תכנות מונחה עצמים ב#C - מטלה 5 מסכמת

עומר כהן- תז. 208715813

טל סטולר תז. 313131880

maim "Program":

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


namespace C_Matala5
{
    class Program
    {
        const int capcityRoot = 2000;
        static void Main(string[] args)
        {
            Folder root = Folder.root;
            Folder curr = root;
            bool quit = false;
            string command, p_command;
            DataFile[] arr = new DataFile[1];
            do
            {
                try
                {
                    Console.Write(curr.GetFullPath().Substring(5) + ">");
                    command = Console.ReadLine();
                    if (command.Substring(0, 2) == "cd")
                    {
                        p_command = command.Substring(3);
                        if (p_command == root.FileName)
                            curr = root;
                        else
                            curr = Folder.Cd(p_command);
                        continue;
                    }
                    if (command.Substring(0, 2) == "FC")
                    {
                        p_command = command.Substring(3,
                       command.Substring(3).IndexOf(' '));
                        command =
                       command.Substring(command.Substring(3).IndexOf(' ')
+ 4);
                        if (curr.FC(p_command, command) == true)
                            Console.WriteLine(" equals");
                        else
                            Console.WriteLine("not equals");
                        continue;
                    }
```

```
                    if (command == "dir")
                    {
                        Console.WriteLine(curr);
                        continue;
                    }
                    if (command.Substring(0, 5) == "mkdir")
                    {
                        if (root.IsFull(capcityRoot))
                        {
                            Console.WriteLine("The root folder is full,
files cannot be added");
                        continue;
                        }
                        p_command = command.Substring(6);
                        curr.MkDir(p_command);
                        continue;
                    }
                    if (command.Substring(0, 4) == "echo")
                    {
                        if (root.IsFull(capcityRoot))
                        {
                            Console.WriteLine("The root folder is full,
files cannot be added");
                        continue;
                        }
                        command = command.Substring(5);
                        arr[arr.Length - 1] =
curr.Mkfile(command.Substring(0,
                        command.IndexOf('>') - 1),
command.Substring(command.IndexOf('>') + 2));
                        Array.Resize(ref arr, arr.Length + 1);
                        continue;
                    }
                }
                catch (FormatException e)
                {
                    Console.WriteLine(e.Message);
                    quit = true;
                }
                catch (NullReferenceException ex)
                {
                    Console.WriteLine(ex.Message);
                    quit = true;
                }
                catch (Exception exs)
                {
                    Console.WriteLine(exs.Message);
                    quit = true;
                }
            } while (!quit);
            Console.WriteLine("The end commend line!\n");
            Array.Sort(arr);
            if (arr[0] != null)
                Console.WriteLine("View all content files sorted by file
size:");
        foreach (DataFile item in arr)
            {
                if (item != null)
                    Console.WriteLine(item);
            }
        }
    }
}
```

Interface "Icompete":

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


namespace C_Matala5
{
    public interface IComplete//checks if the size of a folder or a file
has reached the limit
    {
        bool IsFull(int size);

    }


}
```

DataFile.cs:

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


namespace C_Matala5
{
    public class DataFile : File_AD, IComparable<DataFile>

    {
        private string fileContent;
        public string FileContent
        {
            get
            {
                return fileContent;
            }
            set
            {
                if (IsValidContent(value)) // Checking for correctness of
the file content
                {
                    fileContent = value;
                }
                else
                {
                    throw new ArgumentException("Invalid file content.");
                }
            }
        }
```

```
        public DataFile(string fileName, string fileContent)
        : base(fileName)
        {
            FileContent = fileContent;
        }

        private bool IsValidContent(string fileContent)//checks if the
content of a data file is valid
        {
            return true;
        }

        public override float GetSize()//returns the size of the data file
in kilobytes
        {
            return (float)fileContent.Length / 1024;
        }

        public override string ToString()
        {
            return base.ToString() + ", File Size: " +
this.GetSize().ToString("F2") + " KB";
        }

        public override bool Equals(object obj)
        {
            if (obj is DataFile)
            {
                DataFile other = obj as DataFile;
                return this.FileName == other.FileName && this.FileContent
== other.FileContent;
            }

