Aufgabe 3 – Dateisystem als Klassen

Lösungsidee:

Ich verwende für diese Aufgabe das Composite Pattern, indem ich eine Basis Klasse namens Entity implementiere und jeweils Folder als auch File erben von dieser Basis Klasse, sodass Folder eine Sammlung an Entities hat und Folder und Files gleich behandelt werden können für Operationen wie Move, Delete, Add, ….

Zeitaufwand: ~2h

Code:

```
unit EntityUnit;

interface

uses sysUtils;

type
  EntityType = (FileType, FolderType);

  EntityPtr = ^EntityObj;
  EntityObj = object
  public
    constructor Init(name: string; entityType: EntityType);
    destructor Done; virtual;

    function AsString: string; virtual;

  public
    name: string;
    entityType: EntityType;
    dateModified: TDateTime;
  end;

implementation

constructor EntityObj.Init(name: string; entityType: EntityType);
begin
  self.name := name;
  self.entityType := entityType;
  dateModified := Now;
end;

destructor EntityObj.Done;
begin
end;
```

```
function EntityObj.AsString: string;
var
  typeStr: string;
begin
  case entityType of
    FileType: typeStr := 'file';
    FolderType: typeStr := 'folder';
  else
    typeStr := 'undefiend';
  end;

  AsString := 'name: ' + name + ', type: ' + typeStr + ', dateModified: ' +
DateTimeToStr(dateModified);
end;

end.
```

```
unit FileUnit;

interface

uses EntityUnit;

type
  FilePtr = ^FileObj;
  FileObj = object(EntityObj)
    size: longint;

    constructor Init(name: string; size: longInt);
    destructor Done; virtual;

    function AsString: string; virtual;
  end;

function NewFile(name: string; size: longInt): FilePtr;

implementation

function NewFile(name: string; size: longInt): FilePtr;
var
  f: FilePtr;
begin
  New(f, Init(name, size));
  NewFile := f;
end;
```

```
constructor FileObj.Init(name: string; size: longInt);
begin
  self.size := size;
  inherited Init(name, FileType);
end;

destructor FileObj.Done;
begin
  inherited Done;
end;

function FileObj.AsString: string;
var
  sizeStr: string;
begin
  Str(size, sizeStr);
  AsString := inherited + ' ,size: ' + sizeStr;
end;

end.
```

```
unit FolderUnit;

interface

uses EntityUnit, FileUnit, StringBuilderUnit;

const
  MAX_FOLDER_SIZE = 50;

type
  FolderPtr = ^FolderObj;
  FolderObj = object(EntityObj)
  public
    constructor Init(name: string);
    destructor Done; virtual;

    procedure Add(entity: EntityPtr);
    function Remove(name: STRING): EntityPtr;
    procedure Delete(name: STRING);
    procedure Move(name: string; destination: FolderPtr);
    function Size: longInt;

    function AsString: string; virtual;
  private
    children: array[0..MAX_FOLDER_SIZE] of EntityPtr;
    count: integer;
    function FindEmptySlot: integer;
    function FindIndexByName(name: string): integer;
  end;

function NewFolder(name: string): FolderPtr;

implementation

function NewFolder(name: string): FolderPtr;
var
  f: FolderPtr;
begin
  New(f, Init(name));
  NewFolder := f;
end;

constructor FolderObj.Init(name: string);
begin
  count := 0;
  inherited Init(name, FolderType);
end;

destructor FolderObj.Done;
var
  i: integer;
begin
  inherited Done;
  for i := Low(children) to High(children) do
    if (children[i] <> nil) then
      Dispose(children[i], Done);
end;

procedure FolderObj.Add(entity: EntityPtr);
begin
  if(count = MAX_FOLDER_SIZE) then
  begin
    writeln('ERROR: Max. folder size reached!');
    Halt;
  end;

  children[FindEmptySlot] := entity;
  Inc(count);
end;

function FolderObj.Remove(name: STRING): EntityPtr;
var
  i: Integer;
```

```pascal
begin
  i := FindIndexByName(name);
  if i >= 0 then
  begin
    Remove := children[i];
    children[i] := nil;
    Dec(count);
  end else
    Remove := nil;
end;

procedure FolderObj.Delete(name: STRING);
var
  i: Integer;
begin
  i := FindIndexByName(name);
  if i >= 0 then
  begin
    Dispose(children[i], Done);
    children[i] := nil;
    Dec(count);
  end;
end;

procedure FolderObj.Move(name: string; destination: FolderPtr);
var
  entity: EntityPtr;
begin
  entity := Remove(name);
  if entity <> nil then
    destination^.Add(entity);
end;

function FolderObj.Size: longInt;
var
  i, sum: longInt;
begin
  sum := 0;
  for i := Low(children) to High(children) do
    if (children[i] <> nil) then
      case children[i]^.entityType of
        FileType: sum := sum + FilePtr(children[i])^.size;
        FolderType: sum := sum + FolderPtr(children[i])^.Size;
      end;
  Size := sum;
end;

function FolderObj.FindEmptySlot: Integer;
```

```pascal
var
  i: Integer;
begin
  for i := Low(children) to High(children) do
    if children[i] = nil then
    begin
      FindEmptySlot := i;
      Exit;
    end;
end;

function FolderObj.FindIndexByName(name: string): integer;
var
  i: Integer;
begin
  if(count = 0) then
  begin FindIndexByName := -1; Exit; end;

  for i := Low(children) to High(children) do
    if (children[i] <> nil) and (children[i]^.name = name) then
    begin
      FindIndexByName := i;
      Exit;
    end;
  FindIndexByName := -1;
end;

function FolderObj.AsString: string;
var
  i: integer;
  strBuilder: StringBuilderPtr;
begin
  strBuilder := NewStringBuilder;

  strBuilder^.AppendStr(inherited AsString);
  strBuilder^.AppendStr(', childrenAmount:');
  strBuilder^.AppendInt(count);
  strBuilder^.AppendStr(', size:');
  strBuilder^.AppendLongInt(Size);
  strBuilder^.AppendStr(', children:');

  for i := Low(children) to High(children) do
    if children[i] <> nil then
    begin
      strBuilder^.AppendLine;
      strBuilder^.AppendStr('  ');
      strBuilder^.AppendStr(children[i]^.asString);
    end;
  AsString := strBuilder^.AsString;
  Dispose(strBuilder, Done);
end;

end.
```

