1 Organisatorisches

1.1 Team

- Reinhard Penn, s1110306019
- Bernhard Selymes, s1110306024

1.2 Aufteilung

- Reinhard Penn
  - Planung
  - Klassendiagramm
  - Implementierung der Klassen Milometer, Speedometer, Observer
  - Testen aller Klassen
- Bernhard Selymes
  - Planung
  - Klassendiagramm
  - Implementierung der Klassen Subject, PKW, RevolutionCounter
  - Dokumentation

1.3 Zeitaufwand

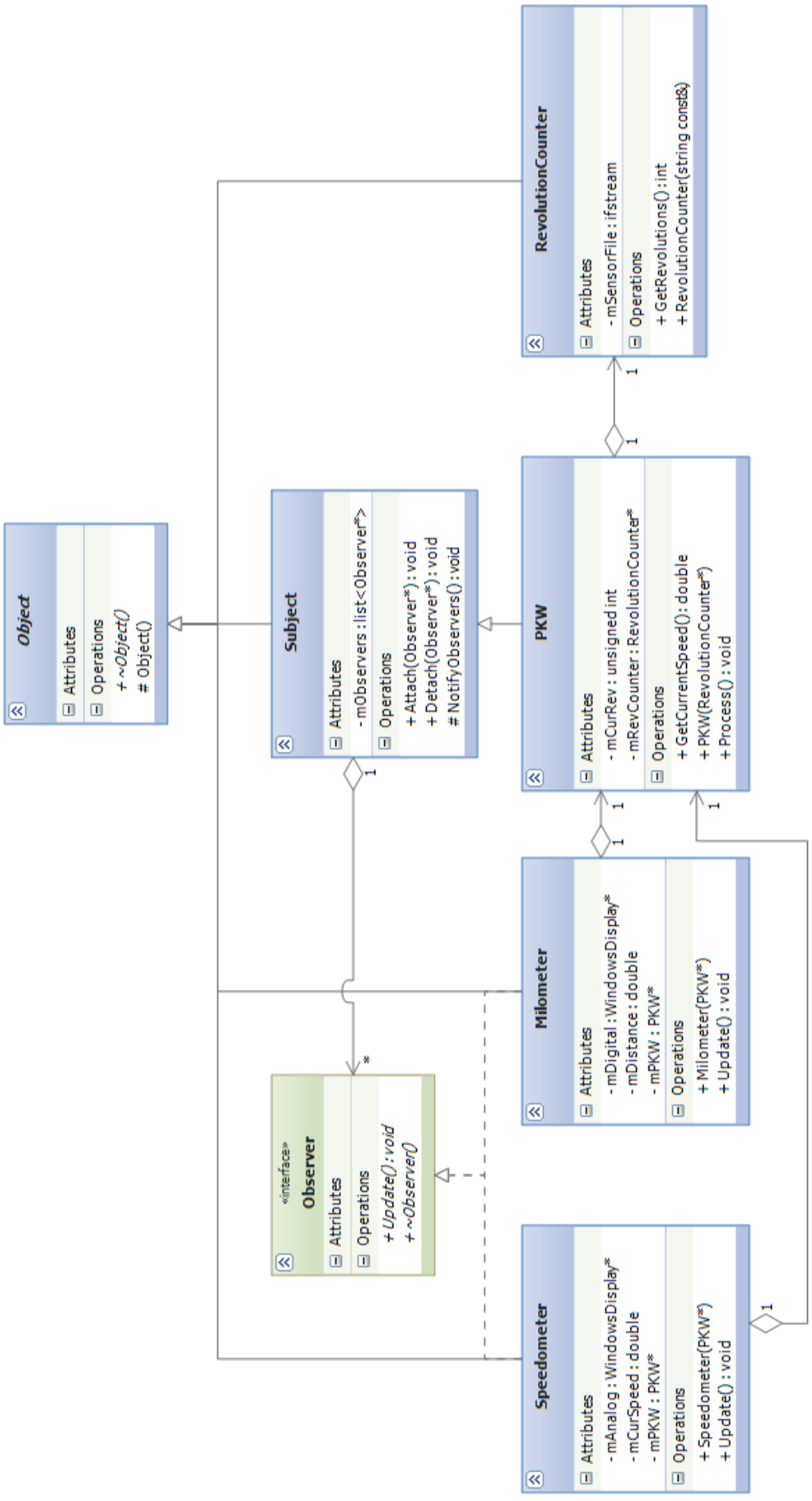
- geschätzte Mh: 10
- tatsächlich: Reinhard (5h), Bernhard (5h)