            return false;
        }

        public int CompareTo(DataFile other)//compares the size of two
data files
        {
            if (other == null)
            {
                return 1;
            }
            return this.GetSize().CompareTo(other.GetSize());
        }
    }
}
```

File_AD.cs:

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


namespace C_Matala5
{
    public abstract class File_AD
    {
        private string fileName;
        private DateTime lastUpdateTime;

        public string FileName
        {
            get
            {
                return fileName;
            }
            set
            {
                if (IsValidFileName(value))
                {
                    fileName = value;
                }
                else
                {
                    throw new ArgumentException("Invalid file name.");
                }
            }
        }

        public DateTime LastUpdateTime
        {
            get
            {
                return lastUpdateTime;
            }
            private set
            {
                lastUpdateTime= value;
            }
        }

        public File_AD(string fileName)
        {
            FileName = fileName;
            lastUpdateTime = DateTime.Now;
        }
        private bool IsValidFileName(string fileName) // checks if a file
name is valid
        {

            if (string.IsNullOrEmpty(fileName))
            {
                return false;
            }
```

```
            foreach (char invalidChar in new char[] { ')', '>', '?', '*',
':', '/', '\\', '|', '<' })
            {
                if (fileName.Contains(invalidChar))
                {
                    return false;
                }
            }

            return true;
        }

        public override string ToString()
        {
            return "FileName: " + FileName + ", Last Update: " +
lastUpdateTime.ToString();

        }

        public abstract float GetSize();// abstract method to get the size

        public override bool Equals(object obj)
        {
            if (obj is File_AD)
            {
                File_AD other = obj as File_AD;
                return this.FileName == other.FileName;
            }

            return false;
        }
    }
}
```

Folder.cs:

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

namespace C_Matala5
{
    public class Folder : File_AD, IComplete
    {
        private File_AD[] files;
        private string path;
        public static Folder root = new Folder("root", "");

        public Folder(string fileName, string path) : base(fileName)
        {
            this.path = path;
            this.files = new File_AD[0];
        }

        public string GetFullPath() // returns the full path of the folder
        {
            return this.path + "\\" + this.FileName;
```

```
        }

        public override float GetSize() //returns the total size of all
the files in the folder
        {
            float totalSize = 0;
            foreach (var file in this.files)
            {
                totalSize += file.GetSize();
            }
            return totalSize;
        }

        public bool IsFull(int size) //checks if the total size is too big
        {
            return this.GetSize() >= size;
        }

        public void AddFileToArray(File_AD file) //adds a new file to the
folder
        {
            if (file == null)
            {
                throw new ArgumentNullException(nameof(file), "file cannot
be null");
            }

            if (this.files.Any(existingFile => existingFile.Equals(file)))
            {
                throw new Exception("file already exists");
            }

            Array.Resize(ref this.files, this.files.Length + 1);
            this.files[this.files.Length - 1] = file;
        }

        public DataFile Mkfile(string fileName, string fileContent)
//creates a new data file in the folder
        {
            try
            {
                DataFile newFile = new DataFile(fileName, fileContent);
                this.AddFileToArray(newFile);
                return newFile;
            }
            catch (ArgumentOutOfRangeException ex)
            {
                Console.WriteLine("Error while creating file: " +
ex.Message);
                return null;
            }
        }

        public Folder MkDir(string folderName) // create new folder inside
        {
            Folder newFolder = new Folder(folderName, this.GetFullPath());
            this.AddFileToArray(newFolder);
            return newFolder;
        }

        public override string ToString()
        {
            string result = "";
```

```
            foreach (File_AD file in files)
            {
                if (file is DataFile)
                {
                    result += "[" + file.FileName + "] [" +
file.LastUpdateTime + "] <" + ((DataFile)file).GetSize() + " KB>\n";
                }
                else if (file is Folder)
                {
                    result += "[" + file.FileName + "] [" +
file.LastUpdateTime + "] <DIR>\n";
                }
            }

            return result;
        }

        public static Folder Cd(string path, Folder startingFolder = null)
//navigates to a specific path
        {
            if (string.IsNullOrEmpty(path))
            {
                return root;
            }

            if (startingFolder == null)
            {
                startingFolder = root;
            }

            if (path.StartsWith("\\"))
            {
                startingFolder = root;
                path = path.Substring(1);
            }

