1 Organisatorisches

1.1 Team

- Reinhard Penn, s1110306019
- Bernhard Selymes, s1110306024

1.2 Aufteilung

- Reinhard Penn
  - Planung
  - Klassendiagramm
  - Implementierung der Klassen MusicFactory, MusicComponent, Song, Album, MusicCollection
  - Testen aller Klassen
- Bernhard Selymes
  - Planung
  - Klassendiagramm
  - Implementierung der Klassen Visitor, TimeVisitor, SearchVisitor, PlayVisitor, MusicPlayer
  - Dokumentation

1.3 Zeitaufwand

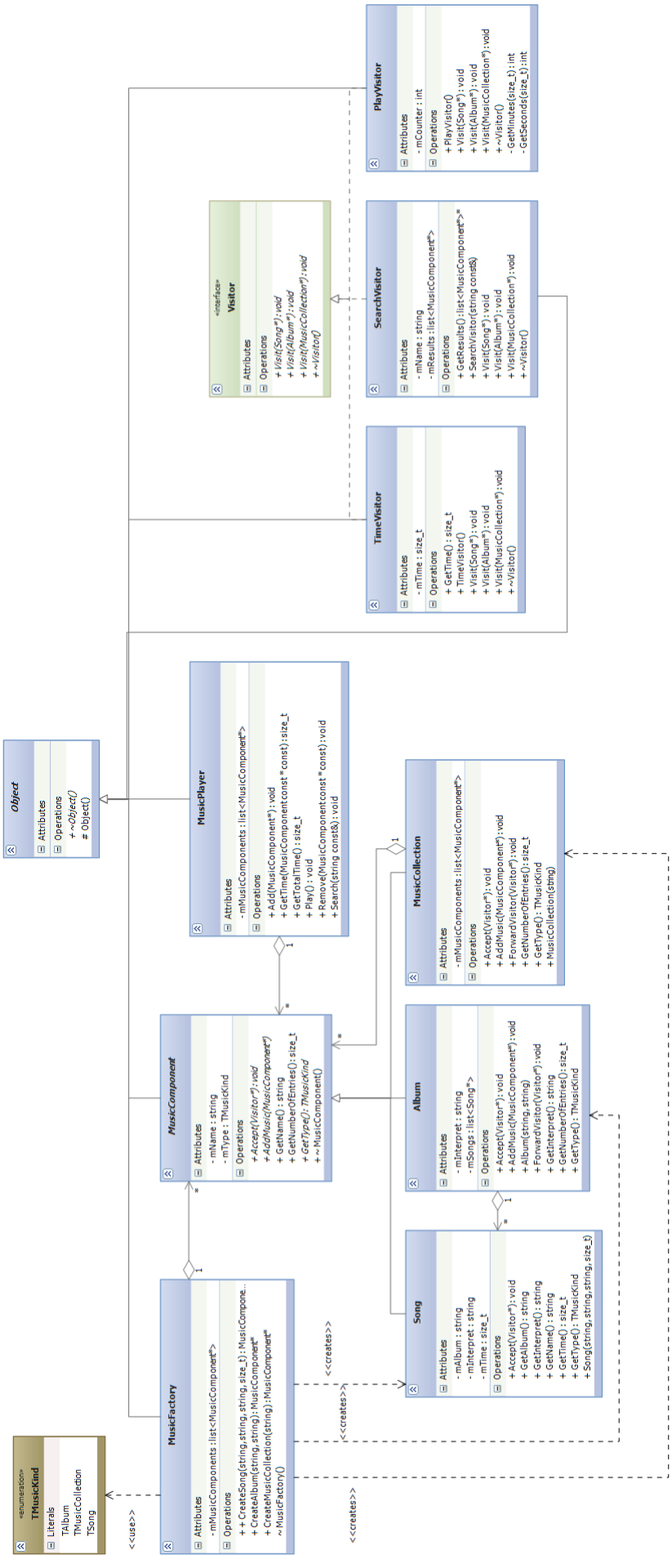
- geschätzte Mh: 15
- tatsächlich: Reinhard (10h), Bernhard (10h)