2 Systemspezifikation

Eine Software für einen PKW-Prüfstand ist zu entwickeln. Die Software soll Raddrehzahlen alle 500ms aus einer Datei einlesen und diese verarbeiten. Sie soll die Momentangeschwindigkeit und die gefahrenen Kilometer berechnen. Die Momentangeschwindigkeit wird auf einem analogen Display, die gefahrenen Kilometer auf einem digitalen Display angezeigt.

3 Systementwurf

3.1 Klassendiagramm



3.2 Komponentenübersicht

- Klasse ”Object”:
  Basis aller Basisklassen.
- Klasse ”Subject”:
  Basisklasse für Subjects, die überwacht werden.
- Interface ”IObserver”:
  Interface für Observer.
- Klasse ”Speedometer”:
  Konkreter Observer, der die Momentangeschwindigkeit ermittelt. Überwacht PKW.
- Klasse ”Milometer”:
  Konkreter Observer, der die gefahrenen Kilometer ermittelt. Überwacht PKW.
- Klasse ”PKW”:
  Konkretes Subject, das überwacht wird.
- Klasse ”RevolutionCounter”:
  Klasse die die Umdrehungen pro Minute zur Verfügung stellt.

4 Komponentenentwurf

4.1 Klasse ”Object”

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

4.2 Klasse ”Subject”

Hat eine Liste mit Zeigern auf Observer. Observer können via Methoden hinzugefügt, entfernt oder benachrichtigt.

Methode ”Attach”:
Schnittstelle:
Parameter: Observer*
Rückgabetyp: void.
Fügt einen noch nicht vorhandenen Observer hinzu.

Methode ”Detach”:
Schnittstelle:
Parameter: Observer*
Rückgabetyp: void.
Entfernt einen Observer, wenn er vorhanden ist.

Methode ”NotifyObservers”:
Schnittstelle:
Rückgabetyp: void.
Ruft von allen Observern Update auf.

4.3 Interface ”IObserver”

Bietet die Schnittstelle für einen Observer. Hat einen virtuellen Destruktor.

Methode ”Update”:
Schnittstelle:
Rückgabetyp: void.
Pure virtual function.

4.4 Klasse ”Speedometer”

Gibt die aktuelle Geschwindigkeit auf einem analogen Display aus.

Konstruktor ”Speedometer”:
Schnittstelle:
Parameter: PKW*
Fügt den PKW hinzu und fügt dem PKW sich selbst zu. Fordert Speicher für ein neues Display an.

Methode ”Update”:
Schnittstelle:
Rückgabetyp: void.
Ruft die Funktion GetCurrentSpeed von PKW auf und gibt den Wert dem analogen Display weiter.

4.5 Klasse ”Milometer”

Gibt die gefahrenen Kilometer auf einem Display aus.

Konstruktor ”Milometer”:
Schnittstelle:
Parameter: PKW*
Fügt den PKW hinzu und fügt dem PKW sich selbst zu. Setzt die Kilometer auf 0. Fordert Speicher für ein neues Display an.

Methode ”Update”:
Schnittstelle:
Rückgabetyp: void.
Ruft die Funktion GetCurrentSpeed von PKW auf, rechnet daraus die in den letzten 500ms gefahrenen Kilometer aus, addiert sie zu den gesamt gefahrenen Kilometern und gibt das Ergebnis dem analogen Display weiter.

4.6 Klasse ”PKW”

Stellt einen konkreten PKW dar. Hat einen Member, der die aktuellen Radumdrehungen speichert und einen der eine Referenz auf einen Radumdrehungszähler hat.

