# 1 Organisatorisches

## 1.1 Team

- Reinhard Penn, s1110306019
- Bernhard Selymes, s1110306024

## 1.2 Aufteilung

- Reinhard Penn
  - Planung
  - Klassendiagramm
  - Implementierung der Klassen AdressManager, Writer, AsciiWriter, HtmlWriter, Person, Adress
  - Testen alle Klassen
- Bernhard Selymes
  - Planung
  - Klassendiagramm
  - Implementierung der Klassen Reader, PersonReader, AdressReader, Person, Adress
  - Dokumentation

## 1.3 Zeitaufwand

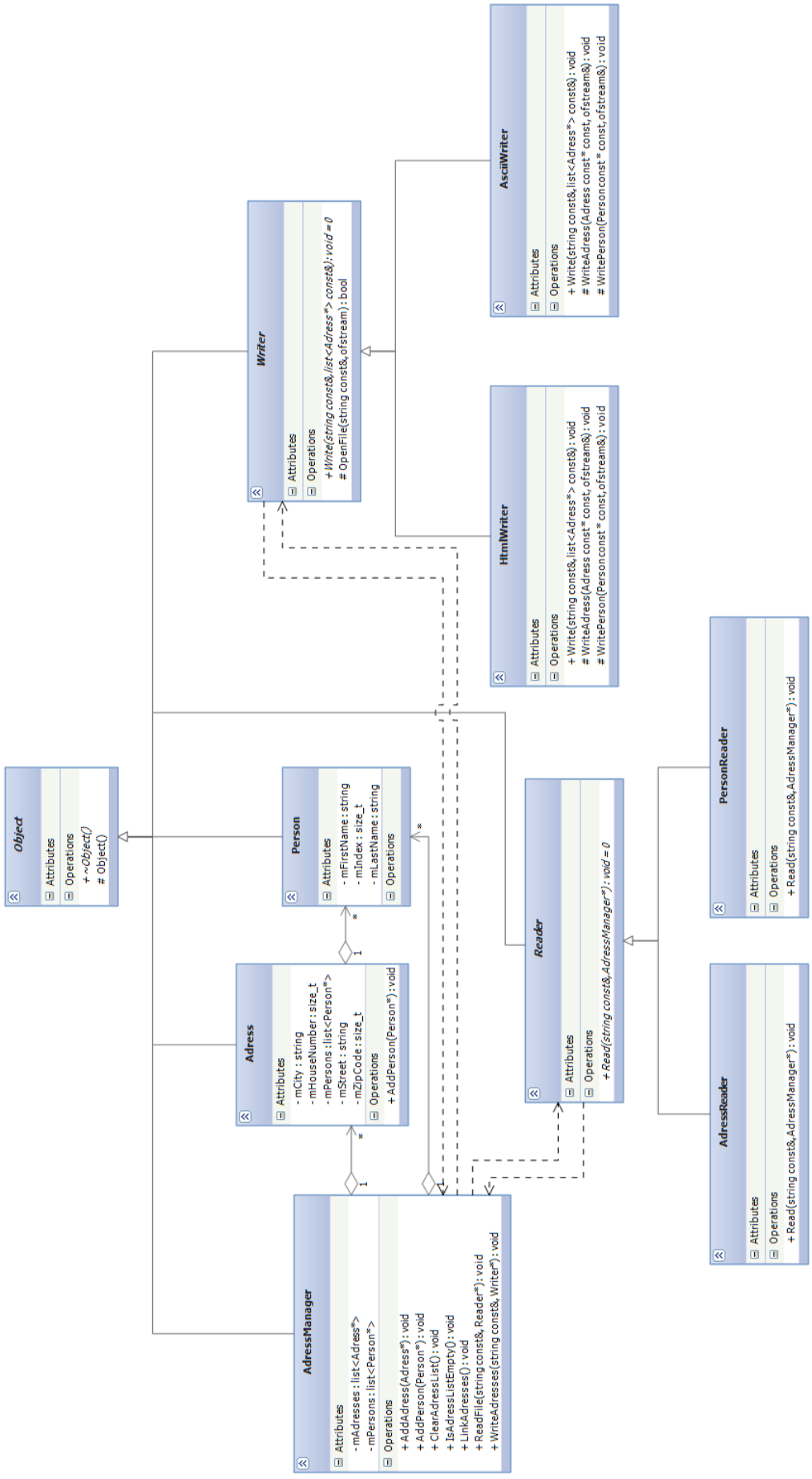
- geschätzte Mh: 7h
- tatsächlich: Reinhard (10h), Bernhard (10h)

# 2 Systemspezifikation

Es soll eine Software für die Verwaltung von Adressen und Personen erstellt werden. Die Daten werden aus verschiedenen Dateien eingelesen, danach werden sie verlinkt und können dann als Html- oder Acsii-Datei ausgegeben werden.
In beiden Dateien gibt es einen Header der überlesen werden kann. Die Einlesenreihenefolge der Dateien ist beliebig. Eine Personendatei enthält den Namen und Index von den Personen, eine Adressdatei Straße, Hausnummer, PLZ und Ort. Die Reihenfolge der Adressen gibt deren Index an.