2 Systemspezifikation

Es soll eine Software für die Verwaltung von Musikmedien entworfen werden. Ein Musikmedium kann ein Lied, ein Album oder eine Musikkollektion sein. Ein Album besteht aus Liedern, eine Kollektion kann aus allen Arten von Musikmedien bestehen. Für alle Medien wirde der Name gespeichert, für das Album zusätzlich der Interpret, für das Lied zusätzlich Interpret, Album Spieldauer in Minuten und Sekunden. Mit dem Musik Player kann man Musikmedien hinzufügen und entfernen, diese wiedergeben (via std::cout), die Spieldauer einzelner oder aller Musikmedien herausfinden und nach Musikmedien suchen.

3 Systementwurf

3.1 Klassendiagramm



3.2 Komponentenübersicht

- Klasse ”Object”:
  Basis aller Basisklassen.
- Enumeration ”MusicKind”:
  Definiert die verschiedenen Arten von Musikmedien.
- Klasse ”MusicFactory”:
  Zuständig für die Erzeugung der Objekte.
- Klasse ”MusicComponent”:
  Basisklasse für Musikmedien.
- Klasse ”Song”:
  Abgeleitet von MusicComponent.
- Klasse ”Album”:
  Abgeleitet von MusicComponent.
- Klasse ”MusicCollection”:
  Abgeleitet von MusicComponent.
- Interface ”Visitor”:
  Abstrakte Basisklasse für Visitors.
- Klasse ”TimeVisitor”:
  Abgeleitet von Visitor. Visitor für Spieldauer.
- Klasse ”SearchVisitor”:
  Abgeleitet von Visitor. Visitor für die Suche nach Medien.
- Klasse ”PlayVisitor”:
  Abgeleitet von Visitor. Visitor für das Abspielen von Medien.
- Klasse ”MusicPlayer”:
  Klasse die die Medien verwaltet und alles mögliche mit ihnen machen kann.

4 Komponentenentwurf

4.1 Klasse ”Object”

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

4.2 Enumeration ”TMusicKind”

- TSong
- TAlbum
- TMusicCollection

4.3 Klasse ”MusicFactory”

Kann Musikmedien dynamisch anlegen und wieder freigeben. In einer Liste werden die Referenzen auf die Objekte gespeichert. Die Methoden legen die Objekte je nach Medientyp mit den übergebenen Parametern an und fügen sie der Liste hinzu. Die Factory darf nur nach dem Musikplayer gelöscht werden, es kann nicht überprüft werden ob die Referenzen auf Objekte noch im Musicplayer gespeichert sind, weil keine Verbindung zwischen den beiden Klassen besteht.

4.4 Klasse ”MusicComponent”

Bietet die Schnittstellen für die Methoden ”Accept”, ”GetType”, ”GetNumberOfEntries” und ”AddMusic”, ”GetName” wird implementiert. Hat zwei Member die Namen und Type speichern.

Methode ”GetName”:
Schnittstelle:
Rückgabetyp: string
Get Funktion.

4.5 Klasse ”Song”

Hat Member die den Namen des Albums und des Interpreten und die Spieldauer in Sekunden speichert. Wir haben uns für Sekunden entschieden, weil die Übergabe leichter ist (keine extra Struktur), das Addieren von mehreren Zeiten umständlich wäre und die Sekunden für die Ausgabe und weitere Verwendung leicht umgerechnet werden können. Hat einige Getter Funktionen.

Methode ”Accept”:
Schnittstelle:
Parameter: Visitor*
Rückgabetyp: void
Ruft die Funktion Visit vom übergebenen Visitor mit sich selbst auf.

4.6 Klasse ”Album”

Hat Member die den Namen des Interpreten und eine Liste die die Lieder des Albums speichert.
Hat einige Getter Funktionen.

Methode ”Accept”:
Schnittstelle:
Parameter: Visitor*
Rückgabetyp: void
Ruft die Funktion Visit vom übergebenen Visitor mit sich selbst auf.

