1 Organisatorisches

1.1 Team

- Reinhard Penn, s1110306019
- Bernhard Selymes, s1110306024

1.2 Aufteilung

- Reinhard Penn
  - Planung
  - Klassendiagramm
  - Implementierung der Klassen Client, Slot, RemoteControl
  - Testen aller Klassen
- Bernhard Selymes
  - Planung
  - Klassendiagramm
  - Implementierung der Device und Command Klassen
  - Dokumentation

1.3 Zeitaufwand

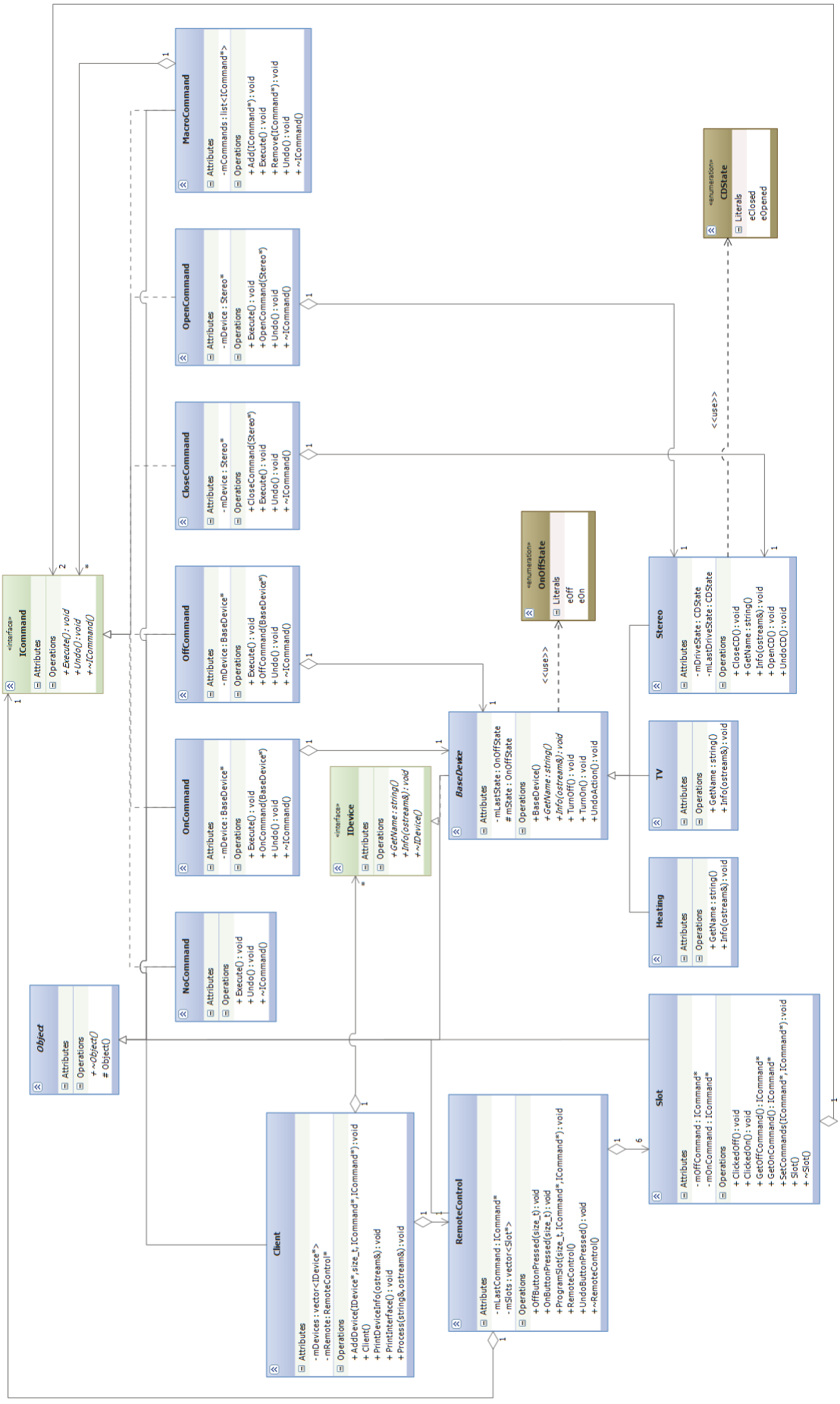
- geschätzte Mh: 15
- tatsächlich: Reinhard (8h), Bernhard (8h)

2 Systemspezifikation

Eine Software für eine programmierbare Fernsteuerung soll entworfen werden. Mit der Fernsteuerung können verschiedene Geräte ein- und ausgeschalten werden. Die Fernbedienung hat 6 Slots die aus je einer On und Off Taste bestehen. Die siebte Taste ist die Undo Taste mit der die letzte Eingabe zurückgenommen werden kann. TV-Geräte, Heizungen und Stereoanlagen können ferngesteuert werden. Alle können ein und ausgeschalten werden, die Stereoanlage zusätzlich geöffnet und geschlossen werden. Ein Kommandozeileninterface und die Geräteinformationen können ausgegeben werden.

