FH-OÖ Hagenberg/HSD
SDP3, WS 2019
Übung 6

Name(1): Daniel Weyrer — Abgabetermin: 17.12.2019

Name(2): Viktoria Streibl — Punkte: 9

Übungsgruppe: 1 — korrigiert: REM

Geschätzter Aufwand in Ph: 12 | 12 — Effektiver Aufwand in Ph: 7 | 7

Beispiel 1 (24 Punkte) Dateisystem-Simulation: Entwerfen Sie aus der nachfolgenden Spezifikation ein Klassendiagramm, instanzieren Sie dieses und implementieren Sie die Funktionalität entsprechend:

Ein Dateisystem für ein einfaches, eingebettetes System besteht aus Dateien, Ordner und Verweise auf Dateien, Ordner oder weitere Verweise. Ein Ordner kann Dateien, Verweise und weitere Ordner beinhalten. Dateien, Ordner und Verweise werden über einen Namen spezifiziert, der verändert werden kann.

Eine Datei hat zusätzlich folgende Eigenschaften:

- aktuelle Dateigröße in Bytes
- Größe eines Blockes auf dem Speichermedium in Bytes
- Anzahl der reservierten Blöcke

Die Größe eines Blockes und die Anzahl der reservierten Blöcke kann für jede Datei bei der Erzeugung unterschiedlich festgelegt werden. Ein nachträgliches Ändern dieser Eigenschaften ist nicht möglich!

Das Schreiben in eine Datei wird durch eine Methode `Write(size_t bytes)` simuliert. Achten Sie darauf, dass die Datei nicht größer werden kann als der für die Datei reservierte Speicher!

Implementieren Sie zur Erzeugung von Dateien, Ordner und Verweise eine Fabrik.

Implementieren Sie einen Visitor (`Dump`) der alle Dateien, Verweise und Ordner in hierarchischer Form ausgibt. Die Ausgabe soll sowohl auf der Standardausgabe als auch in einer Datei möglich sein!

Implementieren Sie einen Visitor (`FilterFiles`) der alle Dateien herausfiltert deren aktuelle Größe innerhalb eines vorgegebenen minimalen und maximalen Wertes liegt. Ein zusätzlicher Filter soll alle Verweise herausfiltern. Die Filter sollen in der Lage sein, alle gefilterten Dateien mit ihrem vollständigen Pfadnamen auszugeben! Bei der Filterung von Verweisen muss zusätzlich auch der Name des Elementes auf das verwiesen wird ausgegeben werden.

Implementieren Sie einen Testtreiber der ein hierarchisches Dateisystem mit mehreren Ebenen erzeugt und die zu implementierenden Besucher ausführlich testet!

Treffen Sie für alle unzureichenden Angaben sinnvolle Annahmen und begründen Sie diese. Verfassen Sie weiters eine Systemdokumentation (Funktionalität, Klassendiagramm, Schnittstellen der beteiligten Klassen, etc.)!

Allgemeine Hinweise: Legen Sie bei der Erstellung Ihrer Übung großen Wert auf eine **saubere Strukturierung** und auf eine **sorgfältige Ausarbeitung!** Dokumentieren Sie alle Schnittstellen und versehen Sie Ihre Algorithmen an entscheidenden Stellen ausführlich mit Kommentaren! Testen Sie ihre Implementierungen ausführlich! Geben Sie den **Testoutput** mit ab!

SDP - Exercise 06

winter semester 2019/20

Viktoria Streibl - S1810306013
Daniel Weyrer - S1820306044

December 17, 2019

Contents

1 Organizational **6**
- 1.1 Team . . . 6
- 1.2 Roles and responsibilities . . . 6
  - 1.2.1 Jointly . . . 6
  - 1.2.2 Viktoria Streibl . . . 6
  - 1.2.3 Daniel Weyrer . . . 6
- 1.3 Effort . . . 6
  - 1.3.1 Viktoria Streibl . . . 6
  - 1.3.2 Daniel Weyrer . . . 7

2 Requirenment Definition(System Specification) **7**

3 System Design **7**
- 3.1 Classdiagram . . . 8
- 3.2 Design Decisions . . . 9
  - 3.2.1 Using a Folder as Root . . . 9
  - 3.2.2 Empty Implementations . . . 9
  - 3.2.3 Add-Function . . . 9
- 3.3 TestDriver . . . 9