# 3 Systementwurf

## 3.1 Klassendiagramm



## 3.2 Komponentenübersicht

- Klasse ”Object”:
  Basis aller Basisklassen.
- Klasse ”AdressManager”:
  Beinhaltet die eingelesene Adressen und Personen und kann diese verlinken.
- Klasse ”Adress”:
  Beinhalten die Daten einer Adresse und zusätzlich die Personen die in dieser Adresse wohnen.
- Klasse ”Person”:
  Beinhaltet die Daten einer Person.
- Klasse ”Writer”:
  Abstrakte Basisklasse.
- Klasse ”HtmlWriter”:
  Wird von ”Writer” abgeleitet, gibt die Daten in einer Html-Datei aus.
- Klasse ”AsciiWriter”:
  Wird von ”Writer” abgeleitet, gibt die Daten in einer Ascii-Datei aus.
- Klasse ”Reader”
  Abstrakte Basisklasse.
- Klasse ”AdressReader”
  Wird von ”Reader” abgeleitet, liest die Adressen ein.
- Klasse ”PersonReader”
  Wird von ”Reader” abgeleitet, liest die Personen ein.

# 4 Komponentenentwurf

## 4.1 Klasse ”Object”

Abstrakte Basisklasse aller Klassen. Von ihr werden alle anderen Klassen abgeleitet. Beinhaltet einen virtuellen Destruktor.

## 4.2 Klasse ”AdressManager”

Hat zwei Listen in denen Zeiger von Adressen und Personen gespeichert werden.

**Methode ”AddAdress”:**
Schnittstelle: Parameter: Adress*. Rückgabetyp: void.
Fügt der Liste via pushback neue Adressen hinzu.

**Methode ”AddPerson”:**
Schnittstelle: Parameter: Person*. Rückgabetyp: void.
Fügt der Liste via pushback neue Personen hinzu.

**Methode ”LinkAdresses”:**
Schnittstelle: Rückgabetyp: void.
Geht die Liste mit den Personen durch und fügt sie bei der entsprechenden Adresse ein. Löscht alle Personen die keiner Adresse hinzugefügt werden können.

**Methode ”ReadFile”:**
Schnittstelle: Parameter: string const& , Reader*. Rückgabetyp: void.
Liest Daten mithilfe eines Readers ein.

**Methode ”WriteFile”:**
Schnittstelle: Parameter: string const& , Writer*. Rückgabetyp: void.
Gibt Daten mithilfe eines Writers aus.

**Methode ”ClearAdressList”:**
Schnittstelle: Rückgabetyp: void.
Löscht die Objekte aus der Liste mit den Adressen.

## 4.3 Klasse ”Adress”

Hat folgende Member: ZipCode, City, Street, HouseNumber, Liste mit Personen. Hat für alle eine set- und eine get-Funktion.

**Methode ”AddPerson”:**
Schnittstelle: Parameter: Person*. Rückgabetyp: void.
Fügt der Liste mit den Personen eine Person hinzu.

## 4.4 Klasse ”Person”

Hat folgender Member: FirstName, LastName, Index. Getter und Setter für alle Member.

## 4.5 Klasse ”Reader”

**Methode ”Read”:**
Schnittstelle: Parameter: string const &, AdressManager*. Rückgabetyp: void.
Pure virtual function, die nur die Schnittstelle definiert und nicht implementiert ist.

## 4.6 Klasse ”AdressReader” und Klasse ”PersonReader”

Der Unterschied der beiden Klassen liegt nur darin, dass sich die Daten die eingelesen werden unterscheiden.

**Methode ”Read”:**
Schnittstelle: Parameter: string const &, AdressManager*. Rückgabetyp: void.
Liest mithilfe der getline Funktion Zeile für Zeile Daten aus einer Textdatei. Der Header der Datei wird überlesen. Aus dem String den die getline Funktion liefert, werden die Daten mithilfe verschiedenster string-Funktionen (find_first_of, substr, erase) extrahiert. Zahlen werden mithilfe eines stringstreams in ein size_t umgewandelt. Die Objekte werden in dieser Funktion erzeugt und befüllt und auch gleich zum AdressManager hinzugefügt. Wir haben definiert, dass die Struktur der Daten eingehalten werden muss, wenn falsche Daten in falscher Reihenfolge in der Datei sind, dann werden irgendwelche undefinierte oder zufällige Werte gespeichert.

## 4.7 Klasse ”Writer”

**Methode ”Write”:**
Schnittstelle: Parameter: string const &, TAdresses const&. Rückgabetyp: void.
Pure virtual function, die nur die Schnittstelle definiert und nicht implementiert ist.

**Methode ”OpenFile”:**
Schnittstelle: Parameter: string const &, ofstream&. Rückgabetyp: bool.
Gibt zurück ob die Datei geöffnet werden konnte oder nicht.

## 4.8 Klasse ”HtmlWriter” und ”AsciiWriter”

Die beiden Klassen unterscheiden sich nur in der Formatierung der Ausgabedateien.

