SWE 316 – HW2

Submission date : 9/12/2023

Hussain Asim Al Sayed Ali

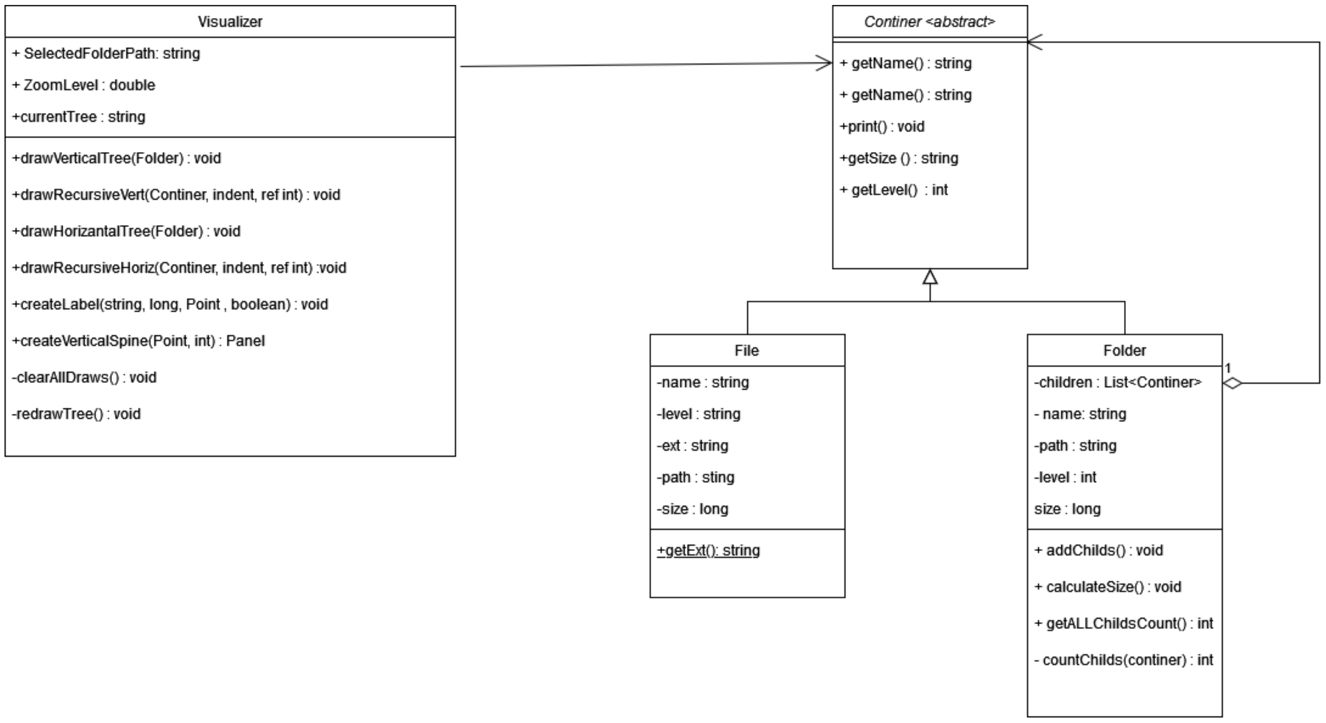
202038340

Task	Grade	Your Grade	Comments
Task # 1: Class Diagram	10		
Task # 2: Implementation	50		
Task #3 : Class diagram	10		
Check list and penalties No Cover page with grade table -10 ☐ File name (report) -5 ☐ Not in PDF format -10 ☐			
Total	70		

1. Class diagram [10 marks]

Design a class diagram showing the above-mentioned structure using the composite design pattern. You have to

show all components including the Application class.



2. Application [50 marks]

Implement a .Net desktop application (C# or VB) by which you can choose a certain folder when the program starts.