4 Test Protocol **10**

5 Source Code **11**
- 5.1 FileSystem . . . 11
  - 5.1.1 FileSystem.h . . . 11
  - 5.1.2 FileSystem.cpp . . . 11
- 5.2 Type . . . 13
  - 5.2.1 Type.h . . . 13
  - 5.2.2 Type.cpp . . . 13
- 5.3 File . . . 15
  - 5.3.1 File.h . . . 15
  - 5.3.2 File.cpp . . . 15
- 5.4 Referral . . . 17
  - 5.4.1 Referral.h . . . 17
  - 5.4.2 Referral.cpp . . . 17
- 5.5 Folder . . . 19
  - 5.5.1 Folder.h . . . 19
  - 5.5.2 Folder.cpp . . . 19
- 5.6 IVisitor . . . 21
  - 5.6.1 IVisitor.h . . . 21
- 5.7 Dump . . . 21
  - 5.7.1 Dump.h . . . 21
  - 5.7.2 Dump.cpp . . . 22
- 5.8 FilterFiles . . . 23
  - 5.8.1 FilterFiles.h . . . 23
  - 5.8.2 FilterFiles.cpp . . . 23

5.9 Factory 26
5.9.1 Factory.h 26
5.9.2 Factory.cpp 26
5.10 TestDriver 27
5.10.1 TestDriver.cpp 27

1 Organizational

1.1 Team

- Viktoria Streibl - S1810306013
- Daniel Weyrer - S1820306044

1.2 Roles and responsibilities

1.2.1 Jointly

- Planning
- Documentation
- Systemdocumentation
- Class Diagram
- Class FileSystem
- TestDriver

1.2.2 Viktoria Streibl

- Visitors
  - Visitor Dump
  - Visitor FilterFiles

1.2.3 Daniel Weyrer

- Base Class Type
- Derived Classes
  - Class File
  - Class Referral
  - Class Folder
- Factory

1.3 Effort

1.3.1 Viktoria Streibl

- estimated: 12 ph
- actually: 7 ph

Saldo sollte ein Hinweis sein Programm & Abgabe nochmal zu überarbeiten

1.3.2 Daniel Weyrer

- estimated: 12 ph
- actually: 7 ph

2 Requirenment Definition(System Specification)

The Filesystem should work like a Linux-Based Filesystem. It can contain Files, Folders and Referrals. The Referrals can refer to a File or a Folder.

Weiters: was soll dann FS können?

