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

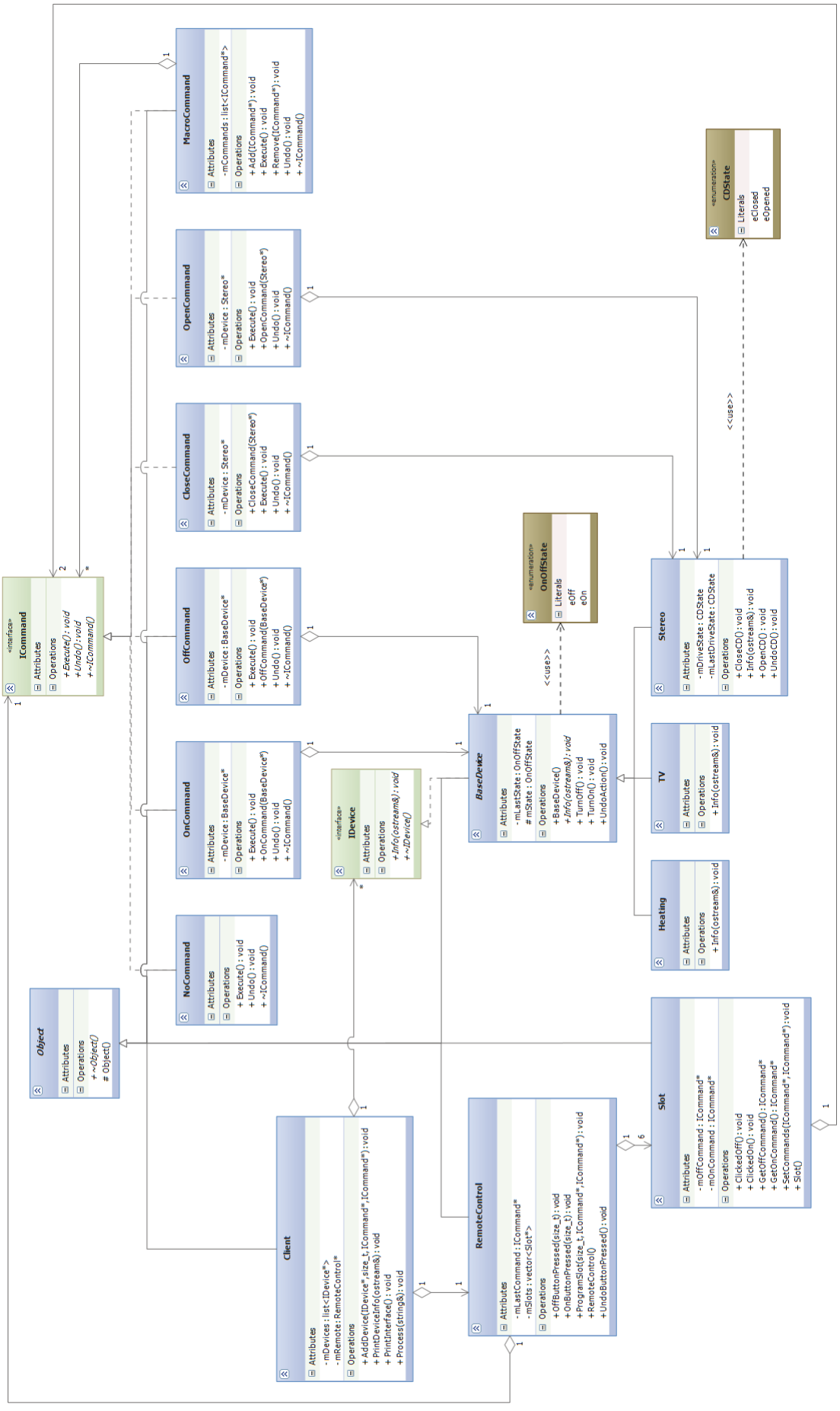
- geschätzte Mh: 15
- tatsächlich: Reinhard (8h), Bernhard (8h)

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.

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.

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

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.

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.

