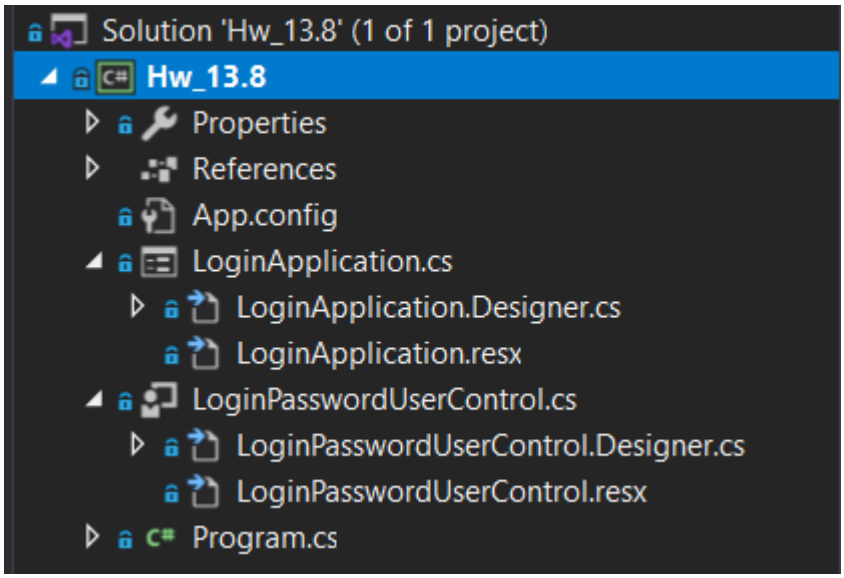
PSEUDOCODE:

- There should be a UserControl which will contain login and password Laybels and TextBoxes
- There can be a DataGridView to show login and password fields
- When user enters input, it should be hidden as '*'
- Later on, created UserControl should be implemented on a Form Application

FLOWCHART:



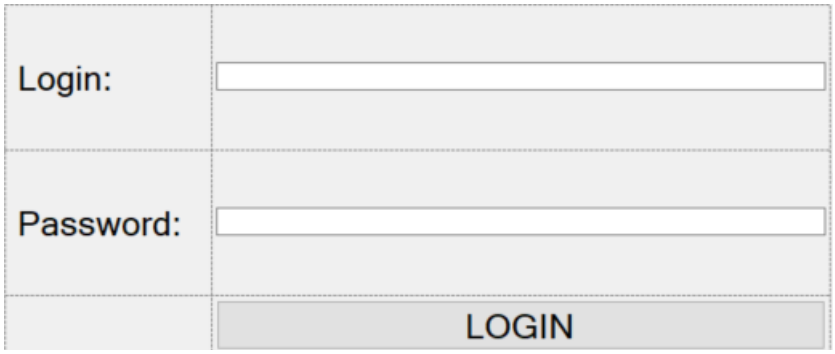
CODE:

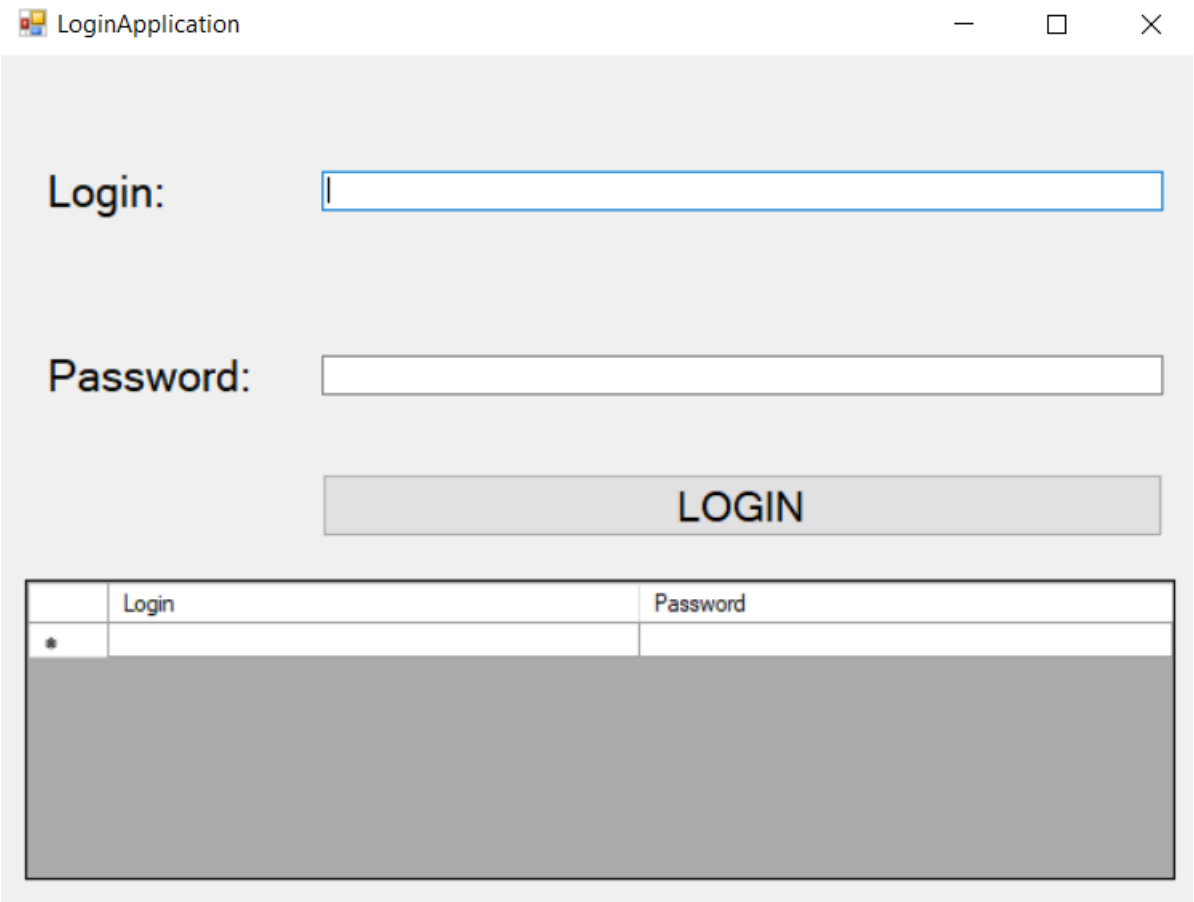


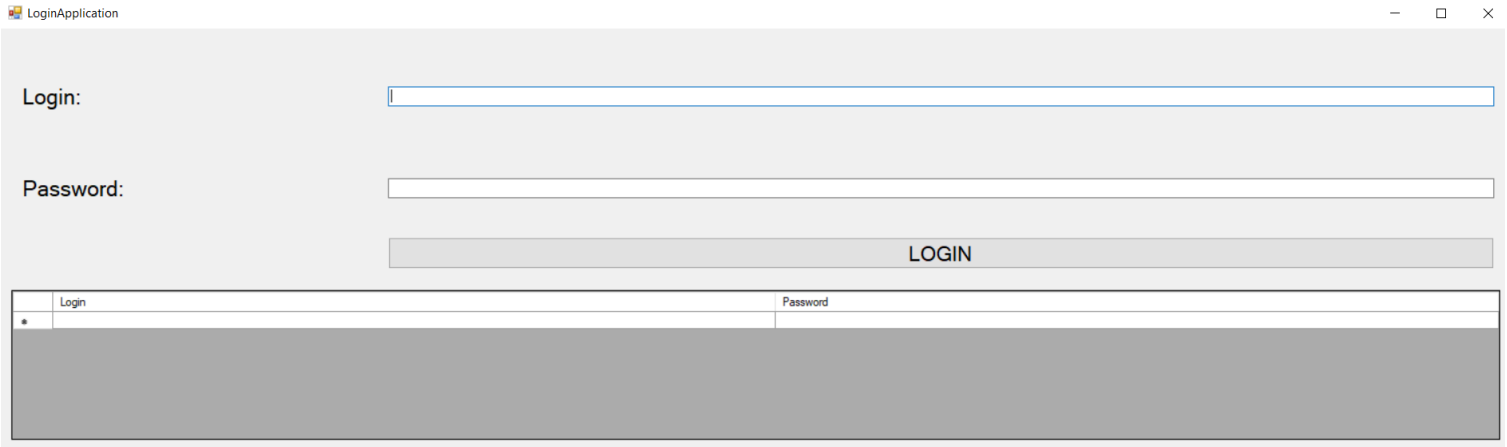

```
using System;
using System.Windows.Forms;

namespace Hw_13._8
{
    4 references
    public partial class LoginPasswordUserControl : UserControl
    {
        public String Login = "";
        public String Password = "";
        1 reference
        public LoginPasswordUserControl()
        {
            InitializeComponent();
        }

        1 reference
        private void btnLogin_Click(object sender, EventArgs e)
        {
            // taking the input of TextBoxes and assiging them in a variable
            Login = loginTextBox.Text.ToString();
            Password = passwordTextBox.Text.ToString();

            // to show user inputs in DataGridView we are adding those datas
            LoginApplication.dataGridViw.Rows.Add(Login, Password);
        }
    }
}
```

```
using System.Windows.Forms;

namespace Hw_13._8
{
    4 references
    public partial class LoginApplication : Form
    {
        // there will be only one DataGridView. Making it static will improve performance.
        public static DataGridView dataGridViw;
        1 reference
        public LoginApplication()
        {
            InitializeComponent();
            dataGridViw = dataGridView;
        }
    }
}
```