3 Systementwurf

3.1 Klassendiagramm



3.2 Komponentenübersicht

- Klasse ”Object”:
  Basis aller Basisklassen.
- Klasse ”Client”:
  Verwaltet die Geräte und kann deren Informationen ausgeben und verarbeitet die Eingaben vom Benutzer.
- Klasse ”RemoteControl”:
  Verwaltet die Slots und kann die Slots programmieren.
- Klasse ”Slot”:
  Verwaltet die Kommandos eines Slots.
- Interface ”ICommand”:
  Schnittstellenbeschreibung für die Kommandos.
- Klassen ”OffCommand, OnCommand, CloseCommand und OpenCommand”:
  Konkrete Kommandoklassen.
- Klasse ”NoCommand”:
  Standard Kommando, das nur etwas ausgibt.
- Klasse ”MacroCommand”:
  Zusammenfassung mehrerer Kommandos.
- Interface ”IDevice”:
  Schnittstellenbeschreibung für die Geräte.
- Klasse ”BaseDevice”:
  Basisklasse für die Geräte.
- Klassen ”Heating, TV und Stereo”:
  Konkrete Geräteklassen.
- Enumeration ”CDState”:
  Status des CD-Laufwerks.
- Enumeration ”OnOffState”:
  Status ob ein- oder ausgeschalten.

4 Komponentenentwurf

4.1 Klasse ”Object”

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

4.2 Klasse ”Client”

Hat eine Liste die die Geräte verwaltet und einen Member der eine Referenz auf die Fernsteuerung speichert.

Konstruktor ”Client”:
Schnittstelle:
Verkleinert die Größe des Vektors auf 6 und initialisiert die Pointer mit Null.

Methode ”AddDevice”:
Schnittstelle:
Parameter: IDevice*, size_t
Rückgabetyp: void
Fügt der Liste ein Gerät hinzu. Falsche Eingaben werden berücksichtigt. Ruft ProgramSlot von der Fernbedienung auf.

Methode ”PrintDeviceInfo”:
Schnittstelle:
Parameter: ostream&
Rückgabetyp: void
Gibt die Informationen der Geräte auf dem mitgegebenen Stream aus.

Methode ”PrintInterface”:
Schnittstelle:
Rückgabetyp: void
Gibt das Kommandozeilen-Interface auf der Konsole aus.

Methode ”Process”:
Schnittstelle:
Parameter: string&
Rückgabetyp: void
Verarbeitet den in der Konsole eingegebenen string und ruft die dazugehörigen Methoden der Fernsteuerung auf.

4.3 Klasse ”RemoteControl”

Hat einen Vektor der Referenzen auf die Slots speichert und einen Member der das letzte Kommando speichert.

Konstruktor ”RemoteControl”:
Erstellt die Slots und setzt die Kommandos auf NoCommand.

Methode ”OffButtonPressed”:
Schnittstelle:
Parameter: size_t
Rückgabetyp: void
Speichert das aktuelle Kommando im Member und ruft das Off-Kommando vom entsprechenden Slot auf.

Methode ”OnButtonPressed”:
Schnittstelle:
Parameter: size_t
Rückgabetyp: void
Speichert das aktuelle Kommando im Member und ruft das On-Kommando vom entsprechenden Slot auf.

Methode ”UndoButtonPressed”:
Schnittstelle:
Rückgabetyp: void
Ruft vom letzten Kommando die Methode Undo auf und setzt den Pointer auf 0, weil nur ein Mal zurückgesetzt werden kann.

Methode ”ProgramSlot”:
Schnittstelle:
Parameter: size_t, ICommand*, ICommand*
Rückgabetyp: void
Mit dieser Methode werden die Slots der Fernbedienung programmiert, das heißt die Kommandos werden zugewiesen.

4.4 Klasse ”Slot”

Speichert einen Pointer auf ein On- und ein Offkommando. Hat 2 Get-Methoden für diese und einen Destruktor der sie freigibt.

Konstruktor ”Slot”:
Weißt den Kommandozeigern 0 zu.

Methode ”ClickedOff”:
Schnittstelle:
Rückgabetyp: void
Überprüft den Pointer und ruft ”Execute” vom Off-Kommando auf.

Methode ”ClickedOn”:
Schnittstelle:
Rückgabetyp: void
Überprüft den Pointer und ruft ”Execute” vom On-Kommando auf.

Methode ”SetCommands”:
Schnittstelle:
Parameter: ICommand*, ICommand*
Rückgabetyp: void

Weist die Kommandos zu.

4.5 Interface ”ICommand”

Schnittstellendefiniton. Hat einen virtuellen Destruktor.

Methode ”Execute”:
Schnittstelle:
Rückgabetyp: void

Methode ”Undo”:
Schnittstelle:
Rückgabetyp: void

4.6 Klassen ”OffCommand, OnCommand, CloseCommand, OpenCommand und NoCommand”

Implementieren die Methoden Execute und Undo entsprechend der jeweiligen Klasse. Bei NoCommand wird einfach auf der Konsole ausgegeben, dass es sich um NoCommand handelt. Die anderen Kommandos rufen die entsprechenden Methoden in den Klassen, auf die sie eine Referenz haben, auf. Sie haben weiters einen Konstruktor dem diese Referenzen mitgegeben werden.

4.7 Klasse ”MacroCommand”

Hat eine Liste die die Referenzen auf mehrere Kommandos speichert. Der Liste können die Kommandos hinzugefügt werden, aber auch wieder entfernt werden. Es können nur maximal 2 Elemente in der Liste gespeichert werden.