**Methode ”Write”:**
Schnittstelle: Parameter: string const &, TAdresses const&. Rückgabetyp: void.
Schreibt Daten in eine Ascii/Html-Datei. Dabei wird die Hilfsfunktion ”WriteAdresses” verwendet. Beim HtmlWriter wird die Formatierung dadurch erreicht, dass Html-Befehle in den filestream geschrieben werden.
**Methode ”WriteAdresses”:**
Schreibt die Adressdaten mit der entsprechenden Formatierung in die Datei. Ruft für alle Personen die in dieser Adresse wohnen ”WritePerson” auf.
**Methode ”WritePersons”:**
Schreibt die Personendaten mit der entsprechenden Formatierung in die Datei.

# 5 Source Code

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : Object.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 5.11.2012
5  // Description : Header for Object.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef OBJECT_H
9  #define OBJECT_H
10
11 class Object
12 {
13 public:
14    //virtual Destructor for baseclass
15    virtual ~Object();
16 protected:
17    //Default CTor for baseclass
18    Object();
19 };
20
21 #endif
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : Object.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 29.10.2012
5  // Description : Baseclass with protected constructor
6  ////////////////////////////////////////////////////////////////////////
7
8  #include "Object.h"
9
10 Object::Object()
11 {}
12
13 Object::~Object()
14 {}
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : AdressManager.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 5.11.2012
5  // Description : Header of AdressManager.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef ADRESSMANAGER_H
9  #define ADRESSMANAGER_H
10
11 #include <string>
12 #include <list>
13 #include <iterator>
14 #include "Object.h"
15 #include "Adress.h"
16 #include "Person.h"
17 #include "Writer.h"
18 #include "Reader.h"
19
20 class AdressManager :
21    public Object
22 {
23 public:
24    ~AdressManager();
25    void AddAdress(Adress* adress);
26    void AddPerson(Person * person);
27    void LinkAdresses();
28    void ReadFile(std::string const& filename, Reader* reader);
29    void WriteAdresses(std::string const& filename, Writer* writer) const;
30    void AdressManager::ClearAdressList();
31    bool IsAdressListEmpty() const;
32
33 private:
34    TAdresses mAdresses;
35    TPersons mPersons;
36 };
37
38 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : AdressManager.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 5.11.2012
// Description : Definition of methods of class AdressManager
////////////////////////////////////////////////////////////////////////

#include <string>
#include <algorithm>
#include <iostream>
#include "AdressManager.h"

AdressManager::~AdressManager()
{
   TAdressesItor itor = mAdresses.begin();

   while (itor != mAdresses.end())
   {
      delete (*itor);
      ++itor;
   }
}


void AdressManager::AddAdress(Adress* adress)
{
   mAdresses.push_back(adress);
}

void AdressManager::AddPerson(Person * person)
{
   mPersons.push_back(person);
}

void AdressManager::LinkAdresses()
{
   TPersonsItor itor = mPersons.begin();
   size_t AdressesSize = mAdresses.size();

   while (itor != mPersons.end())
   {
      TAdressesItor itorAdress = mAdresses.begin();
      size_t currIndex = (*itor)->GetIndex();

      if (AdressesSize <= currIndex)
      {
         TPersonsItor itor_to_delete = itor;
         ++itor;

         delete (*itor_to_delete);
         mPersons.erase(itor_to_delete);

         std::cerr << ("AdressManager.cpp::LinkAdresses: Index is bigger
            than the AdressList")
            << std::endl;
      }
      else
      {
         std::advance(itorAdress,currIndex);

```

```
60            (*itorAdress)->AddPerson(*itor);
61
62            ++itor;
63         }
64      }
65   }
66
67   void AdressManager::ReadFile(std::string const& filename, Reader* reader)
68   {
69      reader->Read(filename,this);
70   }
71
72   void AdressManager::WriteAdresses(std::string const& filename, Writer*
        writer) const
73   {
74      writer->Write(filename,mAdresses);
75   }
76
77   bool AdressManager::IsAdressListEmpty() const
78   {
79      return mAdresses.empty();
80   }
81
82   void AdressManager::ClearAdressList()
83   {
84      TAdressesItor itor = mAdresses.begin();
85
86      while (itor != mAdresses.end())
87      {
88         delete (*itor);
89
90         ++itor;
91      }
92      mAdresses.clear();
93   }
```