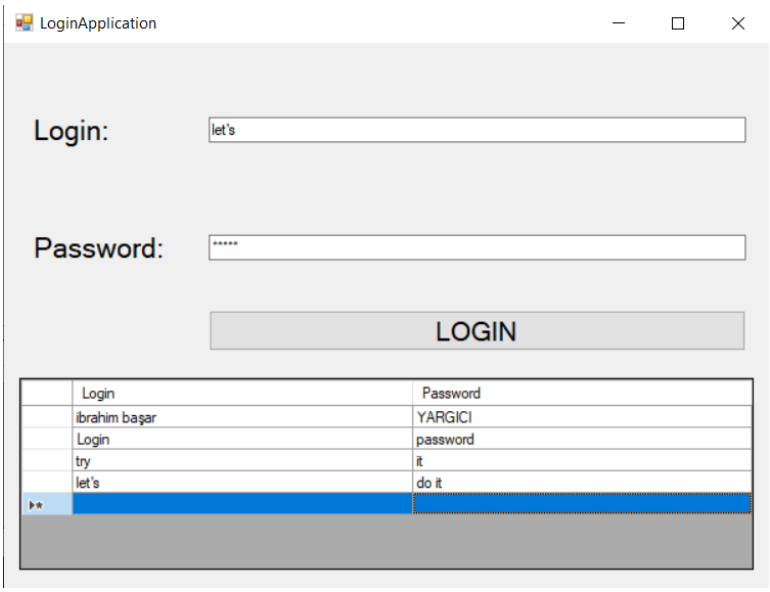
Design:

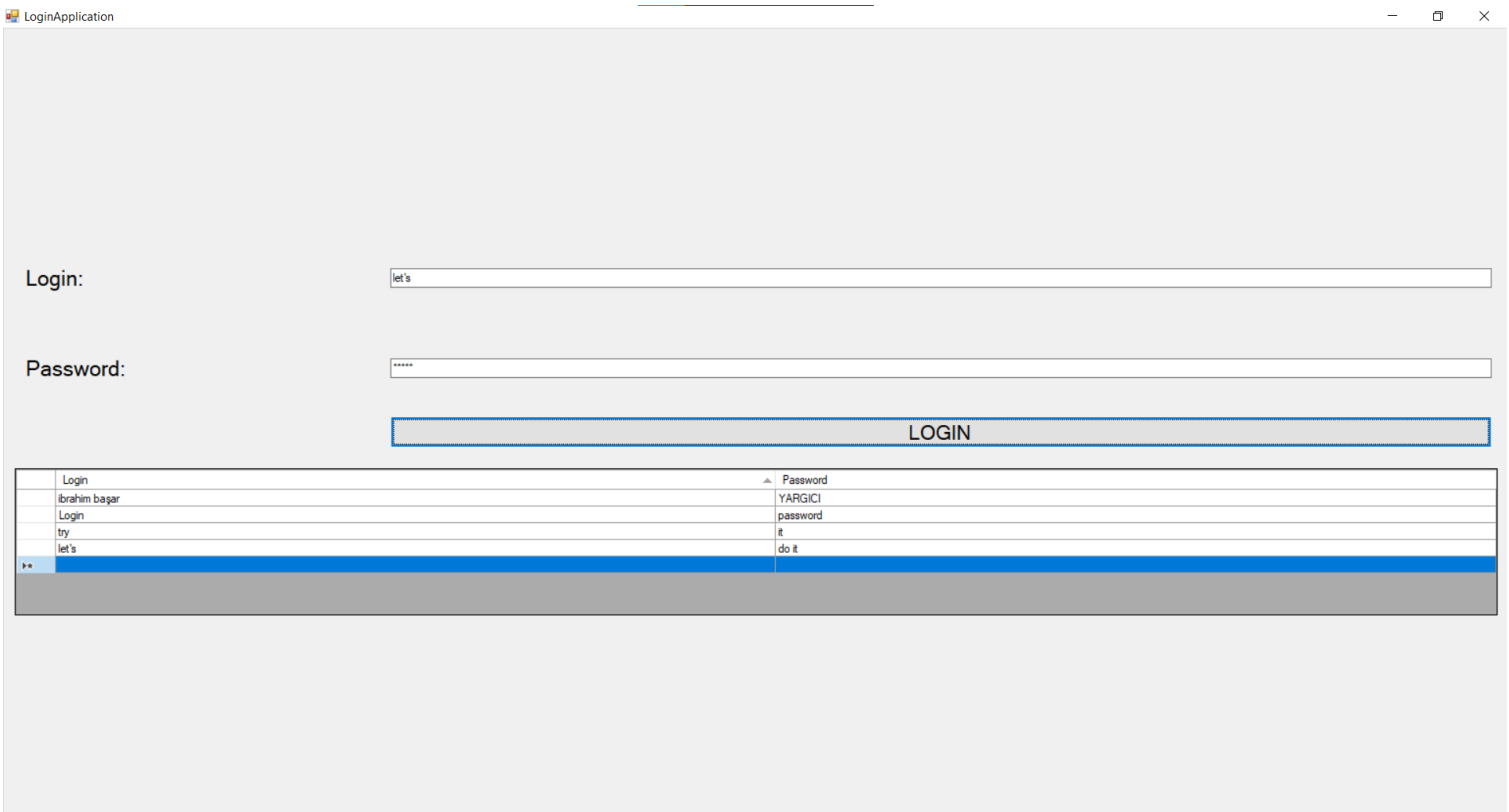

Login:
Password:
LOGIN


LoginApplication
Login:
Password:
LOGIN
Login
Password


LoginApplication
Login:
Password:
LOGIN
Login
Password

Outputs of program:





Question 17.8:

You are the owner of a hardware store and need to keep an inventory of the different tools you sell, how many of each are currently in stock and the cost of each. Write a program that initializes the random-access file hardware.dat to 100 empty records, lets you input data relating to each tool, enables you to list all your tools, lets you delete a record for a tool that you no longer have and lets you update any information in the file. The tool identification number should be the record number. Use the information in Fig. 17.24 to start your file.

Record #	Tool name	Quantity	Price
3	Electric sander	18	35.99
19	Hammer	128	10.00
26	Jig saw	16	14.25
39	Lawn mower	10	79.50
56	Power saw	8	89.99
76	Screwdriver	236	4.99
81	Sledge hammer	32	19.75
88	Wrench	65	6.48

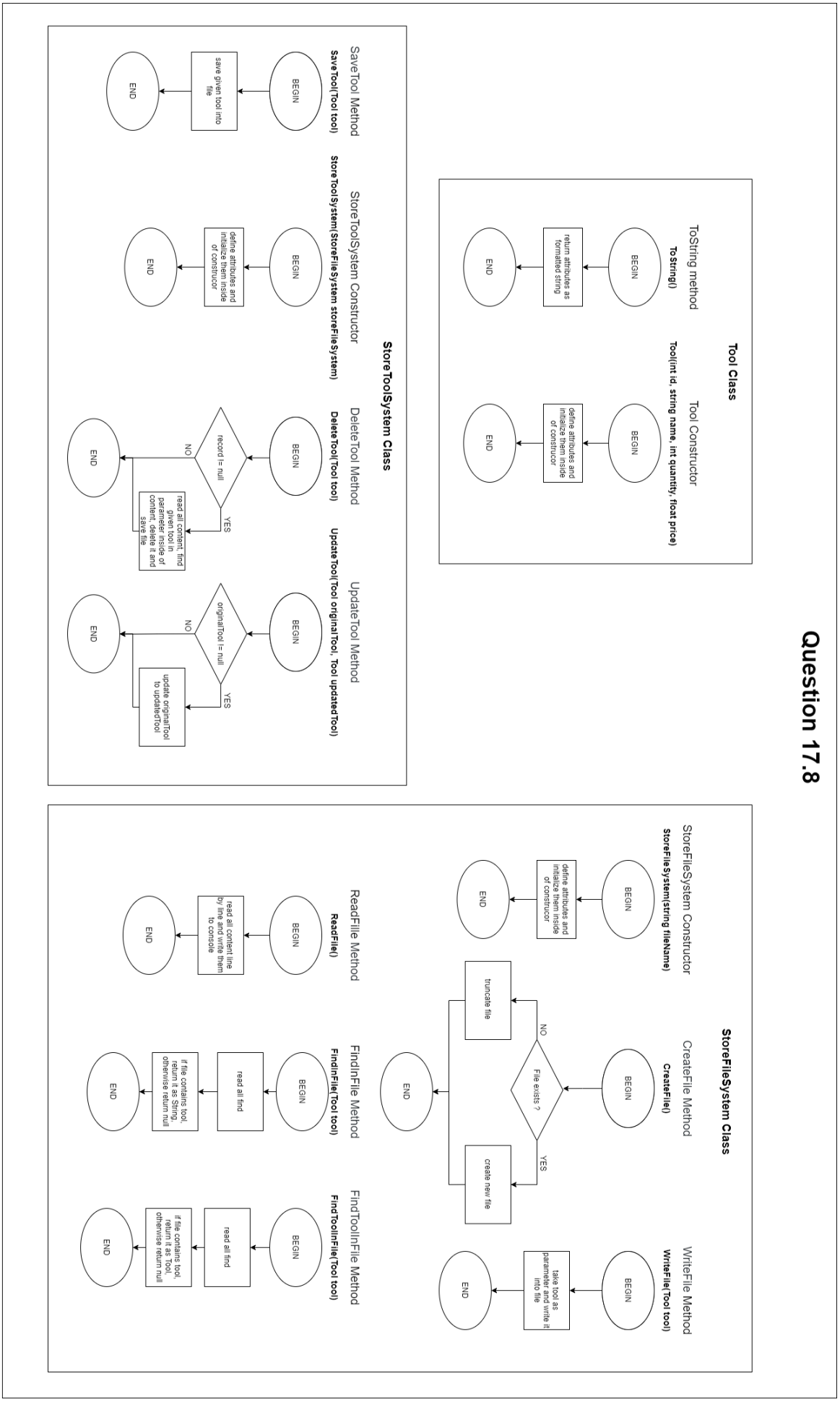
Fig. 17.24 Inventory of a hardware store.