Once you select a folder, you should recursively traverse all of its contents (files and folders) and fill the required

information as follows:

• Folder : only name

• File : name, size, extension

After traversing, your application should traverse the created structure (your structure) again and calculate the size

of all folders by single line call (x.CalculateSize()) where x represent the top most folder.

After calculating the sizes of all folders and subfolders, you should visualize the folder and its contents as shown in

the sample below. You should show the file or folder size besides its name. This should be accomplished using a

single line (x.visualize()) where x represent the top most folder. You should support visualizing the folder either

vertically or horizontally as shown in the samples below.

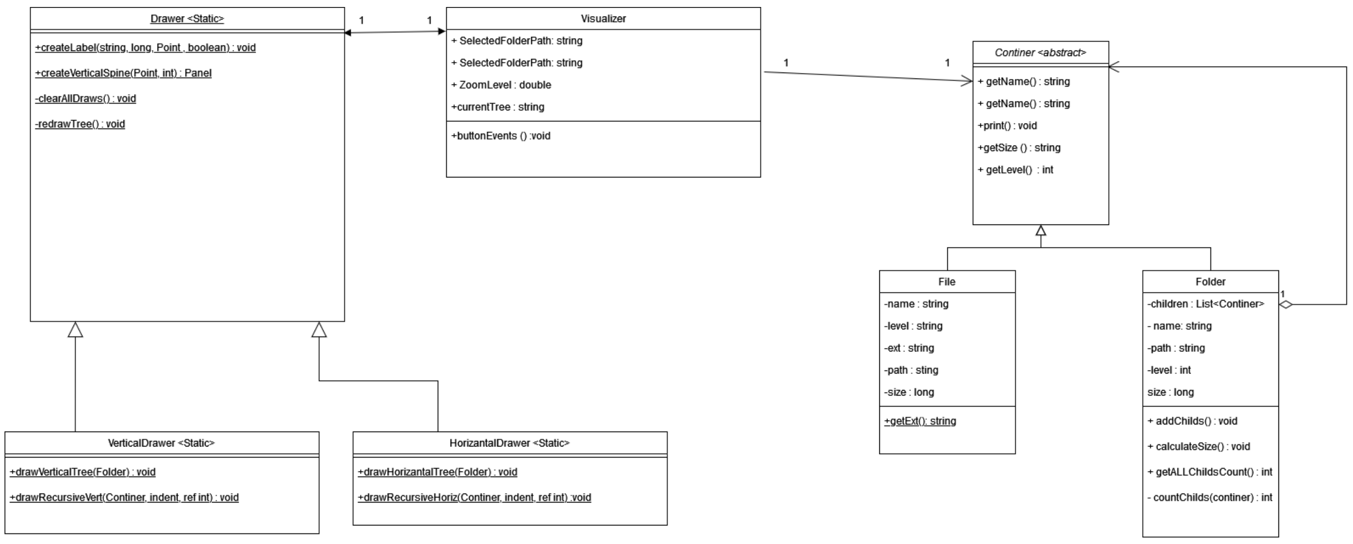
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Linq.Expressions;
using System.Text;
using System.Threading.Tasks;

namespace fileVisualizer.classes
{
    public class File :  Container
    {
        private int level;
        private string path;
        private string name;
        private string ext;
        private long size;
        public File(string name, string ext, long size, string path, int
level)
        {
            this.name = name;
            this.level = level;
            this.path = path;
            this.ext = ext;
            this.size = size;

        }

        public string getExt()
        {
            return ext;
        }
        public override void print() {
            string indentChar = "-";
            string indent = string.Join("", Enumerable.Repeat(indentChar,
level));

            System.Console.WriteLine(indent +this.name + size);