4.8 Interface ”IDevice”

Schnittstellendefiniton. Hat einen virtuellen Destruktor. Die vorgegebene Schnittstelle wurde um die Methode ”GetName” erweitert, da diese für ”PrintInterface” benötigt wird.

Methode ”Info”:
Schnittstelle:
Parameter: ostream&
Rückgabetyp: void

Methode ”GetName”:
Schnittstelle:
Rückgabetyp: string

4.9 Klasse ”BaseDevice”

Speichert den aktuellen und den letzten On-Off-Status. Der Konstruktor setzt beide Statusse auf Off. Die Methoden ”TurnOn” und ”TurnOff” speichern immer den aktuellen Status und weisen dann den neuen zu. Bei ”Undo” wird der letzte Status dem neuen zugewiesen. Die Methode ”GetName” ist abstrakt und wird in den Unterklassen überschrieben.

4.10 Klassen ”Heating, TV und Stereo”

Die Methode ”Info” gibt die Informationen des jeweiligen Objektes entsprechend aus. Die Klasse Stereo definiert zusätzlich die für das CD-Laufwerk benötigten Member und Methoden. In der Methode ”GetName” werden die Namen der Klassen entsprechend returniert.

5 Source Code

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : Object.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 11.01.2012
5  // Description : Header of 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 : 11.01.2012
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 : Client.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 11.01.2012
5  // Description : Header of Client.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef CLIENT_H
9  #define CLIENT_H
10
11 #include <vector>
12 #include <iostream>
13 #include <string>
14 #include "Object.h"
15 #include "RemoteControl.h"
16 #include "IDevice.h"
17
18 typedef std::vector<IDevice*> TDevices;
19
20 std::string const DivLine("-----------------------");
21
22 class Client :
23    public Object
24 {
25 public:
26    //CTor
27    Client();
28
29    void AddDevice(IDevice* Device, size_t SlotNumber, ICommand* OnCommand,
       ICommand* OffCommand);
30    void PrintDeviceInfo(std::ostream& stream);
31    void PrintInterface();
32    void Process(std::string& Input, std::ostream& stream = std::cout);
33 private:
34    TDevices mDevices;
35    RemoteControl mRemote;
36 };
37
38 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : Client.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 11.01.2012
// Description : Implemantation of class Client
////////////////////////////////////////////////////////////////////////

#include <algorithm>
#include "Client.h"

Client::Client()
{
   mDevices.resize(MaxSlots,0);
}

void Client::AddDevice(IDevice* Device, size_t SlotNumber, ICommand*
    OnCommand, ICommand* OffCommand)
{
   try
   {
      if (Device == 0)
      {
         throw std::string("Client::AddDevice: Device is a null pointer");
      }
      if (SlotNumber == 0 || SlotNumber > MaxSlots)
      {
         throw std::string("Client::AddDevice: SlotNumber out of range");
      }
      if (OnCommand == 0)
      {
         throw std::string("Client::AddDevice: OnCommand is a null pointer"
            );
      }
      if (OffCommand == 0)
      {
         throw std::string("Client::AddDevice: OffCommand is a null pointer
            ");
      }

      mDevices[SlotNumber] = Device;
      mRemote.ProgramSlot(SlotNumber,OnCommand,OffCommand);
   }
   catch(std::string const& ex)
   {
      throw(ex);
   }
}

void Client::PrintDeviceInfo(std::ostream& stream)
{
   try
   {
      if (stream == 0)
      {
         throw std::string("Client::PrintDeviceInfo: stream is null");
      }

      stream << "Devices:" << std::endl;
      stream << DivLine << std::endl;

```

```
58          std::for_each(mDevices.begin(),mDevices.end(),[&](IDevice* d)
59          {
60             if (d != 0)
61             {
62                d->Info(stream);
63             }
64          });
65       }
66       catch(std::string const& ex)
67       {
68          throw(ex);
69       }
70    }
71
72    void Client::PrintInterface()
73    {
74       std::cout << "Remote control:" << std::endl;
75       std::cout << DivLine << std::endl;
76
77       for (int i=0; i<MaxSlots; ++i)
78       {
79          std::cout << i+1 << "...";
80          mDevices[i] != 0 ? std::cout << mDevices[i]->GetName() : std::cout <<
               "empty";
81          std::cout << std::endl;
82       }
83       std::cout << "u...undo" << std::endl;
84       std::cout << "i...output device info" << std::endl;
85       std::cout << DivLine << std::endl;
86       std::cout << "input slot number and on('o') or off('f'):" << std::endl;
87    }
88
89    void Client::Process(std::string& Input, std::ostream& stream)
90    {
91       try
92       {
93          if (Input.empty())
94          {
95             throw std::string("Client::Process: Input is null");
96          }
97          if (Input[0] == 'u')
98          {
99             mRemote.UndoButtonPressed();
100         }
101         else if (Input[0] == 'i')
102         {
103            PrintDeviceInfo(stream);
104         }
105         else if (Input.size() >= 2)
106         {
107            bool InputValid = false;
108
109            for (int i=0; i<MaxSlots; ++i)
110            {
111               if (Input[0] == (char)(i+'1'))
112               {
113                  if (Input[1] == 'o')
114                  {
115                     InputValid = true;
116                     mRemote.OnButtonPressed(i+1);
```