Solution 17.8:

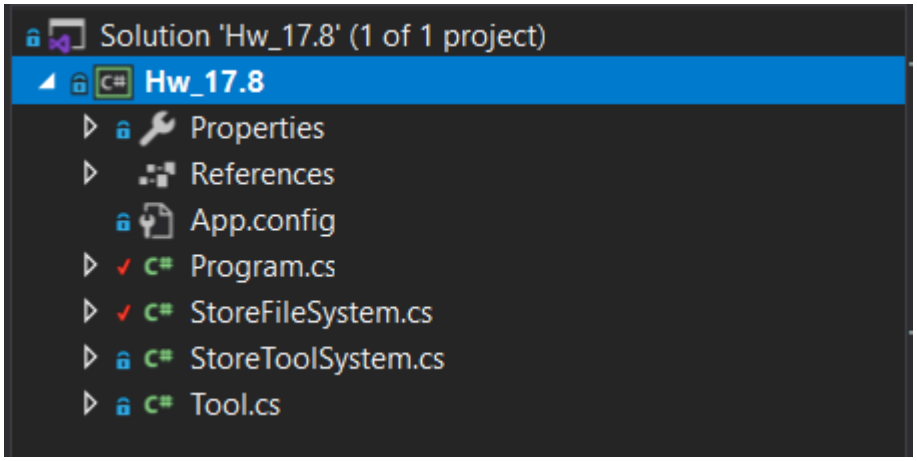
PSEUDOCODE:

- There should be a Tool entity class which represents hardware tool in our program.
- There can be two different system classes: StoreFileSystem, which is responsible of file processes, and StoreToolSystem, which is responsible of adding,updating and deleting product from database.
- Program lets user input data relating to each tool, enables user to list all your tools, lets user delete a record for a tool that user no longer have and lets user update any information in the file.
- The tool identification number should be the record number.

FLOWCHART:


Question 17.8
Tool Class
ToString method
ToString()
BEGIN
return attributes as formatted string
END
Tool Constructor
Tool(int id, string name, int quantity, float price)
BEGIN
define attributes and initialize them inside of construcor
END
StoreToolSystem Class
SaveTool Method
SaveTool(Tool tool)
BEGIN
save given tool into file
END
StoreToolSystem Constructor
StoreToolSystem(StoreFileSystem storeFileSystem)
BEGIN
define attributes and initialize them inside of construcor
END
DeleteTool Method
DeleteTool(Tool tool)
BEGIN
record != null
YES
NO
read all content, find given tool in parameter inside of content, delete it and save file
END
UpdateTool Method
UpdateTool(Tool originalTool, Tool updatedTool)
BEGIN
originalTool != null
YES
NO
update originalTool to updatedTool
END
StoreFileSystem Class
StoreFileSystem Constructor
StoreFileSystem(string fileName)
BEGIN
define attributes and initialize them inside of construcor
END
CreateFile Method
CreateFile()
BEGIN
File exists ?
NO
YES
truncate file
create new file
END
WriteFile Method
WriteFile(Tool tool)
BEGIN
take tool as parameter and write it into file
END
ReadFile Method
ReadFile()
BEGIN
read all content line by line and write them to console
END
FindInFile Method
FindInFile(Tool tool)
BEGIN
read all find
if file contains tool, return it as String, otherwise return null
END
FindToolInFile Method
FindToolInFile(Tool tool)
BEGIN
read all find
if file contains tool, return it as Tool, otherwise return null
END