Methode ”ForwardVisitor”:
Schnittstelle:
Parameter: Visitor*
Rückgabetyp: void
Ruft für alle Lieder die Funktion ”Accept” mit dem Visitor auf.

Methode ”GetTime”:
Schnittstelle:
Rückgabetyp: void
Ruft für alle Lieder die Methode ”Accept” mit dem Visitor auf.

Methode ”AddMusic”:
Schnittstelle:
Parameter: MusicComponent*
Rückgabetyp: void
Fügt ein Lied (und nur ein Lied) zur Liste hinzu.

4.7 Klasse ”MusicCollection”

Hat eine Liste die alle Musikmedien, die in der Kollektion enthalten sind.

Methode ”Accept”:
Schnittstelle:
Parameter: Visitor*
Rückgabetyp: void
Ruft die Funktion Visit vom übergebenen Visitor mit sich selbst auf.

Methode ”ForwardVisitor”:
Schnittstelle:
Rückgabetyp: void
Ruft für alle Musikmedien die Methode ”Accept” mit dem Visitor auf.

Methode ”GetNumberOfEntries”:
Schnittstelle:
Rückgabetyp: size_t
Ruft für alle Musikmedien die Funktion ”GetNumberOfEntries” auf und addiert sie.

4.8 Interface ”Visitor”

Definiert die Schnittstellen der Methoden.

Methoden ”Visit”:
Schnittstelle:
Parameter: Song* oder Album* oder MusicCollection*
Rückgabetyp: void
Pure virtual function.

4.9 Klasse ”TimeVisitor”

Hat einen Member der die gesamte Dauer speichert.

Methode ”Visit”:
Schnittstelle:
Parameter: Song*
Rückgabetyp: void
Addiert zum Gesamtdauermember die Dauer vom Lied.

Methode ”Visit”:
Schnittstelle:
Parameter: Album*
Rückgabetyp: void
Ruft die Funktion ”ForwardVisitor” vom Album mit sich selbst (Visitor) auf.

Methode ”Visit”:
Schnittstelle:
Parameter: MusicCollection*
Rückgabetyp: void
Ruft die Funktion ”ForwardVisitor” von der Kollektion mit sich selbst (Visitor) auf.

4.10 Klasse ”SearchVisitor”

Hat einen Member der den gesuchten Namen speichert und eine Liste die Referenzen zu den gefundenen Objekten speichert.

Methode ”Visit”:
Schnittstelle:
Parameter: Song*
Rückgabetyp: void
Schaut ob der gesuchte Name Teil des Namens vom Lied ist und speichert in ggf. in der Liste.

Methode ”Visit”:
Schnittstelle:
Parameter: Album*
Rückgabetyp: void
Ruft die Funktion ”ForwardVisitor” vom Album mit sich selbst (Visitor) auf.

Methode ”Visit”:
Schnittstelle:
Parameter: MusicCollection*
Rückgabetyp: void
Ruft die Funktion ”ForwardVisitor” von der Kollektion mit sich selbst (Visitor) auf.

4.11 Klasse ”PlayVisitor”

Hat einen Zähler der die Nummer der Lieder bestimmt.

Methode ”Visit”:
Schnittstelle:
Parameter: Song*
Rückgabetyp: void
Gibt die Daten des Liedes formatiert auf der Konsole aus und erhöht den Zähler.

Methode ”Visit”:
Schnittstelle:
Parameter: Album*
Rückgabetyp: void
Legt einen TimeVisitor an. Dieser wird vom Album accepted. Die Informationen des Albums werden auf der Konsole ausgegeben. Danach wird der Visitor den Elementen im Album weitergegeben.

Methode ”Visit”:
Schnittstelle:
Parameter: MusicCollection*
Rückgabetyp: void
Legt einen TimeVisitor an. Dieser wird von der Kollektion akkzeptiert. Die Informationen der Kollektion werden auf der Konsole ausgegeben. Danach wird der Visitor den Elementen in der Kollektion weitergegeben.