```
1  ////////////////////////////////////////////////////////////////////////////
2  // Workfile : Adress.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 5.11.2012
5  // Description : Header of Adress.cpp
6  ////////////////////////////////////////////////////////////////////////////
7
8  #ifndef ADRESS_H
9  #define ADRESS_H
10
11 #include <string>
12 #include <list>
13 #include "Object.h"
14 #include "Person.h"
15
16 typedef std::list<Person*> TPersons;
17 typedef TPersons::const_iterator TPersonsItor;
18
19 class Adress :
20    public Object
21 {
22 public:
23    ~Adress();
24    void AddPerson(Person* person);
25
26    size_t GetZipCode() const;
27    std::string GetCity() const;
28    std::string GetStreet() const;
29    size_t GetHouseNumber() const;
30    TPersons const * GetPersons() const;
31
32    void SetZipCode(size_t const& zipCode);
33    void SetCity(std::string const& city);
34    void SetStreet(std::string const& street);
35    void SetHouseNumber(size_t houseNumber);
36    void SetPersons(TPersons persons);
37 private:
38    size_t mZipCode;
39    std::string mCity;
40    std::string mStreet;
41    size_t mHouseNumber;
42    TPersons mPersons;
43 };
44
45
46
47 #endif
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : Person.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 5.11.2012
5  // Description : Header of Person.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef PERSON_H
9  #define PERSON_H
10
11 #include <string>
12 #include <list>
13 #include "Object.h"
14
15 class Person :
16    public Object
17 {
18 public:
19    size_t GetIndex() const;
20    std::string GetFirstName() const;
21    std::string GetLastName() const;
22
23    void SetIndex(size_t index);
24    void SetFirstName(std::string firstName);
25    void SetLastName(std::string lastName);
26
27 private:
28    size_t mIndex;
29    std::string mFirstName;
30    std::string mLastName;
31 };
32
33
34
35 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : Writer.h
// Author : Reinhard Penn, Bernhard Selymes
// Date : 5.11.2012
// Description : Header of Writer.cpp
////////////////////////////////////////////////////////////////////////

#ifndef WRITER_H
#define WRITER_H

#include <string>
#include <list>
#include <fstream>
#include "Object.h"
#include "Adress.h"

typedef std::list<Adress*>  TAdresses;
typedef TAdresses::const_iterator TAdressesItor;

std::string const header("Adressen mit zugeordneten Personen");

class Writer :
   public Object
{
public:
   virtual void Write(std::string const& filename, TAdresses const&
       adresses) = 0;
protected:
   bool OpenFile(std::string const& filename, std::ofstream& stream);
};

#endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : Writer.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 5.11.2012
// Description : Definition of methods of class Writer
////////////////////////////////////////////////////////////////////////

#include "Writer.h"

bool Writer::OpenFile(std::string const& filename, std::ofstream& stream)
{
   stream.open(filename);
   return stream.is_open();
}
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : HtmlWriter.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 5.11.2012
5  // Description : Header of HtmlWriter.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef HTMLWRITER_H
9  #define HTMLWRITER_H
10
11 #include "Writer.h"
12
13 class HtmlWriter :
14    public Writer
15 {
16 public:
17    ~HtmlWriter();
18    void Write(std::string const& filename, TAdresses const& adresses);
19 private:
20    void WriteAdress(Adress const * const adr, std::ofstream& stream);
21    void WritePerson(Person const * const person, std::ofstream& stream);
22 };
23
24 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : HtmlWriter.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 5.11.2012
// Description : Definition of methods of class HtmlWriter
////////////////////////////////////////////////////////////////////////

#include <string>
#include <iostream>
#include "HtmlWriter.h"

HtmlWriter::~HtmlWriter()
{
}

void HtmlWriter::Write(std::string const& filename, TAdresses const&
    adresses)
{
   try
   {
      std::ofstream stream;

      if (!OpenFile(filename,stream))
      {
         std::string ex("File couldn't be opened");
         throw(ex);
      }

      TAdressesItor itor = adresses.begin();

      stream << "<!DOCTYPE html>" << std::endl;
      stream << "<html>" << std::endl;
      stream << "<body>" << std::endl;
      stream << "<h1>" << header << "</h1>" << std::endl;
      stream << "<hr/>" << std::endl;

      while (itor != adresses.end())
      {
         WriteAdress(*itor, stream);

         ++itor;
      }
      stream << "</body>" << std::endl;
      stream << "</html>" << std::endl;

      stream.close();
   }
   catch(std::string const& ex)
   {
      std::cerr << "HtmlWriter.cpp::Write: " << ex << std::endl;
   }
   catch(...)
   {
      std::cerr << "HtmlWriter.cpp::Write: Unknown Exception occured" <<
          std::endl;
   }
}

void HtmlWriter::WriteAdress(Adress const * const adr, std::ofstream&
    stream)
```

```
58 {
59    stream << "<p><i>" << adr->GetZipCode() << " " << adr->GetCity() << std
          ::endl;
60    stream << "<br/>" << adr->GetStreet() << " " << adr->GetHouseNumber() <<
           "</i>" << std::endl;
61
62    TPersons const * const persons = adr->GetPersons();
63    TPersons::const_iterator person_itor = persons->begin();
64
65    while (person_itor != persons->end())
66    {
67       WritePerson(*person_itor, stream);
68
69       ++person_itor;
70    }
71    stream << "<br/><br/>" << std::endl;
72    stream << "</p>" << std::endl;
73 }
74
75 void HtmlWriter::WritePerson(Person const * const person, std::ofstream&
       stream)
76 {
77    stream << "<br/>" << person->GetFirstName() << " " << person->
          GetLastName() << std::endl;
78 }
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : AsciiWriter.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 5.11.2012
5  // Description : Header of AsciiWriter.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef ASCIIWRITER_H
9  #define ASCIIWRITER_H
10
11 #include "Writer.h"
12
13 std::string const indent("   ");
14 std::string const line("---------------------------------");
15
16 class AsciiWriter :
17    public Writer
18 {
19 public:
20    ~AsciiWriter();
21    void Write(std::string const& filename, TAdresses const& adresses);
22 private:
23    void WriteAdress(Adress const * const adr, std::ofstream& stream);
24    void WritePerson(Person const * const person, std::ofstream& stream);
25 };
26
27 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : AsciiWriter.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 5.11.2012
// Description : Definition of methods of class AsciiWriter
////////////////////////////////////////////////////////////////////////