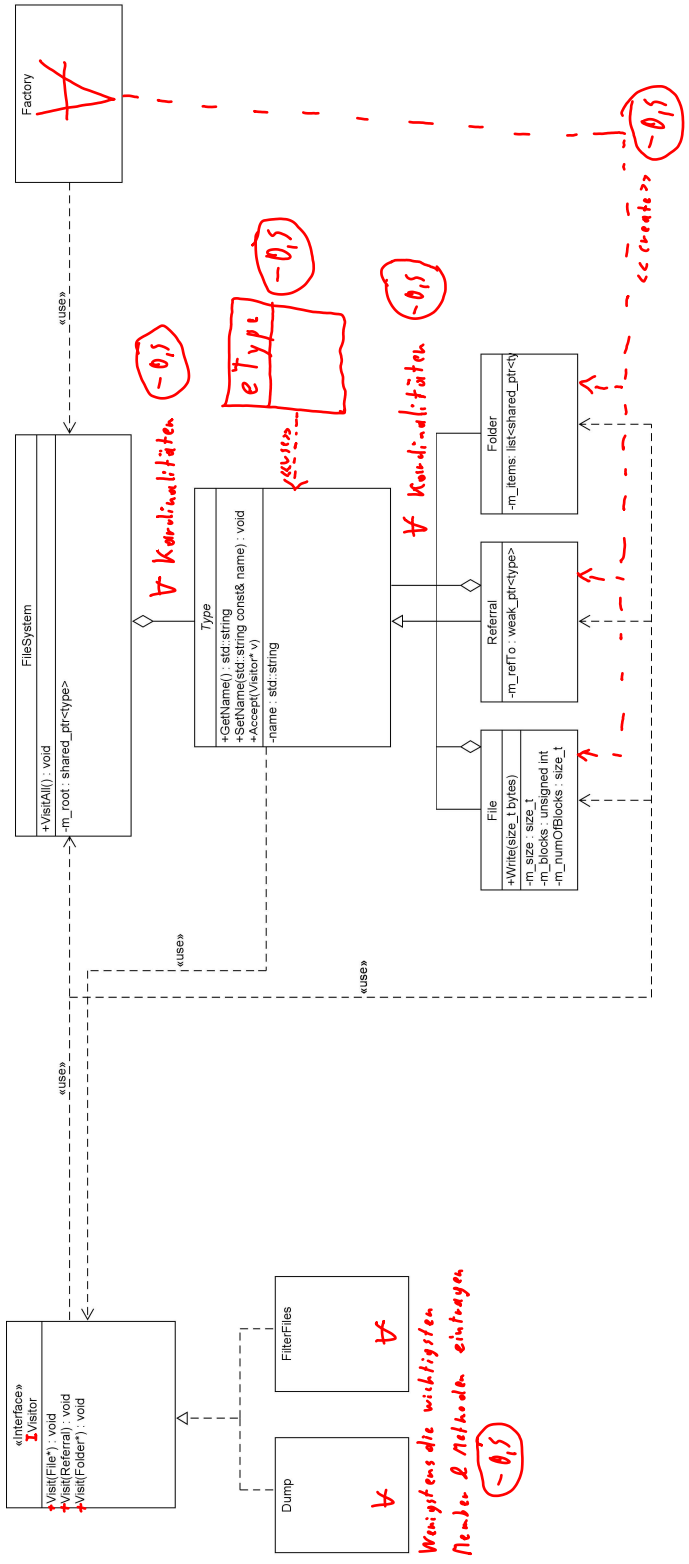
3 System Design

✗ Funktionsbeschreibung

✗ Komponentendiagramm

-2

3.1 Classdiagram



3.2 Design Decisions

3.2.1 Using a Folder as Root

As we wanted to make a Filesystem comparable to the Linux-FS, we chose a Folder as the root of the whole System!

3.2.2 Empty Implementations

As we used pointers of type "Type" we had to make some Functions pure virtual to call them by the pointer. As a File has no list of pointers as Member, we had to keep the Methods empty

3.2.3 Add-Function

We chose to Add Objects by using a path (std::string) as location-giving variable, which comes as close as possible to a "real" linux-filesystem

3.3 TestDriver

```
1  /* ----------------------------------------------------------------------
2  | Workfile : TestDriver.cpp
3  | Description : [ SOURCE ]
4  | Name : Daniel Weyrer      PKZ : S1820306044
5  | Name : Viktoria Streibl   PKZ : S1810306013
6  | Date : 16.12.2019
7  | Remarks : -
8  | Revision : 0
9  | ---------------------------------------------------------------------- */
10
11 #include "FileSystem.h"
12 #include "Type.h"
13 #include "File.h"
14 #include "Folder.h"
15 #include "Referral.h"
16 #include "IVisitor.h"
17 #include "Dump.h"
18 #include "FilterFiles.h"
19
20 #include "vld.h"
21
22 #include <iostream>
23
24 int main() {
25   FileSystem FS{ std::make_shared<Folder>("x") };
26   FS.Add("/", std::make_shared<Folder>("y"));
27   FS.Add("/y/", std::make_shared<Folder>("z"));
28   FS.Add("/y/z/", std::make_shared<File>(1, 1, "file"));
29   FS.Add("/y/", std::make_shared <File>(1, 1, "file1"));
30   FS.Add("/y/", std::make_shared<Referral>(std::make_shared<Folder>("sh"), "Hello"));
31   Dump d;
32   FilterFiles ff;
33   FS.VisitAll(ff);
34   FS.VisitAll(d);
35
36 }
```

✓ leeres Dateisystem Dump, filterFiles, filterLinks

✓ filterLinks

✓ File: writeU, getSizeU, ...

✓ Hinzufügen von: Nullptr, Duplikat

Mindestanforderung an Testtreiben unterschritten

-4

4 Source Code

4.1 FileSystem

4.1.1 FileSystem.h

```
1  /* ----------------------------------------------------------------------
2  | Workfile : FileSystem.h
3  | Description : [ HEADER ]
4  | Name : Daniel Weyrer      PKZ : S1820306044
5  | Name : Viktoria Streibl   PKZ : S1810306013
6  | Date : 09.12.2019
7  | Remarks : -
8  | Revision : 0
9  | ---------------------------------------------------------------------- */
10 #ifndef FILESYSTEM_H
11 #define FILESYSTEM_H
12
13 #include "Type.h"
14 #include "Object.h"
15
16 #include <string>
17 #include <memory>
18
19
20
21 class FileSystem : public Object{
22 public:
23   FileSystem(std::shared_ptr<Type> root) : m_root{ root } {}
24   void Add(std::string const& path, std::shared_ptr<Type> what);
25
26   void VisitAll(IVisitor& v);
27
28   Type::pType GoToPath(std::string const& path) const;
29 private:
30   std::shared_ptr<Type> m_root;
31
32   void VisitRecursive(IVisitor& v, std::shared_ptr<Type>& t);
33 };
34
35 #endif //!FILESYSTEM_H
```