4.12 Klasse ”MusicPlayer”

Hat einen Liste mit Musik-Medien.

Methode ”GetTime”:
Schnittstelle:
Parameter: MusicComponent* const
Rückgabetyp: size_t
Prüft zuerst ob das Element überhaupt vorhanden ist. Dann wird via eines TimeVisitors die Dauer des Elements ermittelt.

Methode ”GetTotalTime”:
Schnittstelle:
Rückgabetyp: size_t
Via eines TimeVisitors die Dauer aller Elemente ermittelt.

Methode ”Play”:
Schnittstelle:
Rückgabetyp: void
Via eines PlayVisitors wird die gesamte Abspielliste ausgegeben.

Methode ”Search”:
Schnittstelle:
Parameter: string const&
Rückgabetyp: void
Via eines SearchVisitors wird der Name überall gesucht. Der Name der gefundenen Elemente wird danach ausgegeben.

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 : TMusicKind.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 16.12.2012
5  // Description : Enum for Musickind
6  ////////////////////////////////////////////////////////////////////////////
7
8  #ifndef TMUSICKIND_H
9  #define TMUSICKIND_H
10
11 enum TMusicKind
12 {
13    TMusicCollection,
14    TAlbum,
15    TSong
16 };
17
18 #endif
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : MusicFactory.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 16.12.2012
5  // Description : Header for MusicFactory.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef MUSICFACTORY_H
9  #define MUSICFACTORY_H
10
11 #include "Object.h"
12 #include "TMusicKind.h"
13 #include "MusicComponent.h"
14
15 class MusicFactory :
16    public Object
17 {
18 public:
19    //virtual Destructor
20    virtual ~MusicFactory();
21
22    MusicComponent* CreateMusicCollection(std::string Name);
23    MusicComponent* CreateAlbum(std::string Name, std::string Interpret);
24    MusicComponent* CreateSong(std::string Name, std::string Album, std::
          string Interpret, size_t time);
25
26 private:
27    TMusicComponents mMusicComponents;
28 };
29
30 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : MusicFactory.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 16.12.2012
// Description : Implementation of class MusicFactory
////////////////////////////////////////////////////////////////////////

#include <algorithm>
#include "MusicFactory.h"
#include "MusicCollection.h"
#include "Album.h"
#include "Song.h"

MusicFactory::~MusicFactory()
{
   std::for_each(mMusicComponents.begin(),mMusicComponents.end(),[&](
      MusicComponent* m)
   {
      delete m;
   });
}

MusicComponent* MusicFactory::CreateMusicCollection(std::string Name)
{
   MusicComponent* m = new MusicCollection(Name);
   mMusicComponents.push_back(m);
   return m;
}

MusicComponent* MusicFactory::CreateAlbum(std::string Name, std::string
    Interpret)
{
   MusicComponent* m = new Album(Name,Interpret);
   mMusicComponents.push_back(m);
   return m;
}

MusicComponent* MusicFactory::CreateSong(std::string Name, std::string
    Album, std::string Interpret, size_t time)
{
   MusicComponent* m = new Song(Name,Album,Interpret,time);
   mMusicComponents.push_back(m);
   return m;
}
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : MusicComponent.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 16.12.2012
5  // Description : Header for MusicComponent.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef MUSICCOMPONENT_H
9  #define MUSICCOMPONENT_H
10
11 #include <list>
12 #include <string>
13 #include "Object.h"
14 #include "TMusicKind.h"
15
16 class Visitor;
17
18 class MusicComponent :
19    public Object
20 {
21 public:
22    //virtual Destructor for baseclass
23    virtual ~MusicComponent();
24
25    virtual void Accept(Visitor* visitor) = 0;
26
27    virtual TMusicKind GetType() = 0;
28    std::string GetName();
29    virtual size_t GetNumberOfEntries() = 0;
30
31    virtual void AddMusic(MusicComponent* m) = 0;
32 protected:
33    std::string mName;
34    TMusicKind mType;
35 };
36
37 typedef std::list<MusicComponent*> TMusicComponents;
38 typedef TMusicComponents::iterator TMusicComponentsItor;
39
40 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////////
// Workfile : MusicComponent.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 16.12.2012
// Description : Implementation of class MusicComponent
////////////////////////////////////////////////////////////////////////////