        }
        public override string getName() {
            return name;
        }
        public override long getSize()
        {
            return size;
        }
        public override int getLevel()
        {
            return level;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Diagnostics.CodeAnalysis;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace fileVisualizer.classes
{

    public class Folder : Container
    {
        private string name;
        private string path;
        private int level;
        private long size;
        private List<Container> children = new List<Container>();
        public Folder(string name, string path , int level)
        {
            this.name = name;
            this.path = path;
            this.level = level;
            addChilds();
            calculateSize();
        }

        public void addChilds() {
            DirectoryInfo selectedFolderInfo = new DirectoryInfo(path);
            string[] files = Directory.GetFiles(path);
            string[] subfolders = Directory.GetDirectories(path);

            for (int i = 0; i < files.Length; i++) {
                string currentPath = files[i];
                FileInfo currentFile = new FileInfo(currentPath);
                string currentName = currentFile.Name;
                long currentSize = currentFile.Length;
                string currentExt = currentFile.Extension;

                File file = new File(currentName, currentExt, currentSize,
currentPath, level + 1);

                children.Add(file);
            }

            for(int i = 0; i < subfolders.Length;i++) {
                DirectoryInfo currentFolder = new
DirectoryInfo(subfolders[i]);

                string folderName = currentFolder.Name;
                Container folder = new Folder(folderName, subfolders[i],
level +1);

                children.Add(folder);
            }
        }

        public override void print() {
            string indentChar = "-";
            string indent = string.Join("", Enumerable.Repeat(indentChar,
level));
            System.Console.WriteLine(indent + name + ":" + size);
            for (int i = 0; i < children.Count; i++) {
               Container child = children[i];
               child.print();
            }
        }
        public void setSize(long size) {
            this.size = size;
        }
        public void calculateSize() {
            DirectoryInfo selectedFolderInfo = new DirectoryInfo(path);
            long totalSize = selectedFolderInfo.EnumerateFiles().Sum(f =>
f.Length);
            size = totalSize;

            for (int i = 0; i < children.Count; i++) {
                if (children[i] is Folder) {
                    ((Folder)children[i]).calculateSize();

                }
            }
        }
        public List<Container> getChildren() {
            return children;
        }
        public override string getName()
        {
            return name;
        }

        public override long getSize()
        {
            return size;
        }
        public override int getLevel()
        {
            return level;
        }

        public int getALLChildsCount() {
            int sum = 0;

            for(int i = 0; i< children.Count; i++)
            {
                sum += countChilds(children[i]);
            }


            return sum;
        }
        private int countChilds(Container cont) {
            if (cont is File)
            {
                return 1;
            }
            else if (cont is Folder)
            {
                Folder currentFold = (Folder)cont;
                List<Container> childs = currentFold.getChildren();
                int sum = 1;
                for (int i = 0; i < childs.Count; i++)
                {
                    sum +=  countChilds(childs[i]);
                }
                return sum;
            }
            else return 0;

        }


    }

}



using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace fileVisualizer.classes
{
    public abstract class Container
    {

        public Container()
        {

        }



        public abstract void print();
        public abstract string getName();
        public abstract long getSize();
        public abstract int getLevel();
    }
}
using fileVisualizer.classes;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Reflection.Emit;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Xml.Linq;
using Container = fileVisualizer.classes.Container;
using Label = System.Windows.Forms.Label;

namespace fileVisualizer
{
    public partial class Form1 : Form
    {
        public string SelectedFolderPath { get; set; } =
"C:\\Users\\Hussain\\Desktop\\HW2 SWE 316\\test folder";
        public double ZoomLevel { get; set; } = 1;
        public string currentTree { get; set; } = "V";
        public Form1()
        {
            InitializeComponent();
            this.AutoScroll = true;
        }


        public void drawVerticalTree(Folder folder)
        {
            int xPos = 0;
            drawRecursiveVert((Container)folder, 0, ref xPos);
            currentTree = "V";

        }
        public void drawRecursiveVert(Container continer, int indent , ref
int xPos) {
            Boolean isFolder;
            if (continer is Folder)
                isFolder = true;
            else
                isFolder = false;