```
117                 }
118                 else if (Input[1] == 'f')
119                 {
120                    InputValid = true;
121                    mRemote.OffButtonPressed(i+1);
122                 }
123                 else
124                 {
125                    throw std::string("Client::Process: Input is not valid");
126                 }
127              }
128           }
129           if (!InputValid)
130           {
131              throw std::string("Client::Process: Input is not valid");
132           }
133        }
134        else
135        {
136           throw std::string("Client::Process: Input is not valid");
137        }
138     }
139     catch(std::string const& ex)
140     {
141        throw(ex);
142     }
143  }
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : RemoteControl.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 11.01.2012
5  // Description : Header of RemoteControl.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef REMOTECONTROL_H
9  #define REMOTECONTROL_H
10
11 #include <vector>
12 #include "Object.h"
13 #include "Slot.h"
14
15 typedef std::vector<Slot*> TSlots;
16 size_t const MaxSlots = 6;
17
18 class RemoteControl :
19    public Object
20 {
21 public:
22    //CTor
23    RemoteControl();
24
25    //DTor
26    ~RemoteControl();
27
28    void OnButtonPressed(size_t SlotNumber);
29    void OffButtonPressed(size_t SlotNumber);
30    void ProgramSlot(size_t SlotNumber, ICommand* OnCommand, ICommand*
          OffCommand);
31    void UndoButtonPressed();
32
33 private:
34    ICommand* mLastCommand;
35    TSlots mSlots;
36 };
37
38 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : RemoteControl.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 11.01.2012
// Description : Implemantation of class RemoteControl
////////////////////////////////////////////////////////////////////////

#include <string>
#include "RemoteControl.h"
#include "NoCommand.h"

RemoteControl::RemoteControl()
   : mLastCommand(0)
{
   try
   {
      for(int i=0; i<MaxSlots; ++i)
      {
         Slot* DefaultSlot = new Slot;
         ICommand* DefaultOnCommand = new NoCommand;
         ICommand* DefaultOffCommand = new NoCommand;

         DefaultSlot->SetCommands(DefaultOnCommand,DefaultOffCommand);
         mSlots.push_back(DefaultSlot);
      }
   }
   catch(std::bad_alloc& ex)
   {
      throw(ex);
   }
}

void RemoteControl::OnButtonPressed(size_t SlotNumber)
{
   try
   {
      if (SlotNumber == 0 || SlotNumber > MaxSlots)
      {
         throw std::string("RemoteControl::OnButtonPressed: SlotNumber out
            of range");
      }

      mSlots[SlotNumber-1]->ClickedOn();
      mLastCommand = mSlots[SlotNumber-1]->GetOnCommand();
   }
   catch(std::string const& ex)
   {
      throw(ex);
   }
}

void RemoteControl::OffButtonPressed(size_t SlotNumber)
{
   try
   {
      if (SlotNumber == 0 || SlotNumber > MaxSlots)
      {
         throw std::string("RemoteControl::OffButtonPressed: SlotNumber out
             of range");
      }
```

```
59
60          mSlots[SlotNumber-1]->ClickedOff();
61          mLastCommand = mSlots[SlotNumber-1]->GetOffCommand();
62       }
63       catch(std::string const& ex)
64       {
65          throw(ex);
66       }
67    }
68
69    void RemoteControl::ProgramSlot(size_t SlotNumber, ICommand* OnCommand,
          ICommand* OffCommand)
70    {
71       try
72       {
73          (SlotNumber == 0 || SlotNumber > MaxSlots) ? throw std::string("
              RemoteControl::ProgramSlot: SlotNumber out of range")
74             : OnCommand == 0 ? throw std::string("RemoteControl::ProgramSlot:
                 OnCommand is a null pointer")
75             : OffCommand == 0 ? throw std::string("RemoteControl::ProgramSlot:
                  OffCommand is a null pointer")
76             : mSlots[SlotNumber-1]->SetCommands(OnCommand,OffCommand);
77       }
78       catch(std::string const& ex)
79       {
80          throw(ex);
81       }
82    }
83
84    void RemoteControl::UndoButtonPressed()
85    {
86       if (mLastCommand == 0)
87       {
88          throw std::string("RemoteControl::UndoButtonPressed: No command to
              undo");
89       }
90
91       mLastCommand->Undo();
92       mLastCommand = 0;
93    }
94
95    RemoteControl::~RemoteControl()
96    {
97       for(int i=0; i<MaxSlots; ++i)
98       {
99          delete mSlots[i];
100      }
101   }
```

```
1  ////////////////////////////////////////////////////////////////////////////
2  // Workfile : Slot.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 11.01.2012
5  // Description : Header of Slot.cpp
6  ////////////////////////////////////////////////////////////////////////////
7
8  #ifndef SLOT_H
9  #define SLOT_H
10
11 #include "Object.h"
12 #include "ICommand.h"
13
14 class Slot :
15    public Object
16 {
17 public:
18    //Default CTor for baseclass
19    Slot();
20
21    //DTor
22    ~Slot();
23
24    void ClickedOff();
25    void ClickedOn();
26    void SetCommands(ICommand* OnCommand, ICommand* OffCommand);
27
28    ICommand* GetOnCommand() const;
29    ICommand* GetOffCommand() const;
30
31 private:
32    ICommand* mOnCommand;
33    ICommand* mOffCommand;
34 };
35
36 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : Slot.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 11.01.2012
// Description : Implemantation of class Slot
////////////////////////////////////////////////////////////////////////