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 <list>
12 #include <iostream>
13 #include <string>
14 #include "Object.h"
15 #include "RemoteControl.h"
16 #include "IDevice.h"
17
18 typedef std::list<IDevice*> TDevices;
19
20 std::string const DivLine("-----------------------");
21
22 class Client :
23    public Object
24 {
25 public:
26    void AddDevice(IDevice* Device, size_t SlotNumber, ICommand* OnCommand,
       ICommand* OffCommand);
27    void PrintDeviceInfo(std::ostream& stream);
28    void PrintInterface();
29    void Process(std::string& Input, std::ostream& stream = std::cout);
30 private:
31    TDevices mDevices;
32    RemoteControl mRemote;
33 };
34
35 #endif
```

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

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

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.push_back(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;

      std::for_each(mDevices.begin(),mDevices.end(),[&](IDevice* d)
      {
         d->Info(stream);
      });
   }
```

```
58     catch(std::string const& ex)
59     {
60        throw(ex);
61     }
62  }
63
64  void Client::PrintInterface()
65  {
66     std::cout << "Remote control:" << std::endl;
67     std::cout << DivLine << std::endl;
68     std::cout << "1...TV" << std::endl;
69     std::cout << "2...Heating" << std::endl;
70     std::cout << "3...empty" << std::endl;
71     std::cout << "4...Stereo" << std::endl;
72     std::cout << "5...empty" << std::endl;
73     std::cout << "6...empty" << std::endl;
74     std::cout << "u...undo" << std::endl;
75     std::cout << "i...output device info" << std::endl;
76     std::cout << DivLine << std::endl;
77     std::cout << "input slot number and on('o') or off('f'):" << std::endl;
78  }
79
80  void Client::Process(std::string& Input, std::ostream& stream)
81  {
82     try
83     {
84        if (Input.empty())
85        {
86           throw std::string("Client::Process: Input is null");
87        }
88        if (Input[0] == 'u')
89        {
90           mRemote.UndoButtonPressed();
91        }
92        else if (Input[0] == 'i')
93        {
94           PrintDeviceInfo(stream);
95        }
96        else if (Input.size() >= 2)
97        {
98           bool InputValid = false;
99
100          for (int i=0; i<MaxSlots; ++i)
101          {
102             if (Input[0] == (char)(i+'1'))
103             {
104                if (Input[1] == 'o')
105                {
106                   InputValid = true;
107                   mRemote.OnButtonPressed(i+1);
108                }
109                else if (Input[1] == 'f')
110                {
111                   InputValid = true;
112                   mRemote.OffButtonPressed(i+1);
113                }
114                else
115                {
116                   throw std::string("Client::Process: Input is not valid");
117                }
```

```
118             }
119          }
120          if (!InputValid)
121          {
122             throw std::string("Client::Process: Input is not valid");
123          }
124       }
125       else
126       {
127          throw std::string("Client::Process: Input is not valid");
128       }
129    }
130    catch(std::string const& ex)
131    {
132       throw(ex);
133    }
134 }
```

```
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;
}
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : OffCommand.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 11.01.2012
5  // Description : Header of OffCommand.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef OFFCOMMAND_H
9  #define OFFCOMMAND_H
10
11 #include "Object.h"
12 #include "ICommand.h"
13 #include "BaseDevice.h"
14
15 class OffCommand :
16    public Object,
17    public ICommand
18 {
19 public:
20    OffCommand(BaseDevice* device);
21    void Execute();
22    void Undo();
23
24 private:
25    BaseDevice* mDevice;
26 };
27
28 #endif
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : OffCommand.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 11.01.2012
5  // Description : Implementation of class OffCommand
6  ////////////////////////////////////////////////////////////////////////
7
8  #include <iostream>
9  #include <string>
10 #include "OffCommand.h"
11
12 OffCommand::OffCommand(BaseDevice* device)
13 {
14    if(device == 0)
15    {
16       std::string error = "Error in OffCommand::OffCommand: no valid
             pointer";
17       throw (error);
18    }
19    mDevice = device;
20 }
21
22 void OffCommand::Execute()
23 {
24    mDevice->TurnOff();
25 }
26
27 void OffCommand::Undo()
28 {
29    mDevice->UndoAction();
30 }
```

```
1  ////////////////////////////////////////////////////////////////////////////
2  // Workfile : OnCommand.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 11.01.2012
5  // Description : Header of OnCommand.cpp
6  ////////////////////////////////////////////////////////////////////////////
7
8  #ifndef ONCOMMAND_H
9  #define ONCOMMAND_H
10
11 #include "Object.h"
12 #include "ICommand.h"
13 #include "BaseDevice.h"
14
15 class OnCommand :
16    public Object,
17    public ICommand
18 {
19 public:
20    OnCommand(BaseDevice* device);
21    void Execute();
22    void Undo();
23
24 private:
25    BaseDevice* mDevice;
26 };
27
28 #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();
}
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : CloseCommand.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 11.01.2012
5  // Description : Header of CloseCommand.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef CLOSECOMMAND_H
9  #define CLOSECOMMAND_H
10
11 #include "Object.h"
12 #include "ICommand.h"
13 #include "Stereo.h"
14
15 class CloseCommand :
16    public Object,
17    public ICommand
18 {
19 public:
20    CloseCommand(Stereo* stereo);
21    void Execute();
22    void Undo();
23
24 private:
25    Stereo* mStereo;
26 };
27
28 #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();
}
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : OpenCommand.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 11.01.2012
5  // Description : Header of OpenCommand.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef OPENCOMMAND_H
9  #define OPENCOMMAND_H
10
11 #include "Object.h"
12 #include "ICommand.h"
13 #include "Stereo.h"
14
15 class OpenCommand :
16    public Object,
17    public ICommand
18 {
19 public:
20    OpenCommand(Stereo* stereo);
21    void Execute();
22    void Undo();
23
24 private:
25    Stereo* mStereo;
26 };
27
28 #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();
}
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : MacroCommand.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 11.01.2012
5  // Description : Header of MacroCommand.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef MACROCOMMAND_H
9  #define MACROCOMMAND_H
10
11 #include <list>
12 #include "Object.h"
13 #include "ICommand.h"
14
15 typedef std::list<ICommand*> TCommands;
16
17 class MacroCommand :
18    public Object,
19    public ICommand
20 {
21 public:
22    void Execute();
23    void Undo();
24    void Add(ICommand* command);
25    void Remove(ICommand* command);
26 private:
27    TCommands mCommands;
28 };
29
30 #endif
```