#include "MusicComponent.h"

//virtual Destructor for baseclass
MusicComponent::~MusicComponent()
{}

std::string MusicComponent::GetName()
{
   return mName;
}
```

```
1  ////////////////////////////////////////////////////////////////////////////
2  // Workfile : Song.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 16.12.2012
5  // Description : Header for Song.cpp
6  ////////////////////////////////////////////////////////////////////////////
7
8  #ifndef SONG_H
9  #define SONG_H
10
11 #include "MusicComponent.h"
12
13 class Song :
14    public MusicComponent
15 {
16 public:
17    //CTor
18    Song(std::string Name, std::string Album, std::string Interpret, size_t
          time);
19
20    //virtual Destructor
21    virtual ~Song();
22
23    virtual void Accept(Visitor* visitor);
24
25    virtual TMusicKind GetType();
26    size_t GetTime();
27    std::string GetInterpret();
28    std::string GetAlbum();
29    size_t GetNumberOfEntries();
30
31    virtual void AddMusic(MusicComponent* m);
32 private:
33    std::string mAlbum;
34    std::string mInterpret;
35
36    size_t mTime;
37 };
38
39 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : Song.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 16.12.2012
// Description : Implementation of class Song
////////////////////////////////////////////////////////////////////////

#include <iostream>
#include "Song.h"
#include "Visitor.h"

//CTor
Song::Song(std::string Name, std::string Album, std::string Interpret,
    size_t time)
    : mAlbum(Album),mInterpret(Interpret),mTime(time)
{
   mName = Name;
   mType = TSong;
}

//virtual Destructor
Song::~Song()
{}

void Song::Accept(Visitor* visitor)
{
   try
   {
      if(visitor == 0)
      {
         std::string error = "no valid pointer";
         throw (error);
      }
      visitor->Visit(this);
   }
   catch (std::string const& error)
   {
      std::cerr << "Error in Song::Accept: " << error << std::endl;
   }
   catch(...)
   {
      std::cerr << "Song::Accept: Unknown Exception occured" << std::endl;
   }
}

size_t Song::GetTime()
{
   return mTime;
}

std::string Song::GetInterpret()
{
   return mInterpret;
}

void Song::AddMusic(MusicComponent* m)
{
   std::string error = "This functions is not implemented and should not be
       used";
   std::cerr << "Error in Song::AddMusic: " << error << std::endl;
```

```
59 }
60
61 TMusicKind Song::GetType()
62 {
63    return mType;
64 }
65
66 size_t Song::GetNumberOfEntries()
67 {
68    return 1;
69 }
```

```cpp
////////////////////////////////////////////////////////////////////////////
// Workfile : Album.h
// Author : Reinhard Penn, Bernhard Selymes
// Date : 16.12.2012
// Description : Header for Album.cpp
////////////////////////////////////////////////////////////////////////////

#ifndef ALBUM_H
#define ALBUM_H

#include "MusicComponent.h"
#include "Song.h"

class Album :
   public MusicComponent
{
public:
   //CTor
   Album(std::string Name, std::string Interpret);

   //virtual Destructor
   virtual ~Album();

   virtual void Accept(Visitor* visitor);
   void ForwardVisitor(Visitor* visitor);

   virtual TMusicKind GetType();
   std::string GetInterpret();
   size_t GetNumberOfEntries();

   virtual void AddMusic(MusicComponent* m);

private:
   std::string mInterpret;
   TMusicComponents mSongs;
};

#endif
```

```cpp
////////////////////////////////////////////////////////////////////////////
// Workfile : Album.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 16.12.2012
// Description : Implementation of class Album
////////////////////////////////////////////////////////////////////////////