Konstruktor ”PKW”:
Schnittstelle:
Parameter: RevolutionCounter*
Fügt dem PKW einen Radumdrehungszähler hinzu.

Methode ”GetCurrentSpeed”:
Schnittstelle:
Rückgabetyp: double
Berechnet die aktuelle Geschwindigkeit.

Methode ”Process”:
Schnittstelle:
Rückgabetyp: void
Aktualisiert die aktuellen Radumdrehungen und benachrichtigt dann die Observer.

4.7 Klasse ”RevolutionCounter”

Klasse für die Radumdrehungen. Hat einen Member der den Filestream mit den Daten speichert. Im Konstruktor wird dieser Stream geöffnet, im Destruktor geschlossen.

Konstruktor ”RevolutionCounter”:
Schnittstelle:
Parameter: std::string const& filename
Öffnet die gegebene Datei.

Methode ”GetRevolutions”:
Schnittstelle:
Rückgabetyp: int
Liest die Daten aus der Daten aus und gibt sie zurück.

5 Source Code

```
1  ////////////////////////////////////////////////////////////////////////////
2  // Workfile : Object.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 6.11.2012
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 : 6.11.2012
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 : Subject.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 6.11.2012
5  // Description : Header of Subject.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef SUBJECT_H
9  #define SUBJECT_H
10
11 #include "Object.h"
12 #include "IObserver.h"
13 #include <list>
14 #include <iterator>
15
16 typedef std::list<IObserver*> TObservers;
17 typedef TObservers::const_iterator TObserversConstItor;
18
19 class Subject :
20    public Object
21 {
22 public:
23    //adds an observer to observer list
24    void Attach(IObserver* ob);
25
26    //removes an observer from observer list
27    void Detach(IObserver* ob);
28
29 protected:
30    //notify all observers in observer list
31    void NotifyObservers();
32
33 private:
34    TObservers mObservers;
35 };
36
37 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : Subject.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 6.11.2012
// Description : Implementation of class Subject
////////////////////////////////////////////////////////////////////////

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

void Subject::Attach(IObserver* ob)
{
   try
   {
      if(ob == 0)
      {
         std::string ex = "no valid observer pointer";
         throw(ex);
      }
      //searches observer in observer list
      TObserversConstItor found =
         std::find(mObservers.begin(),
         mObservers.end(),ob);
      if (found == mObservers.end()){
         //observer not found in list
         mObservers.push_back(ob) ;
      }
   }
   catch(std::string const& ex)
   {
      std::cerr << "Subject.cpp::Attach: " << ex << std::endl;
   }
   catch(...)
   {
      std::cerr << "Subject.cpp::Attach: Unknown Exception occured" << std
          ::endl;
   }
}

void Subject::Detach(IObserver* ob)
{
   try
   {
      if(ob == 0)
      {
         std::string ex = "no valid observer pointer";
         throw(ex);
      }
      //searches observer in observer list
      TObserversConstItor foundObserver =
         find(mObservers.begin(),
         mObservers.end(), ob);
      if(foundObserver != mObservers.end() ) {
         //observer found in list
         mObservers.erase(foundObserver) ;
      }
   }
   catch(std::string const& ex)
```

```
60      {
61          std::cerr << "Subject.cpp::Detach: " << ex << std::endl;
62      }
63      catch(...)
64      {
65          std::cerr << "Subject.cpp::Detach: Unknown Exception occured" << std
                ::endl;
66      }
67  }
68
69  void Subject::NotifyObservers()
70  {
71      std::for_each(mObservers.begin(),mObservers.end(),[](IObserver* first)
72      {
73          first->Update();
74      });
75  }
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : Observer.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 04.12.2012
5  // Description : Observer interface
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef OBSERVER_H
9  #define OBSERVER_H
10
11 //observer interface
12 class IObserver
13 {
14 public:
15    virtual void Update() = 0 ;
16    virtual ~IObserver() {};
17 };
18
19 #endif
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : Milometer.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 04.12.2012
5  // Description : Header for Milometer.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef MILOMETER_H
9  #define MILOMETER_H
10
11 #include "Object.h"
12 #include "IObserver.h"
13 #include "WindowsDisplay.h"
14 #include "PKW.h"
15
16 unsigned int const cClock = 2;
17
18 class Milometer :
19    public Object,
20    public IObserver
21 {
22 public:
23    //CTor
24    Milometer(PKW* pkw);
25
26    //virtual Destructor
27    virtual ~Milometer();
28
29    //virtual update function
30    virtual void Update();
31
32 private:
33    WindowsDisplay* mDigital;
34    double mDistance;
35    PKW* mPKW;
36 };
37
38 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////////
// Workfile : Milometer.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 04.12.2012