4.1.2 FileSystem.cpp

```
1  /* ----------------------------------------------------------------------
2  | Workfile : FileSystem.cpp
3  | Description : [ SOURCE ]
4  | Name : Daniel Weyrer      PKZ : S1820306044
5  | Name : Viktoria Streibl   PKZ : S1810306013
6  | Date : 09.12.2019
7  | Remarks : -
8  | Revision : 0
9  | ---------------------------------------------------------------------- */
10
11 #include <iostream>
12 #include <algorithm>
13 #include "FileSystem.h"
14
15 bool operator== (std::shared_ptr<Type> const& lhs, std::string const& rhs) {
16   return (lhs->GetName() == rhs);
17 }
18
19 void FileSystem::Add(std::string const& path, std::shared_ptr<Type> what) {
20   try {
21     Type::pType spType;
22
23     //Get Item from Path
24     spType = GoToPath(path);
25     //Add Item at specific location
26     spType->AddItem(what);
```

```
27    }
28
29    catch (std::exception const& ex) {
30      std::cerr << "Error while adding File: " << ex.what() << std::endl;
31    }
32  }
33
34  void FileSystem::VisitAll(IVisitor& v) {
35    VisitRecursive(v, m_root);
36  }
37
38  Type::pType FileSystem::GoToPath(std::string const& path) const {
39    std::string::const_iterator pos1{ std::find(path.cbegin(), path.cend(), '/') };
40    std::string::const_iterator pos = path.cbegin();
41    std::string partOfPath;
42
43    Type::cIterItems cItem;
44    Type::pType spType = m_root;
45
46    //check if path starts with a slash
47    if (pos1++ != path.cbegin()) {
48      throw std::exception("Path needs to begin with a '/'!");
49    }
50    //stay in root when there's just a slash
51    if (path.size() != 1) {
52      //get name of the given directory
53      pos1 = std::find(pos1, path.cend(), '/');
54
55      //loop through path and stop at the end of path string
56      while (pos1 != path.cend()) {
57        partOfPath.assign(++pos, pos1);
58        pos = pos1;
59        cItem = std::find(spType->GetcBegin(), spType->GetcEnd(), partOfPath);
60
61        if (cItem == spType->GetcEnd() || (*cItem)->GetType() != eType::FOLDER) {
62          throw std::exception("Invalid Path!");
63        }
64
65        partOfPath.clear();
66        spType = *cItem;
67
68        pos1 = std::find(++pos1, path.cend(), '/');
69      }
70    }
71    return spType;
72  }
73
74  void FileSystem::VisitRecursive(IVisitor& v, std::shared_ptr<Type>& f) {
75    Type::cIterItems iter = f->GetcBegin();
76    size_t max = std::distance(iter, f->GetcEnd());
77    Type::IterItems pos = f->GetBegin();
78
79    for (size_t i = 0; i < max; i++) {
80      if ((*pos)->GetType() == eType::FILE) {
81        (*pos)->Accept(v);
82      }
83      if ((*pos)->GetType() == eType::FOLDER) {
84        VisitRecursive(v, (*pos));
85      }
86      advance(pos, 1);
87    }
88  }
```

else {
(*pos)->Accept(v);
}

Verbessert Ausgabe erheblich

4.2 Type

4.2.1 Type.h

```
/* ----------------------------------------------------------------------
| Workfile : Type.h
| Description : [ HEADER ]
| Name : Daniel Weyrer      PKZ : S1820306044
| Date : 16.12.2019
| Remarks : -
| Revision : 0
| ---------------------------------------------------------------------- */

#ifndef TYPE_H
#define TYPE_H

#include "IVisitor.h"
#include "Object.h"

#include <string>
#include <memory>
#include <list>

enum class eType {
  FILE,
  FOLDER,
  REFERRAL
};

class Type : public Object {
public:
  typedef std::shared_ptr<Type> pType;
  typedef std::list<std::shared_ptr<Type>>::const_iterator cIterItems;
  typedef std::list<std::shared_ptr<Type>>::iterator IterItems;

  //cTor
  Type() : m_prev{ nullptr } {}

  //Accept for Visitor
  virtual void Accept(IVisitor& v) = 0;

  //Getter and Setter
  std::string GetName() const;
  virtual cIterItems GetcBegin() const = 0;
  virtual IterItems GetBegin() = 0;
  virtual cIterItems GetcEnd() const = 0;

  void SetPrev(Type* const& to);
  Type* GetPrev() const;

  //Adds Item if it isn't contained yet
  virtual void AddItem(std::shared_ptr<Type> const& item) = 0;

  //returns type
  virtual eType GetType() const = 0;

protected:
  std::shared_ptr<IVisitor> m_pVisitor;
  Type* m_prev;
  std::string m_name;
};
#endif//!TYPE_H
```