            string[] eachPartOfPath = path.Split('\\');
            Folder currentFolder = startingFolder;

            foreach (string part in eachPartOfPath)
            {
                Folder nextFolder = null;

                foreach (File_AD file in currentFolder.files)
                {
                    if (file is Folder folder && folder.FileName == part)
                    {
                        nextFolder = folder;
                        break;
                    }
                }

                if (nextFolder == null)
                {
                    return null;
                }

                currentFolder = nextFolder;
            }

            return currentFolder;
        }
```

```
        public override bool Equals(object obj)  // Override the Equals
method and check if equals
        {
            if (obj is Folder)
            {
                Folder other = obj as Folder;
                if (!this.FileName.Equals(other.FileName))
                    return false;

                if (this.files.Length != other.files.Length)
                    return false;

                for (int i = 0; i < this.files.Length; i++)
                {
                    if (!this.files[i].Equals(other.files[i]))
                        return false;
                }
                return true;
            }
            return false;
        }

        public bool FC(string source, string dest)//compares two files in
the file system
        {
            File_AD FileA = null, FileB = null;
            string fileNameSource = source.Substring(0,
source.LastIndexOf("\\") + 1);
            string filenameDest = dest.Substring(dest.LastIndexOf("\\") +
1);
            string folderPathSource = source.Substring(0,
source.LastIndexOf("\\"));
            string folderPathDest = dest.Substring(0,
dest.LastIndexOf("\\"));
            foreach (File_AD ad in Cd(folderPathSource).files)
            {

                if (ad is DataFile)
                    FileA = (DataFile)ad;
                else
                    FileA = (Folder)ad;
                continue;
            }
            if (FileA == null)
                throw new Exception("There are no files with that name in
this source.");
            foreach (File_AD ad in Cd(folderPathDest).files)
                if (ad.FileName == filenameDest)
                {
                    if (ad is DataFile)
                        FileB = (DataFile)ad;
                    else
                        FileB = (Folder)ad;
                    continue;

                }
            if (((FileA is DataFile) && (FileB is Folder)) || ((FileA is
Folder) && (FileB is DataFile)))
                throw new Exception("It is not possible for the folder and
data file to be identical.");
            return FileA.Equals(FileB);
```

```
        }
    }
}
```

The outputs:

```
>mkdir c
>cd c
\c>mkdir users
\c>mkdir omer
\c>echo this is a beautifull day > firstFile.txt
\c>cd c\users
\c\users>mkdir firstFolder
\c\users>echo its rainy outside > secondFile.txt
\c\users>cd root
>dir
[c] [04/06/2023 21:40:46] <DIR>

>cd c
\c>dir
[users] [04/06/2023 21:40:52] <DIR>
[omer] [04/06/2023 21:40:57] <DIR>
[this is a beautifull day] [04/06/2023 21:41:16] <0.01269531 KB>

\c>cd c\users
\c\users>dir
[firstFolder] [04/06/2023 21:41:41] <DIR>
[its rainy outside] [04/06/2023 21:42:14] <0.01367188 KB>

\c\users>cd c\users\firstFolder
\c\users\firstFolder>dir

\c\users\firstFolder>
```

```
\c\users>cd c\users\firstFolder
\c\users\firstFolder>dir

\c\users\firstFolder>cd root
>cd c
\c>mkdir secondFolder
\c>mkdir thirdFolder
\c>cd c\secondFolder
\c\secondFolder>echo its sunny outside > sunnyDay.txt
\c\secondFolder>cd c
\c>cd c\thirdFolder
\c\thirdFolder>echo i am trying to make them equal > equalFiles.txt
\c\thirdFolder>cd c\firstFolder
>cd c\firstFolder
>cd c\secondFolder
\c\secondFolder>echo i am trying to make them equal > equalFiles2.txt

\c\secondFolder>
```

```
\c\users>cd c\users\firstFolder
\c\users\firstFolder>dir

\c\users\firstFolder>cd root
>cd c
\c>mkdir secondFolder
\c>mkdir thirdFolder
\c>cd c\secondFolder
\c\secondFolder>echo its sunny outside > sunnyDay.txt
\c\secondFolder>cd c
\c>cd c\thirdFolder
\c\thirdFolder>echo i am trying to make them equal > equalFiles.txt
```