            Point currentPoint = new Point(50 + xPos , 50 + 150 *
continer.getLevel());
            Label label = CreateLabel(continer.getName(),
continer.getSize(), currentPoint, isFolder);
            this.Controls.Add(label);
            this.Invalidate();
            xPos += (label.Right - label.Left) + 40;
            Console.WriteLine(label.Right - label.Left);

            if (continer.getLevel() != 0)
            {

                Panel verticalSpine = CreateVerticalSpine(new
Point((label.Right + label.Left)/2, label.Top -125 ) ,  125);
                this.Controls.Add(verticalSpine);
            }
            if (isFolder)
            {
                Folder currFold = (Folder)continer;
                List<Container> children =
((Folder)continer).getChildren();
                if (children.Count != 0)
                {
                    Panel horizantalLine = CreateHorizantalLine(new
Point(label.Right, (label.Top + label.Bottom) / 2), 240 *
currFold.getALLChildsCount() + 20);
                    this.Controls.Add(horizantalLine);
                }


                for (int i = 0; i < children.Count; i++)
                {
                    drawRecursiveVert(children[i], indent + 70, ref xPos);
                    //drawSpine(new Point(), children[i].getLevel(),
children.Count);

                }
            }

        }



        public void drawHorizantalTree(Folder folder) {
            int yPos = 0;
            drawRecursiveHoriz((Container)folder, 0 , ref yPos );
            currentTree = "H";
        }

        public void drawRecursiveHoriz(Container continer , int indent ,
ref int yPos) {

            Boolean isFolder;
            if (continer is Folder)
                 isFolder = true;
            else
                isFolder = false;

            Point currentPoint = new Point(50 + 150 * continer.getLevel(),
50 + yPos);
            Label label = CreateLabel(continer.getName(),
continer.getSize(), currentPoint, isFolder);
            //TextRenderer.DrawText(e.Graphics, label.Text, label.Font,
label.Location, label.ForeColor);

            this.Controls.Add(label);
            this.Invalidate();
            yPos += label.Height + 40;


            if (continer.getLevel() != 0) {
                Panel horizantalLine = CreateHorizantalLine(new
Point(label.Left, (label.Top + label.Bottom) / 2), 200);
                this.Controls.Add(horizantalLine);
            }

            if (isFolder) {
                Folder currFold = (Folder)continer;
                List<Container> children =
((Folder)continer).getChildren();
                if (children.Count != 0) {
                    Panel verticalSpine = CreateVerticalSpine(new
Point((label.Right + label.Left) / 2, label.Bottom ), 85 *
currFold.getALLChildsCount());
                    this.Controls.Add(verticalSpine);
                }


                for (int i = 0; i < children.Count; i++)
                {
                    drawRecursiveHoriz(children[i], indent + 70 ,ref yPos);
                    //drawSpine(new Point(), children[i].getLevel(),
children.Count);

                }
            }



        }

        public Label CreateLabel(string text, long size,  Point location,
Boolean isFolder)
        {
            // Create label
            Label label = new Label();

            // Set properties
            label.Text = text+ "\n" + size ;
            Console.WriteLine("text is "+text);

            label.Font = new Font("Arial", 12);

            label.Location = location;
            label.Width =(int)( 200 * ZoomLevel);
            label.Height = (int)(50 * ZoomLevel);
            // Customize further
            if (isFolder)
                label.BackColor = Color.LightBlue;
            else {
                label.BackColor = Color.AntiqueWhite;
            }

            // Add event handler


            return label;
        }
        public Panel CreateVerticalSpine(Point location, int height)
        {
            Panel spine = new Panel
            {
                BackColor = Color.Black,
                Location = location,
                Size = new Size(2, (int)(height * ZoomLevel))
            };

            return spine;
        }
        public Panel CreateHorizantalLine(Point location , int width) {
            location.Offset(-50, 0);
            Panel spine = new Panel
            {

                BackColor = Color.Black,
                Location = location,
                Size = new Size((int) (ZoomLevel * width), 2)
            };

            return spine;