#include <string>
#include "Slot.h"

Slot::Slot()
   : mOnCommand(0), mOffCommand(0)
{}

void Slot::ClickedOff()
{
   mOffCommand == 0 ? throw std::string("Slot::ClickedOff: mOffCommand is a
        null pointer") : mOffCommand->Execute();
}

void Slot::ClickedOn()
{
   mOnCommand == 0 ? throw std::string("Slot::ClickedOn: mOnCommand is a
       null pointer") : mOnCommand->Execute();
}

void Slot::SetCommands(ICommand* OnCommand, ICommand* OffCommand)
{
   delete mOnCommand; mOnCommand = OnCommand;
   delete mOffCommand; mOffCommand = OffCommand;
}

ICommand* Slot::GetOnCommand() const
{
   return mOnCommand;
}

ICommand* Slot::GetOffCommand() const
{
   return mOffCommand;
}

Slot::~Slot()
{
   delete mOnCommand;
   delete mOffCommand;
}
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : ICommand.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 11.01.2012
5  // Description : Interface for Commands
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef COMMAND_H
9  #define COMMAND_H
10
11 class ICommand
12 {
13 public:
14    virtual ~ICommand(){};
15    virtual void Execute() = 0;
16    virtual void Undo() = 0;
17 };
18
19 #endif
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : NoCommand.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 11.01.2012
5  // Description : Header of NoCommand.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef NOCOMMAND_H
9  #define NOCOMMAND_H
10
11 #include "Object.h"
12 #include "ICommand.h"
13
14 class NoCommand :
15    public Object,
16    public ICommand
17 {
18 public:
19    void Execute();
20    void Undo();
21 };
22
23 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : NoCommand.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 11.01.2012
// Description : Implementation of class NoCommand
////////////////////////////////////////////////////////////////////////

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

void NoCommand::Execute()
{
   std::cout << "Can not execute: No Command" << std::endl;
}

void NoCommand::Undo()
{
   std::cout << "Can not undo: No Command" << std::endl;
}
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : OffCommand.h
// Author : Reinhard Penn, Bernhard Selymes
// Date : 11.01.2012
// Description : Header of OffCommand.cpp
////////////////////////////////////////////////////////////////////////

#ifndef OFFCOMMAND_H
#define OFFCOMMAND_H

#include "Object.h"
#include "ICommand.h"
#include "BaseDevice.h"

class OffCommand :
   public Object,
   public ICommand
{
public:
   OffCommand(BaseDevice* device);
   void Execute();
   void Undo();

private:
   BaseDevice* mDevice;
};

#endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : OffCommand.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 11.01.2012
// Description : Implementation of class OffCommand
////////////////////////////////////////////////////////////////////////

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

OffCommand::OffCommand(BaseDevice* device)
{
   if(device == 0)
   {
      std::string error = "Error in OffCommand::OffCommand: no valid
         pointer";
      throw (error);
   }
   mDevice = device;
}

void OffCommand::Execute()
{
   mDevice->TurnOff();
}

void OffCommand::Undo()
{
   mDevice->UndoAction();
}
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : OnCommand.h
// Author : Reinhard Penn, Bernhard Selymes
// Date : 11.01.2012
// Description : Header of OnCommand.cpp
////////////////////////////////////////////////////////////////////////

#ifndef ONCOMMAND_H
#define ONCOMMAND_H

#include "Object.h"
#include "ICommand.h"
#include "BaseDevice.h"

class OnCommand :
   public Object,
   public ICommand
{
public:
   OnCommand(BaseDevice* device);
   void Execute();
   void Undo();

private:
   BaseDevice* mDevice;
};

#endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : OnCommand.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 11.01.2012
// Description : Implementation of class OnCommand
////////////////////////////////////////////////////////////////////////

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

OnCommand::OnCommand(BaseDevice* device)
{
   if(device == 0)
   {
      std::string error = "Error in OnCommand::OnCommand: no valid pointer"
         ;
      throw (error);
   }
   mDevice = device;
}

void OnCommand::Execute()
{
   mDevice->TurnOn();
}

void OnCommand::Undo()
{
   mDevice->UndoAction();
}
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : CloseCommand.h
// Author : Reinhard Penn, Bernhard Selymes
// Date : 11.01.2012
// Description : Header of CloseCommand.cpp
////////////////////////////////////////////////////////////////////////

#ifndef CLOSECOMMAND_H
#define CLOSECOMMAND_H

#include "Object.h"
#include "ICommand.h"
#include "Stereo.h"

class CloseCommand :
   public Object,
   public ICommand
{
public:
   CloseCommand(Stereo* stereo);
   void Execute();
   void Undo();

private:
   Stereo* mStereo;
};

#endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : CloseCommand.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 11.01.2012
// Description : Implementation of class CloseCommand
////////////////////////////////////////////////////////////////////////

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

CloseCommand::CloseCommand(Stereo* stereo)
{
   if(stereo == 0)
   {
      std::string error = "Error in CloseCommand::CloseCommand: no valid
         pointer";
      throw (error);
   }
   mStereo = stereo;
}

void CloseCommand::Execute()
{
   mStereo->CloseCD();
}

void CloseCommand::Undo()
{
   mStereo->UndoCD();
}
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : OpenCommand.h
// Author : Reinhard Penn, Bernhard Selymes
// Date : 11.01.2012
// Description : Header of OpenCommand.cpp
////////////////////////////////////////////////////////////////////////