4.2.2 Type.cpp

```
/* ----------------------------------------------------------------------
| Workfile : Type.cpp
| Description : [ SOURCE ]
| Name : Daniel Weyrer      PKZ : S1820306044
| Date : 16.12.2019
| Remarks : -
| Revision : 0
```

```
8  | ----------------------------------------------------------------------- */
9
10 #include "Type.h"
11
12 std::string Type::GetName() const {
13   return m_name;
14 }
15
16 void Type::SetPrev(Type* const& to) {
17   m_prev = to;
18 }
19
20 Type* Type::GetPrev() const {
21   return m_prev;
22 }
```

set fileName() → Anforderung -1

4.3 File

4.3.1 File.h

```
/* ----------------------------------------------------------------------
| Workfile : File.h
| Description : [ HEADER ]
| Name : Daniel Weyrer      PKZ : S1820306044
| Date : 16.12.2019
| Remarks : -
| Revision : 0
| ---------------------------------------------------------------------- */

#ifndef FILE_H
#define FILE_H

#include "Type.h"
#include <iostream>

class File : public Type {
public:
  File(size_t const blockSize, size_t const numberOfBlocks, std::string const& name) :
    m_size{ 0 },
    m_blockSize{ blockSize },
    m_numberOfBlocks(numberOfBlocks) { m_name = name; }

  //Visitor_func
  virtual void Accept(IVisitor& v) override;

  //Getters
  size_t GetSize() const;
  virtual eType GetType() const override;


  //Writes the number of bytes to the file if there is enough space
  void Write(size_t const bytes);

  //UNUSED FOR FILES!!!
  virtual IterItems GetBegin() override;
  virtual cIterItems GetcBegin() const override;
  virtual cIterItems GetcEnd() const override;
  virtual void AddItem(std::shared_ptr<Type> const& item) override;

private:
  size_t m_size;
  size_t m_blockSize;
  size_t m_numberOfBlocks;
};

#endif //!FILE_H
```

4.3.2 File.cpp

```
/* ----------------------------------------------------------------------
| Workfile : File.cpp
| Description : [ SOURCE ]
| Name : Daniel Weyrer      PKZ : S1820306044
| Date : 16.12.2019
| Remarks : -
| Revision : 0
| ---------------------------------------------------------------------- */

#include "File.h"

void File::Accept(IVisitor& v) {
  v.Visit(*this);
}

Type::IterItems File::GetBegin() {
  return IterItems();
}

```

```
20 Type::cIterItems File::GetcBegin() const {
21   return Type::cIterItems();
22 }
23
24 Type::cIterItems File::GetcEnd() const {
25   return Type::cIterItems();
26 }
27
28 size_t File::GetSize() const {
29   return m_size;
30 }
31
32 eType File::GetType() const {
33   return eType::FILE;
34 }
35
36 void File::AddItem(std::shared_ptr<Type> const& item) {
37   std::cerr << "You Cannot Add a Object to a File!" << std::endl;
38 }
39
40 void File::Write(size_t const bytes) {
41   if (bytes > (m_blockSize * m_numberOfBlocks) - m_size) {
42     std::cerr << "Write not possible - File is not big enough!" << std::endl;
43   }
44   else {
45     m_size += bytes;
46   }
47 }
```

4.4 Referral

4.4.1 Referral.h

```
/* ------------------------------------------------------------------
| Workfile : Referral.h
| Description : [ HEADER ]
| Name : Daniel Weyrer      PKZ : S1820306044
| Date : 16.12.2019
| Remarks : -
| Revision : 0
| ------------------------------------------------------------------ */

#ifndef REFERRAL_H
#define REFERRAL_H
#include "Type.h"

#include <memory>

class Referral : public Type {
public:
  Referral(std::shared_ptr<Type> const& to, std::string const& name);

  //visitorfunc
  virtual void Accept(IVisitor& v) override;

  //Getters
  virtual IterItems GetBegin() override;
  virtual cIterItems GetcBegin() const override;
  virtual cIterItems GetcEnd() const override;
  virtual eType GetType() const override;

  //Adds Item to Folder the referral is pointing to
  virtual void AddItem(std::shared_ptr<Type> const& item) override;


private:
  std::shared_ptr<Type> m_ref;
};

#endif //!REFERRAL_H
```