CODE:




```
using System.Threading;

namespace Hw_17._8
{
    /// <summary>
    ///     This class is an entity class which describes Tool objects
    /// </summary>
    35 references
    class Tool
    {
        private int id = 0;
        private string name;
        private int quantity;
        private float price;
        public static int globalID;

        4 references
        public float Price
        {
            get { return price; }
            set { price = value; }
        }

        4 references
        public int Quantity
        {
            get { return quantity; }
            set { quantity = value; }
        }

        4 references
        public string Name
        {
            get { return name; }
            set { name = value; }
        }


        5 references
        public int RecordId
        {
            get { return id; }
            set { id = value; }
        }

        1 reference
        public Tool(int id, string name, int quantity, float price)
        {
            this.id = id;
            this.name = name;
            this.quantity = quantity;
            this.price = price;
        }

        10 references
        public Tool(string name, int quantity, float price)
        {
            id = Interlocked.Increment(ref globalID);
            this.name = name;
            this.quantity = quantity;
            this.price = price;
        }

        /// <summary>
        ///     method formatted as written in hardware.dat file
        /// </summary>
        /// <returns></returns>
        1 reference
        public override string ToString()
        {
            return string.Format("{0,-10} {1,-15} {2,-10} {3,-10}", RecordId, Name, Quantity, Price);
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;

namespace Hw_17._8
{
    3 references
    class StoreToolSystem
    {
        StoreFileSystem storeFileSystem;

        /// <summary>
        ///     Constructor for initializing StoreToolSystem
        /// </summary>
        /// <param name="storeFileSystem"> to access file processes</param>
        1 reference
        public StoreToolSystem(StoreFileSystem storeFileSystem)
        {
            this.storeFileSystem = storeFileSystem;
        }


        /// <summary>
        ///     Saves given tool into hardware.dat file via storeFileSystem instance
        /// </summary>
        /// <param name="tool"> is a Tool object that will be saved </param>
        2 references
        public void SaveTool(Tool tool)
        {
            if (tool is null)
            {
                throw new ArgumentNullException(nameof(tool));
            }

            storeFileSystem.WriteFile(tool);
        }

        /// <summary>
        ///     Saves given list of tools into hardware.dat file via storeFileSystem instance
        /// </summary>
        /// <param name="toolList"> is a list of Tool objects that will be saved </param>
        1 reference
        public void SaveTool(List<Tool> toolList)
        {
            toolList.ForEach(l => SaveTool(l));
        }

        /// <summary>
        ///     Deletes given tool from hardware.dat file via storeFileSystem instance
        /// </summary>
        /// <param name="tool"> is a Tool object that will be deleted </param>
        1 reference
        public void DeleteTool(Tool tool)
        {
            if (tool is null)
            {
                throw new ArgumentNullException(nameof(tool));
            }

            String record = storeFileSystem.FindInFile(tool);

            // create temporary file, make the changes and restore main file
            if (record != null)
            {
                var tempFile = Path.GetTempFileName();
                var linesToKeep = File.ReadLines(storeFileSystem.FileName).Where(l => l.Substring(0, 5) != record.Substring(0, 5));
                File.WriteAllLines(tempFile, linesToKeep);
                File.Delete(storeFileSystem.FileName);
                File.Move(tempFile, storeFileSystem.FileName);
            }

        }