#ifndef OPENCOMMAND_H
#define OPENCOMMAND_H

#include "Object.h"
#include "ICommand.h"
#include "Stereo.h"

class OpenCommand :
   public Object,
   public ICommand
{
public:
   OpenCommand(Stereo* stereo);
   void Execute();
   void Undo();

private:
   Stereo* mStereo;
};

#endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : OpenCommand.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 11.01.2012
// Description : Implementation of class OpenCommand
////////////////////////////////////////////////////////////////////////

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

OpenCommand::OpenCommand(Stereo* stereo)
{
   if(stereo == 0)
   {
      std::string error = "Error in OpenCommand::OpenCommand: no valid
         pointer";
      throw (error);
   }
   mStereo = stereo;
}

void OpenCommand::Execute()
{
   mStereo->OpenCD();
}

void OpenCommand::Undo()
{
   mStereo->UndoCD();
}
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : MacroCommand.h
// Author : Reinhard Penn, Bernhard Selymes
// Date : 11.01.2012
// Description : Header of MacroCommand.cpp
////////////////////////////////////////////////////////////////////////

#ifndef MACROCOMMAND_H
#define MACROCOMMAND_H

#include <list>
#include "Object.h"
#include "ICommand.h"

typedef std::list<ICommand*> TCommands;

class MacroCommand :
   public Object,
   public ICommand
{
public:
   void Execute();
   void Undo();
   void Add(ICommand* command);
   void Remove(ICommand* command);
private:
   TCommands mCommands;
};

```

```
30 #endif

1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : MacroCommand.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 11.01.2012
5  // Description : Implementation of class MacroCommand
6  ////////////////////////////////////////////////////////////////////////
7
8  #include <string>
9  #include <iostream>
10 #include <algorithm>
11 #include "MacroCommand.h"
12
13 void MacroCommand::Execute()
14 {
15    std::for_each(mCommands.begin(), mCommands.end(), [&](ICommand* command)
16    {
17       command->Execute();
18    });
19 }
20
21 void MacroCommand::Undo()
22 {
23    std::for_each(mCommands.begin(), mCommands.end(), [&](ICommand* command)
24    {
25       command->Undo();
26    });
27 }
28
29 void MacroCommand::Add(ICommand* command)
30 {
31    if(command == 0)
32    {
33       std::string error = "Error in MacroCommand::Add: no valid pointer";
34       throw (error);
35    }
36    if(mCommands.size() >= 2)
37    {
38       std::string error = "Error in MacroCommand::Add: only two entries are
             allowed";
39       throw (error);
40    }
41    mCommands.push_back(command);
42 }
43
44 void MacroCommand::Remove(ICommand* command)
45 {
46    if(command == 0)
47    {
48       std::string error = "Error in MacroCommand::Remove: no valid pointer"
          ;
49       throw (error);
50    }
51    mCommands.remove(command);
52 }
```

```cpp
////////////////////////////////////////////////////////////////////////////
// Workfile : IDevice.h
// Author : Reinhard Penn, Bernhard Selymes
// Date : 11.01.2012
// Description : Interface vor Devices
////////////////////////////////////////////////////////////////////////////

#ifndef IDEVICE_H
#define IDEVICE_H

#include <fstream>
#include <string>

class IDevice
{
public:
   virtual ~IDevice(){};
   virtual void Info(std::ostream& stream) = 0;
   virtual std::string GetName() const = 0;
};

#endif
```

```cpp
////////////////////////////////////////////////////////////////////////////
// Workfile : BaseDevice.h
// Author : Reinhard Penn, Bernhard Selymes
// Date : 11.01.2012
// Description : Header of BaseDevice.cpp
////////////////////////////////////////////////////////////////////////////

#ifndef BASEDEVICE_H
#define BASEDEVICE_H

#include "Object.h"
#include "IDevice.h"
#include "OnOffState.h"

class BaseDevice :
   public Object,
   public IDevice
{
public:
   BaseDevice();
   void Info(std::ostream& stream) = 0;
   void TurnOff();
   void TurnOn();
   void UndoAction();
   std::string GetName() const = 0;
protected:
   OnOffState mState;

private:
   OnOffState mLastState;
};

#endif
```

```cpp
////////////////////////////////////////////////////////////////////////////
// Workfile : BaseDevice.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 11.01.2012
// Description : Implementation of class BaseDevice
////////////////////////////////////////////////////////////////////////////

#include "BaseDevice.h"

BaseDevice::BaseDevice()
   : mState(eOff), mLastState(eOff)
{}

void BaseDevice::TurnOff()
{
   mLastState = mState;
   mState = eOff;
}

void BaseDevice::TurnOn()
{
   mLastState = mState;
   mState = eOn;
}

void BaseDevice::UndoAction()
{
   mState = mLastState;
}
```

```cpp
////////////////////////////////////////////////////////////////////////////
// Workfile : Heating.h
// Author : Reinhard Penn, Bernhard Selymes
// Date : 11.01.2012
// Description : Header of Heating.cpp
////////////////////////////////////////////////////////////////////////////

#ifndef HEATING_H
#define HEATING_H

#include "BaseDevice.h"

class Heating :
   public BaseDevice
{
public:
   void Info(std::ostream& stream);
   std::string GetName() const;
};

#endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : Heating.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 11.01.2012
// Description : Implementation of class Heating
////////////////////////////////////////////////////////////////////////

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

void Heating::Info(std::ostream& stream)
{
   if(stream == 0)
   {
      std::string error = "Error in Heating::Info: no valid stream";
      throw (error);
   }
   if(mState == eOn)
   {
      stream << "Heating is On" << std::endl;
   }
   else
   {
      stream << "Heating is Off" << std::endl;
   }
}

std::string Heating::GetName() const
{
   return "Heating";
}
```

```
1  ////////////////////////////////////////////////////////////////////////////
2  // Workfile : TV.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 11.01.2012
5  // Description : Header of TV.cpp
6  ////////////////////////////////////////////////////////////////////////////
7
8  #ifndef TV_H
9  #define TV_H
10
11 #include "BaseDevice.h"
12
13 class TV :
14    public BaseDevice
15 {
16 public:
17    void Info(std::ostream& stream);
18    std::string GetName() const;
19 };
20
21 #endif
```

```
1  ////////////////////////////////////////////////////////////////////////////
2  // Workfile : TV.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 11.01.2012
5  // Description : Implementaion of class TV
6  ////////////////////////////////////////////////////////////////////////////
7
8  #include <iostream>
9  #include <string>
10 #include "TV.h"
11
12 void TV::Info(std::ostream& stream)
13 {
14    if(stream == 0)
15    {
16       std::string error = "Error in TV::Info: no valid stream";
17       throw (error);
18    }
19    if(mState == eOn)
20    {
21       stream << "TV is On" << std::endl;
22    }
23    else
24    {
25       stream << "TV is Off" << std::endl;
26    }
27 }
28
29 std::string TV::GetName() const
30 {
31    return "TV";
32 }
```

```
1  ////////////////////////////////////////////////////////////////////////////
2  // Workfile : Stereo.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 11.01.2012
5  // Description : Header of Stereo.cpp
6  ////////////////////////////////////////////////////////////////////////////
7
8  #ifndef STEREO_H
9  #define STEREO_H
10
11 #include "BaseDevice.h"
12 #include "CDState.h"
13
14 class Stereo :
15    public BaseDevice
16 {
17 public:
18    void Info(std::ostream& stream);
19    void OpenCD();
20    void CloseCD();
21    void UndoCD();
22    std::string GetName() const;
23 private:
24    CDState mDriveState;
25    CDState mLastDriveState;
26 };
27
28 #endif
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : Stereo.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 11.01.2012
5  // Description : Implementaion of class Stereo
6  ////////////////////////////////////////////////////////////////////////
7
8  #include <iostream>
9  #include <string>
10 #include "Stereo.h"
11
12 void Stereo::Info(std::ostream& stream)
13 {
14    if(stream == 0)
15    {
16       std::string error = "Error in Stereo::Info: no valid stream";
17       throw (error);
18    }
19    if(mState == eOn)
20    {
21       if(mDriveState == eOpened)
22       {
23          stream << "Stereo is On" << " " << "CD is opened" <<  std::endl;
24       }
25       else
26       {
27          stream << "Stereo is On" << " " << "CD is closed" <<  std::endl;
28       }
29    }
30    else
31    {
32       if(mDriveState == eOpened)
33       {
34          stream << "Stereo is Off" << " " << "CD is opened" <<  std::endl;
35       }
36       else
37       {
38          stream << "Stereo is Off" << " " << "CD is closed" <<  std::endl;
39       }
40    }
41 }
42
43 void Stereo::OpenCD()
44 {
45    mLastDriveState = mDriveState;
46    mDriveState = eOpened;
47 }
48
49 void Stereo::CloseCD()
50 {
51    mLastDriveState = mDriveState;
52    mDriveState = eClosed;
53 }
54
55 void Stereo::UndoCD()
56 {
57    mDriveState = mLastDriveState;
58 }
59
60 std::string Stereo::GetName() const
```

```
61 {
62    return "Stereo";
63 }
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : OnOffState.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 11.01.2012
5  // Description : Definition of enumeration OnOffState
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef ONOFFSTATE_H
9  #define ONOFFSTATE_H
10
11 enum OnOffState
12 {
13    eOn,
14    eOff
15 };
16
17 #endif
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : CDState.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 11.01.2012
5  // Description : Definition of enumeration CDState
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef CDSTATE_H
9  #define CDSTATE_H
10
11 enum CDState
12 {
13    eOpened,
14    eClosed
15 };
16
17 #endif
```

```cpp
#include <iostream>

#include "ICommand.h"
#include "NoCommand.h"
#include "MacroCommand.h"
#include "OnCommand.h"
#include "OffCommand.h"
#include "OpenCommand.h"
#include "CloseCommand.h"

#include "IDevice.h"
#include "BaseDevice.h"
#include "Heating.h"
#include "TV.h"
#include "Stereo.h"

#include "Client.h"

#include <vld.h>

using namespace std;


void EmptyTestcase()
{
   try
   {
      cout << "Testcase0: Empty testcase with NULL pointer." << endl;