Test:

```
program TestFS;

uses
  EntityUnit,
  FileUnit,
  FolderUnit;

var
  file1, file2, file3: FilePtr;
  folder1, folder2, folder3: FolderPtr;

begin
  // Create files
  file1 := NewFile('file1.txt', 100);
  file2 := NewFile('file2.txt', 200);
  file3 := NewFile('file3.txt', 300);

  // Create folders
  folder1 := NewFolder('folder1');
  folder2 := NewFolder('folder2');
  folder3 := NewFolder('folder3');

  // Add files to folder1
  folder1^.Add(file1);
  folder1^.Add(file2);

  // Add folder1 and file3 to folder2
  folder2^.Add(folder1);
  folder2^.Add(file3);

  // Add folder2 to folder3
  folder3^.Add(folder2);

  // Print the initial folder structure
  writeln('Initial Folder Structure:');
  writeln(folder3^.AsString); writeln;

  // Delete file2 from folder1
```

```pascal
  folder1^.Delete('file2.txt');

  // Print the updated folder structure after removing file2
  writeln('Folder Structure after Removing file2:');
  writeln(folder3^.AsString); writeln;

  // Delete folder1 from folder2
  folder2^.Delete('folder1');

  // Print the updated folder structure after deleting folder1
  writeln('Folder Structure after Deleting folder1:');
  writeln(folder3^.AsString); writeln;

  // Delete file3 from folder2
  folder2^.Delete('file3.txt');

  // Print the updated folder structure after removing file3
  writeln('Folder Structure after Removing file3:');
  writeln(folder3^.AsString); writeln;


  // Create folder1 and file1 again and add file1 to folder1
  folder1 := NewFolder('folder1');
  file1 := NewFile('file1.txt', 100);
  folder1^.Add(file1);

  writeln('Folder Structure after create folder1 and file1 again and add file1 
to folder1: ');
  writeln(folder1^.AsString); writeln;

  // Move file1 from folder1 to folder2
  folder1^.Move('file1.txt', folder2);

  // Print the updated folder structure after moving file1
  writeln('Folder Structure after Moving file1:');
  writeln(folder3^.AsString); writeln;
  writeln(folder1^.AsString); writeln;

  // Delete the folder and file objects
  Dispose(folder3, Done);
  Dispose(folder1, Done);
end.
```

```
Initial Folder Structure:
name: folder3, type: folder, dateModified: 30/05/2023 20:32:06, childrenAmount:1, size:600, children:
  name: folder2, type: folder, dateModified: 30/05/2023 20:32:06, childrenAmount:2, size:600, children:
  name: folder1, type: folder, dateModified: 30

Folder Structure after Removing file2:
name: folder3, type: folder, dateModified: 30/05/2023 20:32:06, childrenAmount:1, size:400, children:
  name: folder2, type: folder, dateModified: 30/05/2023 20:32:06, childrenAmount:2, size:400, children:
  name: folder1, type: folder, dateModified: 30

Folder Structure after Deleting folder1:
name: folder3, type: folder, dateModified: 30/05/2023 20:32:06, childrenAmount:1, size:300, children:
  name: folder2, type: folder, dateModified: 30/05/2023 20:32:06, childrenAmount:1, size:300, children:
  name: file3.txt, type: file, dateModified: 30

Folder Structure after Removing file3:
name: folder3, type: folder, dateModified: 30/05/2023 20:32:06, childrenAmount:1, size:0, children:
  name: folder2, type: folder, dateModified: 30/05/2023 20:32:06, childrenAmount:0, size:0, children:

Folder Structure after create folder1 and file1 again and add file1 to folder1:
name: folder1, type: folder, dateModified: 30/05/2023 20:32:06, childrenAmount:1, size:100, children:
  name: file1.txt, type: file, dateModified: 30/05/2023 20:32:06 ,size: 100

Folder Structure after Moving file1:
name: folder3, type: folder, dateModified: 30/05/2023 20:32:06, childrenAmount:1, size:100, children:
  name: folder2, type: folder, dateModified: 30/05/2023 20:32:06, childrenAmount:1, size:100, children:
  name: file1.txt, type: file, dateModified: 30

name: folder1, type: folder, dateModified: 30/05/2023 20:32:06, childrenAmount:0, size:0, children:

Heap dump by heaptrc unit of C:\Repos\2023SS_ADF\UE8\hu3\TestFS.exe
187 memory blocks allocated : 12632/13056
187 memory blocks freed     : 12632/13056
0 unfreed memory blocks : 0
True heap size : 196608 (96 used in System startup)
True free heap : 196512
```