4.4.2 Referral.cpp

```
/* ------------------------------------------------------------------
| Workfile : Referral.cpp
| Description : [ SOURCE ]
| Name : Daniel Weyrer      PKZ : S1820306044
| Date : 16.12.2019
| Remarks : -
| Revision : 0
| ------------------------------------------------------------------ */
#include "Referral.h"

Referral::Referral(std::shared_ptr<Type> const& to, std::string const& name) {
  m_name = name;
  m_ref = to;
}

void Referral::Accept(IVisitor& v) {
  v.Visit(*this);
}

Type::IterItems Referral::GetBegin() {
  return m_ref->GetBegin();
}

Type::cIterItems Referral::GetcBegin() const {
  return m_ref->GetBegin();
}

Type::cIterItems Referral::GetcEnd() const {
```

```
29   return m_ref->GetcEnd();
30 }
31
32 void Referral::AddItem(std::shared_ptr<Type> const& item) {
33   item->AddItem(item);
34 }
35
36 eType Referral::GetType() const {
37   return eType::REFERRAL;
38 }
```

4.5 Folder

4.5.1 Folder.h

```
/* ----------------------------------------------------------------------
| Workfile : Folder.h
| Description : [ HEADER ]
| Name : Daniel Weyrer      PKZ : S1820306044
| Date : 16.12.2019
| Remarks : -
| Revision : 0
| ---------------------------------------------------------------------- */

#ifndef FOLDER_H
#define FOLDER_H

#include "Type.h"

#include <iostream>
#include <memory>
#include <list>

class Folder : public Type {
public:
  Folder(std::string const& name);

  virtual void Accept(IVisitor& v) override;

  virtual IterItems GetBegin() override;
  virtual cIterItems GetcBegin() const override;
  virtual cIterItems GetcEnd() const override;

  virtual eType GetType() const override;

  virtual void AddItem(std::shared_ptr<Type> const& item) override;

private:
  std::list<std::shared_ptr<Type>> m_items;
};

#endif //!FOLDER_H
```

4.5.2 Folder.cpp

```
/* ----------------------------------------------------------------------
| Workfile : Folder.cpp
| Description : [ SOURCE ]
| Name : Daniel Weyrer      PKZ : S1820306044
| Date : 16.12.2019
| Remarks : -
| Revision : 0
| ---------------------------------------------------------------------- */

#include "Folder.h"

Folder::Folder(std::string const& name) {
  m_name = name;
}

void Folder::Accept(IVisitor& v) {
  v.Visit(*this);
}

Type::IterItems Folder::GetBegin() {
  return m_items.begin();
}

Type::cIterItems Folder::GetcBegin() const {
  return (m_items.cbegin());
}

Type::cIterItems Folder::GetcEnd() const {
```

```
29   return m_items.cend();
30 }
31
32
33
34 eType Folder::GetType() const {
35   return eType::FOLDER;
36 }
37
38 void Folder::AddItem(std::shared_ptr<Type> const& item) {
39   //Add Item if it‘s not a nullptr
40   if (item != nullptr){
41
42     if (std::find(m_items.cbegin(), m_items.cend(), item) == m_items.cend()) {
43
44       m_items.emplace_back(item);
45       item->SetPrev(this);
46     }
47     else {
48       std::cerr << "Error: Duplicate!" << std::endl;
49     }
50   }
51   else {
52     std::cerr << "Error while Adding Item" << std::endl;
53   }
54 }
```

4.6 IVisitor

4.6.1 IVisitor.h

```
/* ----------------------------------------------------------------------
| Workfile : IVisitor.h
| Description : [ HEADER ]
| Name : Viktoria Streibl   PKZ : S1810306013
| Date : 16.12.2019
| Remarks : -
| Revision : 0
| ---------------------------------------------------------------------- */

#ifndef IVISITOR_H
#define IVISITOR_H

class File;
class Folder;
class Referral;

class IVisitor {
public:
  virtual void Visit(File& type) = 0;
  virtual void Visit(Folder& type) = 0;
  virtual void Visit(Referral& type) = 0;
  virtual ~IVisitor() = default;
};

#endif //!IVISITOR_H
```

4.7 Dump

4.7.1 Dump.h

```
/* ----------------------------------------------------------------------
| Workfile : Dump.h
| Description : [ HEADER ]
| Name : Viktoria Streibl   PKZ : S1810306013
| Date : 16.12.2019
| Remarks : -
| Revision : 0
| ---------------------------------------------------------------------- */

#ifndef DUMP_H
#define DUMP_H

#include "IVisitor.h"

#include "File.h"
#include "Folder.h"
#include "Referral.h"

class Dump : public IVisitor {
public:
  void Visit(File& type) override;
  void Visit(Folder& type) override;
  void Visit(Referral& type) override;

private:
  int m_currentDepth = 0;
  std::ostream& m_outputType = std::cout;
  Type* m_root = nullptr;

  void Visit();
  void PrintCurrentElement(std::ostream& out, std::string const filename);
  void FindFirstElement(Type& const type);
  void ReadCurrentPath(Type& const type);
};

#endif //!DUMP_H
```