#include <algorithm>
#include <iostream>
#include "Album.h"
#include "Visitor.h"

//CTor
Album::Album(std::string Name, std::string Interpret)
   : mInterpret(Interpret)
{
   mName = Name;
   mType = TAlbum;
}

//virtual Destructor
Album::~Album()
{}

void Album::Accept(Visitor* visitor)
{
   try
   {
      if(visitor == 0)
      {
         std::string error = "no valid pointer";
         throw (error);
      }
      visitor->Visit(this);
   }
   catch (std::string const& error)
   {
      std::cerr << "Error in Album::Accept: " << error << std::endl;
   }
   catch(...)
   {
      std::cerr << "Album::Accept: Unknown Exception occured" << std::endl;
   }
}

void Album::ForwardVisitor(Visitor* visitor)
{
   try
   {
      if(visitor == 0)
      {
         std::string error = "no valid visitor";
         throw (error);
      }
      std::for_each(mSongs.begin(),mSongs.end(),[=](MusicComponent* s)
      {
         s->Accept(visitor);
      });
   }
   catch (std::string const& error)
```

```
61     {
62        std::cerr << "Error in Album::ForwardVisitor: " << error << std::endl
             ;
63     }
64     catch(...)
65     {
66        std::cerr << "Album::ForwardVisitor: Unknown Exception occured" <<
             std::endl;
67     }
68  }
69
70  void Album::AddMusic(MusicComponent* m)
71  {
72     try
73     {
74        if(m == 0)
75        {
76           std::string error = "no valid pointer";
77           throw (error);
78        }
79        if(m->GetType() != TSong)
80        {
81           std::string error = "Tried to add a wrong object type to the song
                list";
82           throw (error);
83        }
84        mSongs.push_back(m);
85     }
86     catch (std::string const& error)
87     {
88        std::cerr << "Error in Album::AddMusic: " << error << std::endl;
89     }
90     catch(...)
91     {
92        std::cerr << "Album::Accept: Unknown Exception occured" << std::endl;
93     }
94  }
95
96  std::string Album::GetInterpret()
97  {
98     return mInterpret;
99  }
100
101 TMusicKind Album::GetType()
102 {
103    return mType;
104 }
105
106 size_t Album::GetNumberOfEntries()
107 {
108    size_t counter = 0;
109    std::for_each(mSongs.begin(),mSongs.end(),[=, &counter](MusicComponent*
          m)
110    {
111       counter += m->GetNumberOfEntries();
112    });
113    return counter;
114 }
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : MusicCollection.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 16.12.2012
5  // Description : Header for MusicCollection.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef MUSICCOLLECTION_H
9  #define MUSICCOLLECTION_H
10
11 #include "MusicComponent.h"
12
13 class MusicCollection :
14    public MusicComponent
15 {
16 public:
17    //CTor
18    MusicCollection(std::string Name);
19
20    //virtual Destructor
21    virtual ~MusicCollection();
22
23    virtual void Accept(Visitor* visitor);
24    void ForwardVisitor(Visitor* visitor);
25
26    virtual TMusicKind GetType();
27    size_t GetNumberOfEntries();
28
29    virtual void AddMusic(MusicComponent* m);
30 private:
31    TMusicComponents mMusicComponents;
32 };
33 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : MusicCollection.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 16.12.2012
// Description : Implementation of class MusicCollection
////////////////////////////////////////////////////////////////////////

#include <iostream>
#include <algorithm>
#include "MusicCollection.h"
#include "Visitor.h"

//CTor
MusicCollection::MusicCollection(std::string Name)
{
   mName = Name;
   mType = TMusicCollection;
}