// Description : Implemantation of Milometer
////////////////////////////////////////////////////////////////////////////

#include "Milometer.h"

Milometer::Milometer(PKW* pkw)
{
   try
   {
      if(pkw == 0)
      {
         std::string ex = "no valid pkw pointer";
         throw(ex);
      }
      mPKW = pkw;
      mPKW->Attach(this);
      mDistance = 0;
      mDigital = new DigitalDisplay();
   }
   catch(std::string const& ex)
   {
      std::cerr << "Milometer.cpp::Milometer: " << ex << std::endl;
   }
   catch(...)
   {
      std::cerr << "Milometer.cpp::Milometer: Unknown Exception occured" <<
          std::endl;
   }
}

Milometer::~Milometer()
{
   mPKW->Detach(this);
   delete mDigital;
}

void Milometer::Update()
{
   mDistance = ((mPKW->GetCurrentSpeed()/cHourSec)/cClock) + mDistance;

   mDigital->SendValue((unsigned int)mDistance);
}
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : Speedometer.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 04.12.2012
5  // Description : Header for Speedometer.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef SPEEDOMETER_H
9  #define SPEEDOMETER_H
10
11 #include "Object.h"
12 #include "IObserver.h"
13 #include "WindowsDisplay.h"
14 #include "PKW.h"
15
16 class Speedometer :
17    public Object,
18    public IObserver
19 {
20 public:
21    //CTor
22    Speedometer(PKW* pkw);
23
24    //virtual Destructor
25    virtual ~Speedometer();
26
27    //virtual update function
28    virtual void Update();
29
30 private:
31    WindowsDisplay* mAnalog;
32    double mCurSpeed;
33    PKW* mPKW;
34 };
35
36 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////////
// Workfile : Speedometer.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 04.12.2012
// Description : Implemantation of Speedometer
////////////////////////////////////////////////////////////////////////////

#include "Speedometer.h"

Speedometer::Speedometer(PKW* pkw)
{
   try
   {
      if(pkw == 0)
      {
         std::string ex = "no valid pkw pointer";
         throw(ex);
      }
      mPKW = pkw;
      mPKW->Attach(this);

      mAnalog = new AnalogDisplay();
   }
   catch(std::string const& ex)
   {
      std::cerr << "Speedometer.cpp::Speedometer: " << ex << std::endl;
   }
   catch(...)
   {
      std::cerr << "Speedometer.cpp::Speedometer: Unknown Exception occured
         " << std::endl;
   }
}

Speedometer::~Speedometer()
{
   mPKW->Detach(this);
   delete mAnalog;
}

void Speedometer::Update()
{
   mCurSpeed = mPKW->GetCurrentSpeed();

   mAnalog->SendValue((unsigned int)mCurSpeed);
}
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : PKW.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 6.11.2012
5  // Description : Header of PKW.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef PKW_H
9  #define PKW_H
10
11 #include "Subject.h"
12 #include "RevolutionCounter.h"
13
14 double const cHourSec = 3.6;
15 double const cMinSec = 60;
16 double const cWheelDiameter = 0.6;
17 double const PI = 3.14159265358979323846;
18
19 class PKW :
20    public Subject
21 {
22 public:
23    PKW(RevolutionCounter* revCounter);
24
25    double GetCurrentSpeed();
26    void Process();
27
28 private:
29    unsigned int mCurRev;
30    RevolutionCounter* mRevCounter;
31 };
32
33 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : PKW.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 6.11.2012
// Description : Implementation of class PKW
////////////////////////////////////////////////////////////////////////

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

PKW::PKW(RevolutionCounter* revCounter)
{
   try
   {
      if(revCounter == 0)
      {
         std::string ex = "no valid RevolutionCounter pointer";
         throw(ex);
      }
      mRevCounter = revCounter;

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

double PKW::GetCurrentSpeed()
{
   return ((mCurRev/cMinSec) * cWheelDiameter * PI * cHourSec);
}

void PKW::Process()
{
   int tmp = mRevCounter->GetRevolutions();

   if (tmp >= 0)
   {
      mCurRev = tmp;
      NotifyObservers();
   }
}
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : RevolutionCounter.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 6.11.2012
5  // Description : Header of RevolutionCounter.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef REVOLUTIONCOUNTER_H
9  #define REVOLUTIONCOUNTER_H
10
11 #include <string>
12 #include <fstream>
13 #include "Object.h"
14
15 class RevolutionCounter :
16    public Object
17 {
18 public:
19    RevolutionCounter(std::string const& filename);
20    ~RevolutionCounter();
21
22    int GetRevolutions();
23
24 private:
25    std::ifstream mSensorFile;
26 };
27
28 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////////
// Workfile : RevolutionCounter.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 6.11.2012
// Description : Implementation of class RevolutionCouter
////////////////////////////////////////////////////////////////////////////

#include "RevolutionCounter.h"
#include <string>