4.7.2 Dump.cpp

```cpp
/* ----------------------------------------------------------------------
| Workfile : File.cpp
| Description : [ SOURCE ]
| Name : Daniel Weyrer      PKZ : S1820306044
| Date : 16.12.2019
| Remarks : -
| Revision : 0
| ---------------------------------------------------------------------- */

#include "File.h"

void File::Accept(IVisitor& v) {
  v.Visit(*this);
}

Type::IterItems File::GetBegin() {
  return IterItems();
}

Type::cIterItems File::GetcBegin() const {
  return Type::cIterItems();
}

Type::cIterItems File::GetcEnd() const {
  return Type::cIterItems();
}

size_t File::GetSize() const {
  return m_size;
}

eType File::GetType() const {
  return eType::FILE;
}

void File::AddItem(std::shared_ptr<Type> const& item) {
  std::cerr << "You Cannot Add a Object to a File!" << std::endl;
}

void File::Write(size_t const bytes) {
  if (bytes > (m_blockSize * m_numberOfBlocks) - m_size) {
    std::cerr << "Write not possible - File is not big enough!" << std::endl;
  }
  else {
    m_size += bytes;
  }
}
```

Falsche Datei (-0,5)

Fehler in richtiger Datei:

Stream muss veränderbar sein → Anforderung (-0,5)
(Ausserdem ungeprüft)

Ausgabe weder formatiert, noch Ordner oder Links ausgegeben (-1)

4.8 FilterFiles

4.8.1 FilterFiles.h

```
/* ------------------------------------------------------------------------
| Workfile : FilterFiles.h
| Description : [ HEADER ]
| Name : Viktoria Streibl   PKZ : S1810306013
| Date : 16.12.2019
| Remarks : -
| Revision : 0
| ------------------------------------------------------------------------ */

#ifndef FILTERFILES_H
#define FILTERFILES_H

#include "IVisitor.h"

#include "File.h"
#include "Folder.h"
#include "Referral.h"

/*
  Implementieren Sie einen Visitor (FilterFiles) der alle Dateien herausfiltert deren aktuelle Größe
  innerhalb eines vorgegebenen minimalen und maximalen Wertes liegt. Ein zusätzlicher Filter
  soll alle Verweise herausfiltern. Die Filter sollen in der Lage sein, alle gefilterten Dateien mit
       ihrem
  vollständigen Pfadnamen auszugeben! Bei der Filterung von Verweisen muss zusätzlich auch der
  Name des Elementes auf das verwiesen wird ausgegeben werden
*/

class FilterFiles : public IVisitor {
public:
  void Visit(File& type) override;
  void Visit(Folder& type) override;
  void Visit(Referral& type) override;

  void SetMinSize(int minSize);
  void SetMaxSize(int maxSize);

private:
  Type* m_root = nullptr;
  int m_minSize = 1;
  int m_maxSize = 5;

  void FindFirstElement(Type& const type);
  std::string GetPath(Type& const type);
  void FilterBySize(Type& const type);
  void FilterByReferral(Type& const type);
  void Print(Type& const type);
};

#endif //!FILTERFILES_H
```

Besser: Bei Konstruktion übergeben

4.8.2 FilterFiles.cpp

```
/* ------------------------------------------------------------------------
| Workfile : FilterFiles.cpp
| Description : [ SOURCE ]
| Name : Viktoria Streibl   PKZ : S1810306013
| Date : 16.12.2019
| Remarks : -
| Revision : 0
| ------------------------------------------------------------------------ */
#include "FilterFiles.h"

void FilterFiles::Visit(File& type) {
  FindFirstElement(type);
  //Filter Data by size
  FilterBySize(*m_root);
}