//virtual Destructor
MusicCollection::~MusicCollection()
{}

void MusicCollection::Accept(Visitor* visitor)
{
   try
   {
      if(visitor == 0)
      {
         std::string error = "no valid pointer";
         throw (error);
      }
      visitor->Visit(this);
   }
   catch (std::string const& error)
   {
      std::cerr << "Error in MusicCollection::Accept: " << error << std::
         endl;
   }
   catch(...)
   {
      std::cerr << "MusicCollection::Accept: Unknown Exception occured" <<
         std::endl;
   }
}

void MusicCollection::ForwardVisitor(Visitor* visitor)
{
   try
   {
      if(visitor == 0)
      {
         std::string error = "no valid visitor";
         throw (error);
      }
      std::for_each(mMusicComponents.begin(),mMusicComponents.end(),[=](
         MusicComponent* m)
      {
         m->Accept(visitor);
      });
```

```
58     }
59     catch (std::string const& error)
60     {
61        std::cerr << "Error in MusicCollection::ForwardVisitor: " << error <<
              std::endl;
62     }
63     catch(...)
64     {
65        std::cerr << "MusicCollection::ForwardVisitor: Unknown Exception
              occured" << std::endl;
66     }
67  }
68
69  TMusicKind MusicCollection::GetType()
70  {
71     return mType;
72  }
73
74  void MusicCollection::AddMusic(MusicComponent* m)
75  {
76     try
77     {
78        //check for Null pointer and this pointer
79        if(m == 0 || m == this)
80        {
81           std::string error = "no valid pointer";
82           throw (error);
83        }
84        mMusicComponents.push_back(m);
85     }
86     catch (std::string const& error)
87     {
88        std::cerr << "Error in MusicCollection::AddMusic: " << error << std::
              endl;
89     }
90     catch(...)
91     {
92        std::cerr << "MusicCollection::AddMusic: Unknown Exception occured"
              << std::endl;
93     }
94  }
95
96  size_t MusicCollection::GetNumberOfEntries()
97  {
98     size_t counter = 0;
99     std::for_each(mMusicComponents.begin(),mMusicComponents.end(),[=, &
           counter](MusicComponent* m)
100    {
101       counter += m->GetNumberOfEntries();
102    });
103    return counter;
104 }
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : Visitor.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 6.11.2012
5  // Description : Interface for Visitors
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef VISITOR_H
9  #define VISITOR_H
10
11 class Song;
12 class Album;
13 class MusicCollection;
14
15 class Visitor
16 {
17 public:
18    //virtual Destructor
19    virtual ~Visitor() {};
20
21    virtual void Visit(Song* song) = 0;
22    virtual void Visit(Album* album) = 0;
23    virtual void Visit(MusicCollection* musicCollection) = 0;
24 };
25
26 #endif
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : TimeVisitor.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 6.11.2012
5  // Description : Header of TimeVisitor
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef TIMEVISITOR_H
9  #define TIMEVISITOR_H
10
11 #include "Object.h"
12 #include "Visitor.h"
13 #include "Song.h"
14 #include "Album.h"
15 #include "MusicCollection.h"
16
17 class TimeVisitor :
18    public Visitor,
19    public Object
20 {
21 public:
22    TimeVisitor() : mTime(0) {}
23
24    void Visit(Song* song);
25    void Visit(Album* album);
26    void Visit(MusicCollection* musicCollection);
27
28    size_t GetTime() const;
29
30 private:
31    size_t mTime;
32 };
33
34 #endif
```

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

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

void TimeVisitor::Visit(Song* song)
{
   try
   {
      if(song == 0)
      {
         std::string error = "no valid pointer";
         throw (error);
      }
      mTime = mTime + song->GetTime();
   }
   catch (std::string const& error)
   {
      std::cerr << "error in TimeVisitor::Visit(Song*): " << error << std::
         endl;
   }
   catch(...)
   {
      std::cerr << "TimeVisitor::Visit: Unknown Exception occured" << std::
         endl;
   }
}

void TimeVisitor::Visit(Album* album)
{
   try
   {
      if(album == 0)
      {
         std::string error = "no valid pointer";
         throw (error);
      }
      album->ForwardVisitor(this);
   }
   catch (std::string const& error)
   {
      std::cerr << "error in TimeVisitor::Visit(Album*): " << error << std
         ::endl;
   }
   catch(...)
   {
      std::cerr << "TimeVisitor::Visit: Unknown Exception occured" << std::
         endl;
   }
}

void TimeVisitor::Visit(MusicCollection* musicCollection)
{
   try
```

```
57     {
58        if(musicCollection == 0)
59        {
60           std::string error = "no valid pointer";
61           throw (error);
62        }
63        musicCollection->ForwardVisitor(this);
64     }
65     catch (std::string const& error)
66     {
67        std::cerr << "error in TimeVisitor::Visit(MusicCollection*): " <<
             error << std::endl;
68     }
69     catch(...)
70     {
71        std::cerr << "TimeVisitor::Visit: Unknown Exception occured" << std::
             endl;
72     }
73  }
74
75  size_t TimeVisitor::GetTime() const
76  {
77     return mTime;
78  }
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : SearchVisitor.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 6.11.2012
5  // Description : Header of SearchVisitor
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef SEARCHVISITOR_H
9  #define SEARCHVISITOR_H
10
11 #include <string>
12 #include <list>
13 #include "Object.h"
14 #include "Visitor.h"
15 #include "Song.h"
16 #include "Album.h"
17 #include "MusicCollection.h"
18
19 class SearchVisitor :
20    public Visitor,
21    public Object
22 {
23 public:
24    SearchVisitor(std::string const& name);
25
26    virtual void Visit(Song* song);
27    virtual void Visit(Album* album);
28    virtual void Visit(MusicCollection* musicCollection);
29
30    TMusicComponents* GetResults();
31
32 private:
33    std::string mName;
34    TMusicComponents mResults;
35 };
36
37 #endif
```