#include <string>
#include <iostream>
#include "AsciiWriter.h"

AsciiWriter::~AsciiWriter()
{
}

void AsciiWriter::Write(std::string const& filename, TAdresses const&
    adresses)
{
   try
   {
      std::ofstream stream;

      if (!OpenFile(filename,stream))
      {
         std::string ex("File couldn't be opened");
         throw(ex);
      }

      TAdressesItor itor = adresses.begin();

      stream << header << std::endl;
      stream << line << std::endl;

      while (itor != adresses.end())
      {
         WriteAdress(*itor, stream);

         ++itor;
      }
      stream.close();
   }
   catch(std::string const& ex)
   {
      std::cerr << "AsciiWriter.cpp::Write: " << ex << std::endl;
   }
   catch(...)
   {
      std::cerr << "AsciiWriter.cpp::Write: Unknown Exception occured" <<
          std::endl;
   }
}

void AsciiWriter::WriteAdress(Adress const * const adr, std::ofstream&
    stream)
{
   stream << adr->GetZipCode() << " " << adr->GetCity() << std::endl;
   stream << adr->GetStreet() << " " << adr->GetHouseNumber() << std::endl;

   TPersons const * const persons = adr->GetPersons();
   TPersons::const_iterator person_itor = persons->begin();
```

```
58
59      while (person_itor != persons->end())
60      {
61          WritePerson(*person_itor, stream);
62
63          ++person_itor;
64      }
65      stream << std::endl << std::endl;
66  }
67
68  void AsciiWriter::WritePerson(Person const * const person, std::ofstream&
        stream)
69  {
70      stream << indent << person->GetFirstName() << " " << person->GetLastName
            () << std::endl;
71  }
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : Reader.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 5.11.2012
5  // Description : Declaration of abstract Class Reader
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef READER_H
9  #define READER_H
10
11 #include <string>
12 #include "Object.h"
13
14 class AdressManager; //forward declaration
15
16 class Reader :
17    public Object
18 {
19 public:
20    virtual void Read(std::string const& filename, AdressManager*
          adressManager) = 0;
21 };
22
23 #endif
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : PersonReader.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 5.11.2012
5  // Description : Header of PersonReader.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef PERSONREADER_H
9  #define PERSONREADER_H
10
11 #include "Reader.h"
12
13 class PersonReader :
14    public Reader
15 {
16 public:
17    ~PersonReader();
18    void Read(std::string const& filename, AdressManager* adressManager);
19 };
20
21 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////////
// Workfile : PersonReader.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 5.11.2012
// Description : Definition of methods of class PersonReader
////////////////////////////////////////////////////////////////////////////

#include <string>
#include <sstream>
#include <fstream>
#include <iostream>
#include "PersonReader.h"
#include "AdressManager.h"

PersonReader::~PersonReader()
{
}

void PersonReader::Read(std::string const& filename, AdressManager* adressManager)
{
   try
   {
      std::ifstream file(filename);
      std::string buffer;
      size_t pos = 0;   //help variable

      if (!file.is_open())
      {
         std::string ex("File couldn't be opened");
         throw(ex);
      }

      while(!file.eof())
      {
         getline(file,buffer);
         if ((buffer != "") && (buffer[0] != '#'))
         {
            Person* person = new Person;

            pos = buffer.find_first_of(' ');
            person->SetFirstName(buffer.substr(0,pos));
            buffer.erase(0,pos+1);

            pos = buffer.find_first_of(' ');
            person->SetLastName(buffer.substr(0, pos));
            buffer.erase(0,pos+1);

            size_t index;
            std::stringstream (buffer) >> index;   //make string to size_t
            person->SetIndex(index);

            adressManager->AddPerson(person);
         }
      }
      file.close();
   }
   catch(std::string const& ex)
   {
      std::cerr << "PersonReader.cpp::Read: " << ex << std::endl;
```

```
60      }
61      catch(...)
62      {
63         std::cerr << "PersonReader.cpp::Read: Unknown Exception occured" <<
               std::endl;
64      }
65  }
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : AdressReader.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 5.11.2012
5  // Description : Header of AdressReader.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef ADRESSREADER_H
9  #define ADRESSREADER_H
10
11 #include "Reader.h"
12
13 class AdressReader :
14    public Reader
15 {
16 public:
17    ~AdressReader();
18    void Read(std::string const& filename, AdressManager* adressManager);
19 };
20
21 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : AdressReader.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 5.11.2012
// Description : Definition of methods of class AdressReader
////////////////////////////////////////////////////////////////////////

#include <string>
#include <sstream>
#include <fstream>
#include <iostream>
#include "AdressReader.h"
#include "AdressManager.h"