        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.KeyPreview = true;  // Enable key events for the form
            this.KeyDown += Form1_KeyDown;  // Handle the KeyDown event
        }

        private void Form1_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.OemMinus)  // Zoom out when '-' key is
pressed
            {
                ZoomLevel -= 0.1;
                RedrawTree();
            }
            else if (e.KeyCode == Keys.Oemplus)  // Zoom in when '=' key is
pressed
            {
                ZoomLevel += 0.1;
                RedrawTree();
            }
        }
        private void RedrawTree()
        {
            clearAllDraws();
            this.Invalidate();

            DirectoryInfo selectedFolderInfo = new
DirectoryInfo(SelectedFolderPath);

            Folder mainFolder = new Folder(selectedFolderInfo.Name,
SelectedFolderPath, 0);

            if (currentTree.Equals("H"))
            {
                drawHorizantalTree(mainFolder);
            }
            else {
                drawVerticalTree(mainFolder);

            }
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Console.WriteLine("draw vertica");
            clearAllDraws();
            this.Invalidate();

            DirectoryInfo selectedFolderInfo = new
DirectoryInfo(SelectedFolderPath);

            Folder mainFolder = new Folder(selectedFolderInfo.Name,
SelectedFolderPath, 0);
            drawVerticalTree(mainFolder);
        }
        private void button2_Click(object sender, EventArgs e)
        {
            Console.WriteLine("draw horizanta");
            clearAllDraws();
            this.Invalidate();

            DirectoryInfo selectedFolderInfo = new
DirectoryInfo(SelectedFolderPath);

            Folder mainFolder = new Folder(selectedFolderInfo.Name,
SelectedFolderPath, 0);
            drawHorizantalTree(mainFolder);
        }


        private void button3_Click(object sender, EventArgs e)
        {
            FolderBrowserDialog folderDialog = new FolderBrowserDialog();

            DialogResult result = folderDialog.ShowDialog();

            if (result == DialogResult.OK)
            {
                string folderPath = folderDialog.SelectedPath;
                SelectedFolderPath = folderDialog.SelectedPath;
                Console.WriteLine("Folder selected: " + folderPath);

                label1.Text = "Folder Selected " + folderPath;
                // Use selected folder path
            }
        }

        private void clearAllDraws() {

            List<Control> exclude = new List<Control> { button1, button2,
button3, label1 };

            this.Controls.Clear();

            foreach (Control c in exclude)
            {
                this.Controls.Add(c);
            }



        }

        private void label1_Click(object sender, EventArgs e)
        {

        }
    }
}
```

3. Class diagram (with Strategy Pattern) [10 marks]

Drawing the folders in two different ways represents a good case for the Strategy Design Pattern. Draw a class

diagram showing the application using this pattern. You have to show all components including the Application class


Drawer <Static>
+createLabel(string, long, Point , boolean) : void
+createVerticalSpine(Point, int) : Panel
-clearAllDraws() : void
-redrawTree() : void
1
1
Visualizer
+ SelectedFolderPath: string
+ SelectedFolderPath: string
+ ZoomLevel : double
+currentTree : string
+buttonEvents () :void
1
1
Continer <abstract>
+ getName() : string
+ getName() : string
+print() : void
+getSize () : string
+ getLevel() : int
File
-name : string
-level : string
-ext : string
-path : sting
-size : long
+getExt(): string
Folder
-children : List<Continer>
- name: string
-path : string
-level : int
size : long
+ addChilds() : void
+ calculateSize() : void
+ getALLChildsCount() : int
- countChilds(continer) : int
1
VerticalDrawer <Static>
+drawVerticalTree(Folder) : void
+drawRecursiveVert(Continer, indent, ref int) : void
HorizantalDrawer <Static>
+drawHorizantalTree(Folder) : void
+drawRecursiveHoriz(Continer, indent, ref int) :void