```

```
        /// <summary>
        ///     Updates given originalTool to updatedTool via storeFileSystem instance
        /// </summary>
        /// <param name="originalTool"> original instance </param>
        /// <param name="updatedTool"> the form of updated instance </param>
        1 reference
        public void UpdateTool(Tool originalTool, Tool updatedTool)
        {
            Tool t = storeFileSystem.FindToolInFile(originalTool);

            if (t != null)
            {
                t.Name = updatedTool.Name;
                t.Quantity = updatedTool.Quantity;
                t.Price = updatedTool.Price;

                // read all content of hardware.dat and replace required places
                string str = File.ReadAllText(storeFileSystem.FileName);
                File.ReadLines(storeFileSystem.FileName).ToList().ForEach(l =>
                {
                    string subString = l.Substring(0, 5).Trim();

                    if (subString.Equals(t.RecordId.ToString()))
                    {
                        File.WriteAllText(storeFileSystem.FileName, str.Replace(l.ToString(), t.ToString()));
                    }
                });
            }

        }
    }
}
```

```
using System;
using System.IO;
using System.Linq;

namespace Hw_17._8
{
    5 references
    class StoreFileSystem
    {
        private string fileName;

        6 references
        public string FileName
        {
            get { return fileName; }
            set { fileName = value; }
        }

        FileStream fs;

        /// <summary>
        ///     Constructor for initializing StoreFileSystem
        /// </summary>
        /// <param name="fileName"> name of .dat file </param>
        1 reference
        public StoreFileSystem(string fileName)
        {
            this.fileName = fileName;
        }

        /// <summary>
        ///     If given file with a fileName name does not exists, creates file and writes column names,
        ///     otherwise truncates file and writes column names
        /// </summary>
        1 reference
        public void CreateFile()
        {
            if (fileName is null)
            {
                throw new ArgumentNullException(nameof(fileName));
            }
            string columnNames = string.Format("{0,-10} {1,-15} {2,-10} {3,-10}", "Record #", "Tool name", "Quantity", "Price");


            if (File.Exists(fileName))
            {
                fs = new FileStream(fileName, FileMode.Truncate, FileAccess.Write);
                Console.WriteLine(columnNames);
                fs.Close();
                return;
            }

            try
            {
                fs = new FileStream(fileName, FileMode.CreateNew, FileAccess.Write);
                Console.WriteLine(columnNames);
                fs.Close();
            }
            catch (Exception e)
            {
                throw new Exception(e.Message);
            }
        }

```

```
        /// <summary>
        ///     Writes given Tool into hardware.dat file
        /// </summary>
        /// <param name="tool"> is a Tool object that will be saved </param>
        1 reference
        public void WriteFile(Tool tool)
        {
            if (tool is null)
            {
                throw new ArgumentNullException(nameof(tool));
            }
            // formatted as written in hardware.dat file
            string text = string.Format("{0,-10} {1,-15} {2,-10} {3,-10}", tool.RecordId, tool.Name, tool.Quantity, tool.Price);

            using (FileStream fs = new FileStream(fileName, FileMode.Append, FileAccess.Write))
            {
                using (StreamWriter write = new StreamWriter(fs))
                {
                    write.WriteLine(text);
                }
                fs.Close();
            }
        }

        /// <summary>
        ///     Reaads all file and prints to the console
        /// </summary>
        1 reference
        public void ReadFile()
        {
            File.ReadLines(fileName).ToList().ForEach(l => Console.WriteLine(l));
        }

        /// <summary>
        ///     Finds given tool in hardware.dat file and returns its line as string
        /// </summary>
        /// <param name="tool"> is a Tool object that will be found </param>
        /// <returns> the line that tool is written </returns>
        1 reference
        public String FindInFile(Tool tool)
        {
            try
            {
                return File.ReadLines(fileName).First(l => Int32.Parse(l.Substring(0, 5)).Equals(tool.RecordId));
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            return null;
        }

        /// <summary>
        ///     Finds given tool in hardware.dat file and returns it as Tool object
        /// </summary>
        /// <param name="tool"> is a Tool object that will be found </param>
        /// <returns> the tool in file </returns>
        1 reference
        public Tool FindToolInFile(Tool tool)
        {
            try
            {
                String foundTool = File.ReadLines(fileName).First(l => Int32.Parse(l.Substring(0, 5)).Equals(tool.RecordId));
                if (foundTool != null)
                {
                    int toolId = Int32.Parse(foundTool.Substring(0, 10));
                    string toolName = foundTool.Substring(11, 15);
                    int toolQuantity = Int32.Parse(foundTool.Substring(27, 10));
                    float toolPrice = float.Parse(foundTool.Substring(38, 10));
                    return new Tool(toolId, toolName, toolQuantity, toolPrice);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            return null;
        }
    }
}
```

```
using System;
using System.Collections.Generic;

namespace Hw_17._8
{
    0 references
    class Program
    {
        private const string FILE_NAME = "hardware.dat";

        0 references
        static void Main(string[] args)
        {
            // creating StoreFileSystem to manage file processes, StoreToolSystem for editing the file
            StoreFileSystem storeFileSystem = new StoreFileSystem(FILE_NAME);
            StoreToolSystem storeToolSystem = new StoreToolSystem(storeFileSystem);

            // creating Tool instances
            Tool electricSander = new Tool("Electric sander", 18, 35.99f);
            Tool hammer = new Tool("Hammer", 128, 10.00f);
            Tool jigSaw = new Tool("Jig saw", 16, 14.25f);
            Tool lawnMower = new Tool("Lawn mower", 10, 79.50f);
            Tool powerSaw = new Tool("Power saw", 8, 89.99f);
            Tool screwdriver = new Tool("Screwdriver", 236, 4.99f);
            Tool sledgehammer = new Tool("Sledge hammer", 32, 19.75f);
            Tool wrench = new Tool("Wrench", 65, 6.48f);
            Tool newTool = new Tool("New Tool", 9999, 99.99f);
            Tool updatedNewTool = new Tool("Updated Tool", 12345, 1.11f);

            List<Tool> toolList = new List<Tool>()
            {
                electricSander,
                hammer,
                jigSaw,
                lawnMower,
                powerSaw,
                screwdriver,
                sledgehammer,
                wrench
            };

            storeFileSystem.CreateFile();

            // saving toolList into hardware.dat file
            storeToolSystem.SaveTool(toolList);

            // newTool will be saved to hardware.dat file but before toolList list
            toolList.Add(newTool);
            storeToolSystem.SaveTool(newTool);

            // deleting an element of toolList
            storeToolSystem.DeleteTool(electricSander);

            // finding an element of toolList
            // Console.WriteLine(storeFileSystem.FindInFile(wrench));

            // updating tool which is added by SaveTool() method
            storeToolSystem.UpdateTool(newTool, updatedNewTool);

            // read content of hardware.dat file
            storeFileSystem.ReadFile();

            // to hold console
            Console.ReadKey();
        }
    }
}
```