AdressReader::~AdressReader()
{
}

void AdressReader::Read(std::string const& filename, AdressManager* adressManager)
{
   try
   {
      std::ifstream file(filename);
      std::string buffer;
      size_t pos = 0;   //help variable

      if (!file.is_open())
      {
         std::string ex("File couldn't be opened");
         throw(ex);
      }

      if (!adressManager->IsAdressListEmpty())
      {
         adressManager->ClearAdressList();
      }

      while(!file.eof())
      {
         getline(file,buffer);
         // not an empty string
         if ((buffer != "") && (buffer[0] != '#'))
         {
            Adress* adress = new Adress;

            pos = buffer.find_first_of(' ');
            adress->SetStreet(buffer.substr(0,pos));
            buffer.erase(0,pos+1);

            pos = buffer.find_first_of(' ');
            size_t houseNumber;
            std::stringstream (buffer) >> houseNumber;
            adress->SetHouseNumber(houseNumber);
            buffer.erase(0,pos+1);

            pos = buffer.find_first_of(' ');
            adress->SetCity(buffer.substr(0,pos));
            buffer.erase(0,pos+1);

```

```
60              pos = buffer.find_first_of(' ');
61              size_t zipCode;
62              std::stringstream (buffer) >> zipCode;
63              adress->SetZipCode(houseNumber);
64
65              adressManager->AddAdress(adress);
66           }
67        }
68        file.close();
69     }
70     catch(std::string const& ex)
71     {
72        std::cerr << "AdressReader.cpp::Read: " << ex << std::endl;
73     }
74     catch(...)
75     {
76        std::cerr << "AdressReader.cpp::Read: Unknown Exception occured" <<
             std::endl;
77     }
78  }
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : main.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 5.11.2012
5  // Description : Testdriver for the whole program
6  ////////////////////////////////////////////////////////////////////////
7
8  #include <vld.h>
9  #include <iostream>
10 #include "AdressManager.h"
11 #include "PersonReader.h"
12 #include "AdressReader.h"
13 #include "AsciiWriter.h"
14 #include "HtmlWriter.h"
15
16
17 //empty adress & person files
18 void testcase0()
19 {
20    AdressManager* adressManager = new AdressManager;
21    Reader* personReader = new PersonReader;
22    Reader* adressReader = new AdressReader;
23    AsciiWriter* asciiWriter = new AsciiWriter;
24    HtmlWriter* htmlWriter = new HtmlWriter;
25
26    std::string const person1("testcase0_person.txt");
27    std::string const adress("testcase0_adress.txt");
28    std::string const ascii("testcase0_ascii.txt");
29    std::string const html("testcase0_html.html");
30
31
32    std::cout << "Testcase0: Empty adress file and person files" << std::
         endl;
33
34    std::cout << "Read: " << person1 << " ... ";
35    adressManager->ReadFile(person1, personReader);
36    std::cout << "Finished" << std::endl;
37
38    std::cout << "Read: " << adress << " ... ";
39    adressManager->ReadFile(adress, adressReader);
40    std::cout << "Finished" << std::endl;
41
42    std::cout << "LinkAdresses: " << "... ";
43    adressManager->LinkAdresses();
44    std::cout << "Finished" << std::endl;
45
46    std::cout << "Write: " << ascii << " ... ";
47    adressManager->WriteAdresses(ascii, asciiWriter);
48    std::cout << "Finished" << std::endl;
49
50    std::cout << "Write: " << html << " ... ";
51    adressManager->WriteAdresses(html, htmlWriter);
52    std::cout << "Finished" << std::endl;
53
54    std::cout << std::endl << std::endl;
55
56    delete adressManager; adressManager = 0;
57    delete personReader; personReader = 0;
58    delete adressReader; adressReader = 0;
59    delete asciiWriter; asciiWriter = 0;
```

```
60     delete htmlWriter; htmlWriter = 0;
61  }
62
63  //empty person file & valid adress file(single adress)
64  void testcase1()
65  {
66     AdressManager* adressManager = new AdressManager;
67     Reader* personReader = new PersonReader;
68     Reader* adressReader = new AdressReader;
69     AsciiWriter* asciiWriter = new AsciiWriter;
70     HtmlWriter* htmlWriter = new HtmlWriter;
71
72     std::string const person1("testcase0_person.txt");
73     std::string const adress("testcase1_adress.txt");
74     std::string const ascii("testcase1_ascii.txt");
75     std::string const html("testcase1_html.html");
76
77
78     std::cout << "Testcase1: Valid adress file(single adress) "
79        << "and empty person files" << std::endl;
80
81     std::cout << "Read: " << person1 << " ... ";
82     adressManager->ReadFile(person1, personReader);
83     std::cout << "Finished" << std::endl;
84
85     std::cout << "Read: " << adress << " ... ";
86     adressManager->ReadFile(adress, adressReader);
87     std::cout << "Finished" << std::endl;
88
89     std::cout << "LinkAdresses: " << "... ";
90     adressManager->LinkAdresses();
91     std::cout << "Finished" << std::endl;
92
93     std::cout << "Write: " << ascii << " ... ";
94     adressManager->WriteAdresses(ascii, asciiWriter);
95     std::cout << "Finished" << std::endl;
96
97     std::cout << "Write: " << html << " ... ";
98     adressManager->WriteAdresses(html, htmlWriter);
99     std::cout << "Finished" << std::endl;
100
101    std::cout << std::endl << std::endl;
102
103    delete adressManager; adressManager = 0;
104    delete personReader; personReader = 0;
105    delete adressReader; adressReader = 0;
106    delete asciiWriter; asciiWriter = 0;
107    delete htmlWriter; htmlWriter = 0;
108 }
109
110 //valid person file(single person) & valid adress file(single adress)
111 void testcase2()
112 {
113    AdressManager* adressManager = new AdressManager;
114    Reader* personReader = new PersonReader;
115    Reader* adressReader = new AdressReader;
116    AsciiWriter* asciiWriter = new AsciiWriter;
117    HtmlWriter* htmlWriter = new HtmlWriter;
118
119    std::string const person1("testcase2_person.txt");
```