#include <iostream>
#include <sstream>

RevolutionCounter::RevolutionCounter(std::string const& filename)
{
   try
   {
      mSensorFile.open(filename);
      if(!mSensorFile.is_open())
      {
         std::string ex = "error in open file";
         throw(ex);
      }
   }
   catch(std::string const& ex)
   {
      std::cerr << "RevolutionCounter.cpp::RevolutionCounter: " << ex <<
          std::endl;
   }
   catch(...)
   {
      std::cerr << "RevolutionCounter.cpp::RevolutionCounter: Unknown
          Exception occured" << std::endl;
   }
}

RevolutionCounter::~RevolutionCounter()
{
   mSensorFile.close();
}

int RevolutionCounter::GetRevolutions()
{
   try
   {
      std::string help;
      getline(mSensorFile,help);

      if((help == "-1") || (help == ""))
      {
         std::string ex = "end of data";
         throw(ex);
      }

      unsigned int rev = 0;
      if(std::istringstream(help) >> rev)
      {
         return rev;
      }
      else
      {
```

```
59          std::string ex = "conversion string to int failed";
60          throw(ex);
61       }
62    }
63    catch(std::string const& ex)
64    {
65       std::cerr << "RevolutionCounter.cpp::GetRevolutions: " << ex << std::
             endl;
66       return -1;
67    }
68    catch(...)
69    {
70       std::cerr << "RevolutionCounter.cpp::GetRevolutions: Unknown
             Exception occured" << std::endl;
71       return -1;
72    }
73 }
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : Main.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 6.11.2012
5  // Description : Testdriver for DrivingSimulation
6  ////////////////////////////////////////////////////////////////////////
7
8  #include <string>
9  #include <fstream>
10 #include <time.h>
11
12 #include <windows.h>
13 #include "WindowsDisplay.h"
14
15 #include "RevolutionCounter.h"
16 #include "PKW.h"
17 #include "Milometer.h"
18 #include "Speedometer.h"
19
20 using namespace std;
21
22 int GetRandVal( int Lo, int Hi)
23 {
24    int Range = Hi - Lo + 1;
25    int Num = rand() / (RAND_MAX + 1) * Range;
26    return Lo + Num;
27 }
28
29 void TestCase(string const& filename, size_t RevEntries, string const&
      Header)
30 {
31    cout << Header << endl;
32
33    RevolutionCounter* revCounter = new RevolutionCounter(filename);
34    PKW* pkw = new PKW(revCounter);
35    Speedometer* speedometer = new Speedometer(pkw);
36    Milometer* milometer = new Milometer(pkw);
37
38    cout << "Process RevEntries: ... ";
39    for (int i = 0; i < RevEntries; i++) {
40       pkw->Process();
41       Sleep(500);
42    }
43    cout << "Done" << endl;
44
45    delete milometer;
46    delete speedometer;
47    delete pkw;
48    delete revCounter;
49
50    cout << endl << endl;
51 }
52
53 void FillFile(string const& filename, size_t RevEntries)
54 {
55    try
56    {
57       //create testfile
58       ofstream Out(filename);
59
```

```
60          if(!Out.is_open())
61          {
62             std::string ex = "TestFile could not be opened";
63             throw(ex);
64          }
65
66          srand((unsigned int)time(0));
67          int Var = 0;
68          int Delta = 8;
69          for (int i = 1; i <= RevEntries; i++)
70          {
71             Out << GetRandVal(550 + Var, 580 + Var) << endl;
72             Var += Delta;
73             if (i % 50 == 0)
74             {
75                Delta = -Delta;
76             }
77          }
78          Out << -1 << endl;
79          Out.close();
80       }
81       catch(std::string const& ex)
82       {
83          std::cerr << "Main.cpp::FillFile: " << ex << std::endl;
84       }
85       catch(...)
86       {
87          std::cerr << "Main.cpp::FillFile: Unknown Exception occured" << std::
                endl;
88       }
89    }
90
91    int main()
92    {
93       string filename = "Not existing";
94       size_t RevEntries = 1;
95
96       TestCase(filename,RevEntries,"Testcase0: File does not exist.");
97
98       filename = "EmptyFile.txt";
99       RevEntries = 0;
100      TestCase(filename,RevEntries,"Testcase1: Empty file without fill
             function.");
101
102      filename = "EmptyFile2.txt";
103      FillFile(filename,RevEntries);
104      TestCase(filename,RevEntries,"Testcase2: Empty file with fill function."
             );
105
106      filename = "InvalidDriveData.txt";
107      RevEntries = 7;
108      TestCase(filename,RevEntries,"Testcase3: File with invalid content.");
109
110      filename = "SingleDriveData.txt";
111      RevEntries = 1;
112      FillFile(filename,RevEntries);
113      TestCase(filename,RevEntries,"Testcase4: File with a single entry.");
114
115      RevEntries = 3;
116      TestCase(filename,RevEntries,"Testcase5: File with a single entry and
```

```
        too much Process calls.");
117
118     filename = "DriveData.txt";
119     RevEntries = 400;
120     FillFile(filename,RevEntries);
121     TestCase(filename,RevEntries,"Testcase6: File with multiple entries.");
122
123     return 0;
124 }
```