```

```
17 void FilterFiles::Visit(Folder& type) {
18   FindFirstElement(type);
19   //Filter Data by size
20   FilterBySize(*m_root);
21
22   //Filter Referral
23   FilterByReferral(*m_root);
24 }
25
26 void FilterFiles::Visit(Referral& type) {
27   FindFirstElement(type);
28   //Filter Referral
29   FilterByReferral(*m_root);
30 }
31
32 void FilterFiles::SetMinSize(int minSize) {
33   m_minSize = minSize;
34 }
35 void FilterFiles::SetMaxSize(int maxSize) {
36   m_maxSize = maxSize;
37 }
38
39 std::string FilterFiles::GetPath(Type& const type) {
40   bool isRoot = false;
41   Type* currentType = &type;
42
43   std::string path = "";
44
45   //loop until find root
46   while (!isRoot) {
47     currentType = currentType->GetPrev();
48
49     if (currentType == nullptr) {
50       isRoot = true;
51     }
52     else {
53       path = currentType->GetName() + "/" + path;
54     }
55   }
56   return path;
57 }
58
59 void FilterFiles::FindFirstElement(Type& const type) {
60   bool isRoot = false;
61   Type* currentType = &type;
62
63   //loop until find root
64   while (!isRoot) {
65     currentType = currentType->GetPrev();
66
67     //check if root
68     if (currentType == nullptr) {
69       isRoot = true;
70       m_root = &type;
71     }
72   }
73 }
74 void FilterFiles::FilterBySize(Type& const type) {
75   //check for filetype
76   if (type.GetType() == eType::FOLDER) {
77     //loop through folder
78     Type::cIterItems currentItem = type.GetBegin();
79     while (currentItem != type.GetcEnd()) {
80       FilterBySize(*(*currentItem));
81       currentItem++;
82     }
83   }
84   else if (type.GetType() == eType::REFERRAL) {
85     FilterBySize(*(*type.GetBegin()));
86   }
87   //print file and referral
88   else if (type.GetType() == eType::FILE) {
89     File file = dynamic_cast<File&>(type);
```

```
90      size_t size = file.GetSize();
91
92      if (size >= m_minSize && size <= m_maxSize) {
93        Print(type);
94      }
95    }
96  }
97
98  void FilterFiles::FilterByReferral(Type& const type) {
99    //check for filetype
100   if (type.GetType() == eType::FOLDER) {
101     //loop through folder
102     Type::cIterItems currentItem = type.GetBegin();
103     while (currentItem != type.GetcEnd()) {
104       FilterBySize(*(*currentItem));
105       currentItem++;
106     }
107   }
108   //print file and referral
109   else if (type.GetType() == eType::REFERRAL) {
110     Print(type);
111   }
112 }
113
114 void FilterFiles::Print(Type& const type) {
115   Type* prevType = type.GetPrev();
116   std::string path = GetPath(*prevType);
117   std::cout << path << "/" << type.GetName();
118 }
```

4.9 Factory

4.9.1 Factory.h

```
1  /* ------------------------------------------------------------------------
2  | Workfile : Factory.h
3  | Description : [ HEADER ]
4  | Name : Daniel Weyrer      PKZ : S1820306044
5  | Date : 16.12.2019
6  | Remarks : -
7  | Revision : 0
8  | ------------------------------------------------------------------------ */
9
10 #ifndef FACTORY_H
11 #define FACTORY_H
12
13 class Factory {
14 };
15
16 #endif //!FACTORY_H
```

4.9.2 Factory.cpp

```
1  /* ------------------------------------------------------------------------
2  | Workfile : Factory.cpp
3  | Description : [ SOURCE ]
4  | Name : Daniel Weyrer      PKZ : S1820306044
5  | Date : 16.12.2019
6  | Remarks : -
7  | Revision : 0
8  | ------------------------------------------------------------------------ */
9
10 #include "Factory.h"
```

factory fehlt

-2

4.10 TestDriver

4.10.1 TestDriver.cpp

```
/* ----------------------------------------------------------------------
| Workfile : TestDriver.cpp
| Description : [ SOURCE ]
| Name : Daniel Weyrer      PKZ : S1820306044
| Name : Viktoria Streibl   PKZ : S1810306013
| Date : 16.12.2019
| Remarks : -
| Revision : 0
| ---------------------------------------------------------------------- */

#include "FileSystem.h"
#include "Type.h"
#include "File.h"
#include "Folder.h"
#include "Referral.h"
#include "IVisitor.h"
#include "Dump.h"
#include "FilterFiles.h"

#include "vld.h"

#include <iostream>

int main() {
  FileSystem FS{ std::make_shared<Folder>("x") };
  FS.Add("/", std::make_shared<Folder>("y"));
  FS.Add("/y/", std::make_shared<Folder>("z"));
  FS.Add("/y/z/", std::make_shared<File>(1, 1, "file"));
  FS.Add("/y/", std::make_shared <File>(1, 1, "file1"));
  FS.Add("/y/", std::make_shared<Referral>(std::make_shared<Folder>("sh"), "Hello"));
  Dump d;
  FilterFiles ff;
  FS.VisitAll(ff);
  FS.VisitAll(d);

}
```