```
120     std::string const adress("testcase1_adress.txt");
121     std::string const ascii("testcase2_ascii.txt");
122     std::string const html("testcase2_html.html");
123 
124 
125     std::cout << "Testcase2: Valid adress file(single adress) "
126        << "and valid person file(single person)" << std::endl;
127 
128     std::cout << "Read: " << person1 << " ... ";
129     adressManager->ReadFile(person1, personReader);
130     std::cout << "Finished" << std::endl;
131 
132     std::cout << "Read: " << adress << " ... ";
133     adressManager->ReadFile(adress, adressReader);
134     std::cout << "Finished" << std::endl;
135 
136     std::cout << "LinkAdresses: " << "... ";
137     adressManager->LinkAdresses();
138     std::cout << "Finished" << std::endl;
139 
140     std::cout << "Write: " << ascii << " ... ";
141     adressManager->WriteAdresses(ascii, asciiWriter);
142     std::cout << "Finished" << std::endl;
143 
144     std::cout << "Write: " << html << " ... ";
145     adressManager->WriteAdresses(html, htmlWriter);
146     std::cout << "Finished" << std::endl;
147 
148     std::cout << std::endl << std::endl;
149 
150     delete adressManager; adressManager = 0;
151     delete personReader; personReader = 0;
152     delete adressReader; adressReader = 0;
153     delete asciiWriter; asciiWriter = 0;
154     delete htmlWriter; htmlWriter = 0;
155 }
156 
157 //valid person file & valid adress file, multiple adresses, persons
158 void testcase3()
159 {
160     AdressManager* adressManager = new AdressManager;
161     Reader* personReader = new PersonReader;
162     Reader* adressReader = new AdressReader;
163     AsciiWriter* asciiWriter = new AsciiWriter;
164     HtmlWriter* htmlWriter = new HtmlWriter;
165 
166     std::string const person1("testcase3_person.txt");
167     std::string const adress("testcase3_adress.txt");
168     std::string const ascii("testcase3_ascii.txt");
169     std::string const html("testcase3_html.html");
170 
171 
172     std::cout << "Testcase3: Valid adress file and valid person file, "
173        << "multiple adresses, persons" << std::endl;
174 
175     std::cout << "Read: " << person1 << " ... ";
176     adressManager->ReadFile(person1, personReader);
177     std::cout << "Finished" << std::endl;
178 
179     std::cout << "Read: " << adress << " ... ";
```

```
180     adressManager->ReadFile(adress, adressReader);
181     std::cout << "Finished" << std::endl;
182 
183     std::cout << "LinkAdresses: " << "... ";
184     adressManager->LinkAdresses();
185     std::cout << "Finished" << std::endl;
186 
187     std::cout << "Write: " << ascii << " ... ";
188     adressManager->WriteAdresses(ascii, asciiWriter);
189     std::cout << "Finished" << std::endl;
190 
191     std::cout << "Write: " << html << " ... ";
192     adressManager->WriteAdresses(html, htmlWriter);
193     std::cout << "Finished" << std::endl;
194 
195     std::cout << std::endl << std::endl;
196 
197     delete adressManager; adressManager = 0;
198     delete personReader; personReader = 0;
199     delete adressReader; adressReader = 0;
200     delete asciiWriter; asciiWriter = 0;
201     delete htmlWriter; htmlWriter = 0;
202 }
203 
204 //valid person files & valid adress file, multiple adresses, persons
205 void testcase4()
206 {
207     AdressManager* adressManager = new AdressManager;
208     Reader* personReader = new PersonReader;
209     Reader* adressReader = new AdressReader;
210     AsciiWriter* asciiWriter = new AsciiWriter;
211     HtmlWriter* htmlWriter = new HtmlWriter;
212 
213     std::string const person1("testcase3_person.txt");
214     std::string const person2("testcase4_person.txt");
215     std::string const adress("testcase3_adress.txt");
216     std::string const ascii("testcase4_ascii.txt");
217     std::string const html("testcase4_html.html");
218 
219 
220     std::cout << "Testcase4: Valid adress file and valid person files, "
221        << "multiple adresses, persons" << std::endl;
222 
223     std::cout << "Read: " << person1 << " ... ";
224     adressManager->ReadFile(person1, personReader);
225     std::cout << "Finished" << std::endl;
226 
227     std::cout << "Read: " << person2 << " ... ";
228     adressManager->ReadFile(person2, personReader);
229     std::cout << "Finished" << std::endl;
230 
231     std::cout << "Read: " << adress << " ... ";
232     adressManager->ReadFile(adress, adressReader);
233     std::cout << "Finished" << std::endl;
234 
235     std::cout << "LinkAdresses: " << "... ";
236     adressManager->LinkAdresses();
237     std::cout << "Finished" << std::endl;
238 
239     std::cout << "Write: " << ascii << " ... ";
```