      Client c;

      cout << "Print Interface:" << endl;
      c.PrintInterface();

      cout << "Add Device:" << endl;
      c.AddDevice(0,0,0,0);
   }
   catch(std::bad_alloc& ex)
   {
      cout << ex.what() << endl;
   }
   catch(std::string const& ex)
   {
      cout << ex << endl;
   }
   catch(...)
   {
      cout << "Unhandled exception occured";
   }
}

void EmptyProcessInfo()
{
   try
   {
      Client c;

      cout << "Print Device Info:" << endl;
      c.PrintDeviceInfo(cout);
   }
```

```
61     catch(std::bad_alloc& ex)
62     {
63        cout << ex.what() << endl;
64     }
65     catch(std::string const& ex)
66     {
67        cout << ex << endl;
68     }
69     catch(...)
70     {
71        cout << "Unhandled exception occured";
72     }
73  }
74
75  void EmptyProcess()
76  {
77     try
78     {
79        Client c;
80        string s(" ");
81
82        cout << "Process";
83        c.Process(s);
84     }
85     catch(std::bad_alloc& ex)
86     {
87        cout << ex.what() << endl;
88     }
89     catch(std::string const& ex)
90     {
91        cout << ex << endl;
92     }
93     catch(...)
94     {
95        cout << "Unhandled exception occured";
96     }
97  }
98
99  void NormalTestcase()
100 {
101    try
102    {
103       cout << "Testcase1: Normal testcase." << endl;
104
105       Client c;
106       string Input("i");
107
108       cout << "Creating Devices:" << endl;
109       Stereo stereo;
110       Heating heater;
111       TV tv;
112
113       cout << "Creating Commands:" << endl;
114       ICommand* onCommandStereo = new OnCommand(&stereo);
115       ICommand* openCommandStereo = new OpenCommand(&stereo);
116       ICommand* offCommandStereo = new OffCommand(&stereo);
117       ICommand* closeCommandStereo = new CloseCommand(&stereo);
118
119       MacroCommand* macroCommandStereoOnOpen = new MacroCommand;
120       macroCommandStereoOnOpen->Add(onCommandStereo);
```

```
121         macroCommandStereoOnOpen->Add(openCommandStereo);
122
123         MacroCommand* macroCommandStereoOffClose = new MacroCommand;
124         macroCommandStereoOffClose->Add(offCommandStereo);
125         macroCommandStereoOffClose->Add(closeCommandStereo);
126
127         ICommand* onCommandHeater = new OnCommand(&heater);
128         ICommand* ofCommandHeater = new OffCommand(&heater);
129
130         ICommand* onCommandTV = new OnCommand(&tv);
131         ICommand* ofCommandTV = new OffCommand(&tv);
132
133         cout << "Adding devices to client:" << endl;
134         c.AddDevice(&stereo,4,macroCommandStereoOnOpen,
               macroCommandStereoOffClose);
135         c.AddDevice(&heater,2,onCommandHeater,ofCommandHeater);
136         c.AddDevice(&tv,1,onCommandTV,ofCommandTV);
137
138         cout << "Print Interface:" << endl;
139         c.PrintInterface();
140
141         cout << "Process:" << endl;
142         cout << Input << endl;
143         c.Process(Input);
144
145         Input = "1o";
146         cout << Input << endl;
147         c.Process(Input);
148
149         Input = "2o";
150         cout << Input << endl;
151         c.Process(Input);
152
153         Input = "6o";
154         cout << Input << endl;
155         c.Process(Input);
156
157         Input = "4o";
158         cout << Input << endl;
159         c.Process(Input);
160
161         Input = "i";
162         cout << Input << endl;
163         c.Process(Input);
164
165         Input = "u";
166         cout << Input << endl;
167         c.Process(Input);
168
169         Input = "i";
170         cout << Input << endl;
171         c.Process(Input);
172
173         cout << endl << endl;
174
175         delete onCommandStereo;
176         delete offCommandStereo;
177         delete openCommandStereo;
178         delete closeCommandStereo;
179     }
```

```
180     catch(std::bad_alloc& ex)
181     {
182        cout << ex.what() << endl;
183     }
184     catch(std::string const& ex)
185     {
186        cout << ex << endl;
187     }
188     catch(...)
189     {
190        cout << "Unhandled exception occured";
191     }
192  }
193
194  int main()
195  {
196     EmptyTestcase();
197     EmptyProcessInfo();
198     EmptyProcess();
199     cout << endl << endl;
200
201     NormalTestcase();
202
203     return 0;
204  }
```

6 Testausgaben

```
Visual Leak Detector Version 2.2.3 installed.
Testcase0: Empty testcase with NULL pointer.
Print Interface:
Remote control:
----------------------
1...empty
2...empty
3...empty
4...empty
5...empty
6...empty
u...undo
i...output device info
----------------------
input slot number and on('o') or off('f'):
Add Device:
Client::AddDevice: Device is a null pointer
Print Device Info:
Devices:
----------------------
ProcessClient::Process: Input is not valid

Testcase1: Normal testcase.
Creating Devices:
Creating Commands:
Adding devices to client:
Print Interface:
Remote control:
----------------------
1...empty
2...TV
3...Heating
4...empty
5...Stereo
6...empty
u...undo
i...output device info
----------------------
input slot number and on('o') or off('f'):
Process:
i
Devices:
----------------------
TV is Off
Heating is Off
Stereo is Off CD is closed
1o
2o
```

```
6o
Can not execute: No Command
4o
i
Devices:
-----------------------
TV is On
Heating is On
Stereo is On CD is opened
u
i
Devices:
-----------------------
TV is On
Heating is On
Stereo is Off CD is closed

No memory leaks detected.
```