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

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

void MacroCommand::Execute()
{
   std::for_each(mCommands.begin(), mCommands.end(), [&](ICommand* command)
   {
      command->Execute();
   });
}

void MacroCommand::Undo()
{
   std::for_each(mCommands.begin(), mCommands.end(), [&](ICommand* command)
   {
      command->Undo();
   });
}

void MacroCommand::Add(ICommand* command)
{
   if(command == 0)
   {
      std::string error = "Error in MacroCommand::Add: no valid pointer";
      throw (error);
   }
   if(mCommands.size() >= 2)
   {
      std::string error = "Error in MacroCommand::Add: only two entries are
          allowed";
      throw (error);
   }
   mCommands.push_back(command);
}

void MacroCommand::Remove(ICommand* command)
{
   if(command == 0)
   {
      std::string error = "Error in MacroCommand::Remove: no valid pointer"
         ;
      throw (error);
   }
   mCommands.remove(command);
}
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : IDevice.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 11.01.2012
5  // Description : Interface vor Devices
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef IDEVICE_H
9  #define IDEVICE_H
10
11 #include <fstream>
12
13 class IDevice
14 {
15 public:
16    virtual ~IDevice(){};
17    virtual void Info(std::ostream& stream) = 0;
18 };
19
20 #endif
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : BaseDevice.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 11.01.2012
5  // Description : Header of BaseDevice.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef BASEDEVICE_H
9  #define BASEDEVICE_H
10
11 #include "Object.h"
12 #include "IDevice.h"
13 #include "OnOffState.h"
14
15 class BaseDevice :
16    public Object,
17    public IDevice
18 {
19 public:
20    BaseDevice() : mState(eOff), mLastState(eOff) {}
21    void Info(std::ostream& stream) = 0;
22    void TurnOff();
23    void TurnOn();
24    void UndoAction();
25
26 protected:
27    OnOffState mState;
28
29 private:
30    OnOffState mLastState;
31 };
32
33 #endif
```

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

#include "BaseDevice.h"

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

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

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

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

```
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 };
19
20 #endif
```

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

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

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

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

#ifndef STEREO_H
#define STEREO_H

#include "BaseDevice.h"
#include "CDState.h"

class Stereo :
   public BaseDevice
{
public:
   void Info(std::ostream& stream);
   void OpenCD();
   void CloseCD();
   void UndoCD();

private:
   CDState mDriveState;
   CDState mLastDriveState;
};

#endif
```

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

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

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

void Stereo::OpenCD()
{
   mLastDriveState = mDriveState;
   mDriveState = eOpened;
}

void Stereo::CloseCD()
{
   mLastDriveState = mDriveState;
   mDriveState = eClosed;
}

void Stereo::UndoCD()
{
   mDriveState = mLastDriveState;
}
```

```
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
```

```
1  #include <iostream>
2
3  #include "ICommand.h"
4  #include "NoCommand.h"
5  #include "MacroCommand.h"
6  #include "OnCommand.h"
7  #include "OffCommand.h"
8  #include "OpenCommand.h"
9  #include "CloseCommand.h"
10
11 #include "IDevice.h"
12 #include "BaseDevice.h"
13 #include "Heating.h"
14 #include "TV.h"
15 #include "Stereo.h"
16
17 #include "Client.h"
18
19 using namespace std;
20
21
22 void EmptyTestcase()
23 {
24    try
25    {
26       cout << "Testcase0: Empty testcase with NULL pointer." << endl;
27
28       Client c;
29
30       cout << "Print Interface:" << endl;
31       c.PrintInterface();
32
33       cout << "Add Device:" << endl;
34       c.AddDevice(0,0,0,0);
35    }
36    catch(std::bad_alloc& ex)
37    {
38       cout << ex.what() << endl;
39    }
40    catch(std::string const& ex)
41    {
42       cout << ex << endl;
43    }
44    catch(...)
45    {
46       cout << "Unhandled exception occured";
47    }
48 }
49
50 void EmptyProcessInfo()
51 {
52    try
53    {
54       Client c;
55
56       cout << "Print Device Info:" << endl;
57       c.PrintDeviceInfo(cout);
58    }
59    catch(std::bad_alloc& ex)
60    {
```

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

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

```
180     }
181     catch(...)
182     {
183        cout << "Unhandled exception occured";
184     }
185 }
186 
187 int main()
188 {
189     EmptyTestcase();
190     EmptyProcessInfo();
191     EmptyProcess();
192     cout << endl << endl;
193 
194     NormalTestcase();
195 
196     return 0;
197 }
```

6 Testausgaben