```
240     adressManager->WriteAdresses(ascii, asciiWriter);
241     std::cout << "Finished" << std::endl;
242 
243     std::cout << "Write: " << html << " ... ";
244     adressManager->WriteAdresses(html, htmlWriter);
245     std::cout << "Finished" << std::endl;
246 
247     std::cout << std::endl << std::endl;
248 
249     delete adressManager; adressManager = 0;
250     delete personReader; personReader = 0;
251     delete adressReader; adressReader = 0;
252     delete asciiWriter; asciiWriter = 0;
253     delete htmlWriter; htmlWriter = 0;
254 }
255 
256 //valid person file & corrupted adress file
257 void testcase5()
258 {
259     AdressManager* adressManager = new AdressManager;
260     Reader* personReader = new PersonReader;
261     Reader* adressReader = new AdressReader;
262     AsciiWriter* asciiWriter = new AsciiWriter;
263     HtmlWriter* htmlWriter = new HtmlWriter;
264 
265     std::string const person1("testcase4_person.txt");
266     std::string const adress("testcase5_adress.txt");
267     std::string const ascii("testcase5_ascii.txt");
268     std::string const html("testcase5_html.html");
269 
270 
271     std::cout << "Testcase5: Corrupted adress file and valid person file" <<
            std::endl;
272 
273     std::cout << "Read: " << person1 << " ... ";
274     adressManager->ReadFile(person1, personReader);
275     std::cout << "Finished" << std::endl;
276 
277     std::cout << "Read: " << adress << " ... ";
278     adressManager->ReadFile(adress, adressReader);
279     std::cout << "Finished" << std::endl;
280 
281     std::cout << "LinkAdresses: " << "... ";
282     adressManager->LinkAdresses();
283     std::cout << "Finished" << std::endl;
284 
285     std::cout << "Write: " << ascii << " ... ";
286     adressManager->WriteAdresses(ascii, asciiWriter);
287     std::cout << "Finished" << std::endl;
288 
289     std::cout << "Write: " << html << " ... ";
290     adressManager->WriteAdresses(html, htmlWriter);
291     std::cout << "Finished" << std::endl;
292 
293     std::cout << std::endl << std::endl;
294 
295     delete adressManager; adressManager = 0;
296     delete personReader; personReader = 0;
297     delete adressReader; adressReader = 0;
298     delete asciiWriter; asciiWriter = 0;
```

```
299     delete htmlWriter; htmlWriter = 0;
300 }
301
302
303 int main()
304 {
305     testcase0();
306     testcase1();
307     testcase2();
308     testcase3();
309     testcase4();
310     testcase5();
311
312     return 0;
313 }
```

# 6 Testausgaben

```
Visual Leak Detector Version 2.2.3 installed.
Testcase0: Empty adress file and person files
Read: testcase0_person.txt ... Finished
Read: testcase0_adress.txt ... Finished
LinkAdresses: ... Finished
Write: testcase0_ascii.txt ... Finished
Write: testcase0_html.html ... Finished


Testcase1: Valid adress file(single adress) and empty person files
Read: testcase0_person.txt ... Finished
Read: testcase1_adress.txt ... Finished
LinkAdresses: ... Finished
Write: testcase1_ascii.txt ... Finished
Write: testcase1_html.html ... Finished


Testcase2:
Valid adress file(single adress) and valid person file(single person)

Read: testcase2_person.txt ... Finished
Read: testcase1_adress.txt ... Finished
LinkAdresses: ... Finished
Write: testcase2_ascii.txt ... Finished
Write: testcase2_html.html ... Finished


Testcase3:
Valid adress file and valid person file, multiple adresses, persons
Read: testcase3_person.txt ... Finished
Read: testcase3_adress.txt ... Finished
LinkAdresses: ... Finished
Write: testcase3_ascii.txt ... Finished
Write: testcase3_html.html ... Finished


Testcase4:
Valid adress file and valid person files, multiple adresses, persons
Read: testcase3_person.txt ... Finished
Read: testcase4_person.txt ... Finished
Read: testcase3_adress.txt ... Finished
LinkAdresses: ... Finished
Write: testcase4_ascii.txt ... Finished
Write: testcase4_html.html ... Finished


Testcase5: Corrupted adress file and valid person file
Read: testcase4_person.txt ... Finished
```

```
Read: testcase5_adress.txt ... Finished
LinkAdresses: ... Finished
Write: testcase5_ascii.txt ... Finished
Write: testcase5_html.html ... Finished


No memory leaks detected.
Visual Leak Detector is now exiting.
```