Outputs of program:

- List all:

```
C:\Users\YARGICI\Desktop\Ceng2\Visual2\Hw_17.8\Hw_17.8\bin\Debug\Hw_1...
Record #    Tool name        Quantity    Price
1           Electric sander  18          35,99
2           Hammer           128         10
3           Jig saw          16          14,25
4           Lawn mower       10          79,5
5           Power saw        8           89,99
6           Screwdriver      236         4,99
7           Sledge hammer    32          19,75
8           Wrench           65          6,48
```

- Save new tool (record #= 9)::

```
C:\Users\YARGICI\Desktop\Ceng2\Visual2\Hw_17.8\Hw_17.8\bin\Debug\Hw_17.8.exe
Record #    Tool name        Quantity    Price
1           Electric sander  18          35,99
2           Hammer           128         10
3           Jig saw          16          14,25
4           Lawn mower       10          79,5
5           Power saw        8           89,99
6           Screwdriver      236         4,99
7           Sledge hammer    32          19,75
8           Wrench           65          6,48
9           New Tool         9999        99,99
```

- Delete record (record #= 1):

```
C:\Users\YARGICI\Desktop\Ceng2\Visual2\Hw_17.8\Hw_17.8\bin\Debug\Hw_17.8.exe
Record #    Tool name        Quantity    Price
2           Hammer           128         10
3           Jig saw          16          14,25
4           Lawn mower       10          79,5
5           Power saw        8           89,99
6           Screwdriver      236         4,99
7           Sledge hammer    32          19,75
8           Wrench           65          6,48
9           New Tool         9999        99,99
```

- Update record (record #= 9):

```
C:\Users\YARGICI\Desktop\Ceng2\Visual2\Hw_17.8\Hw_17.8\bin\Debug\Hw_17.8...
Record #    Tool name       Quantity    Price
2           Hammer          128         10
3           Jig saw         16          14,25
4           Lawn mower      10          79,5
5           Power saw       8           89,99
6           Screwdriver     236         4,99
7           Sledge hammer   32          19,75
8           Wrench          65          6,48
9           Updated Tool    12345       1,11
```

- Content of hardware.dat file:

```
C:\Users\YARGICI\Desktop\Ceng2\Visual2\Hw_17.8\Hw_17.8\bin\Debug\hardware.dat - Notepad++
File  Edit  Search  View  Encoding  Language  Settings  Macro  Run  Plugins  Window  ?
hardware.dat
2           Hammer          128         10
3           Jig saw         16          14,25
4           Lawn mower      10          79,5
5           Power saw       8           89,99
6           Screwdriver     236         4,99
7           Sledge hammer   32          19,75
8           Wrench          65          6,48
9           Updated Tool    12345       1,11
```

- Find record:

```
C:\Users\YARGICI\Desktop\Ceng2\Visual2\Hw_17.8\Hw_17.8\bi...
Record #    Tool name       Quantity    Price
8           Wrench          65          6,48
```