6 Testausgaben

```
Testcase0: File does not exist.
RevolutionCounter.cpp::RevolutionCounter: error in open file
Process RevEntries: ... RevolutionCounter.cpp::GetRevolutions:
end of data
Done


Testcase1: Empty file without fill function.
Process RevEntries: ... Done


Testcase2: Empty file with fill function.
Process RevEntries: ... Done


Testcase3: File with invalid content.
Process RevEntries: ... RevolutionCounter.cpp::GetRevolutions:
conversion string to int failed
RevolutionCounter.cpp::GetRevolutions: conversion string to int failed
RevolutionCounter.cpp::GetRevolutions: conversion string to int failed
RevolutionCounter.cpp::GetRevolutions: conversion string to int failed
RevolutionCounter.cpp::GetRevolutions: conversion string to int failed
RevolutionCounter.cpp::GetRevolutions: conversion string to int failed
RevolutionCounter.cpp::GetRevolutions: conversion string to int failed
Done


Testcase4: File with a single entry.
Process RevEntries: ... Done


Testcase5: File with a single entry and too much Process calls.
WindowsClient: Server not online. Waiting...
WindowsClient: Server not online. Waiting...
WindowsClient: Server not online. Waiting...
WindowsClient: Server not online. Waiting...
Process RevEntries: ... RevolutionCounter.cpp::GetRevolutions:
end of data
RevolutionCounter.cpp::GetRevolutions: end of data
Done


Testcase6: File with multiple entries.
Process RevEntries: ... Done
```