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

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

SearchVisitor::SearchVisitor(std::string const& name)
{
   try
   {
      if(name == "")
      {
         std::string error = "no valid name";
         throw (error);
      }
      mName = name;
   }
   catch (std::string const& error)
   {
      std::cerr << "error in SearchVisitor::SearchVisitor: " << error <<
         std::endl;
   }
   catch(...)
   {
      std::cerr << "SearchVisitor::Visit: Unknown Exception occured" << std
         ::endl;
   }
}

void SearchVisitor::Visit(Song* song)
{
   try
   {
      if(song == 0)
      {
         std::string error = "no valid pointer";
         throw (error);
      }
      //check if searched name is part of name of the song
      if((song->GetName()).find(mName, 0) != std::string::npos)
      {
         mResults.push_back(song);
      }
   }
   catch (std::string const& error)
   {
      std::cerr << "error in SearchVisitor::Visit(Song*): " << error << std
         ::endl;
   }
   catch(...)
   {
      std::cerr << "SearchVisitor::Visit: Unknown Exception occured" << std
         ::endl;
   }
}

```

```
57  void SearchVisitor::Visit(Album* album)
58  {
59     try
60     {
61        if(album == 0)
62        {
63           std::string error = "no valid pointer";
64           throw (error);
65        }
66        //check if searched name is part of name of the album
67        if((album->GetName()).find(mName, 0) != std::string::npos)
68        {
69           mResults.push_back(album);
70        }
71        album->ForwardVisitor(this);
72     }
73     catch (std::string const& error)
74     {
75        std::cerr << "error in SearchVisitor::Visit(Album*): " << error <<
              std::endl;
76     }
77     catch(...)
78     {
79        std::cerr << "SearchVisitor::Visit: Unknown Exception occured" << std
              ::endl;
80     }
81  }
82
83  void SearchVisitor::Visit(MusicCollection* musicCollection)
84  {
85     try
86     {
87        if(musicCollection == 0)
88        {
89           std::string error = "no valid pointer";
90           throw (error);
91        }
92        //check if searched name is part of name of the song
93        if((musicCollection->GetName()).find(mName, 0) != std::string::npos)
94        {
95           mResults.push_back(musicCollection);
96        }
97        musicCollection->ForwardVisitor(this);
98     }
99     catch (std::string const& error)
100    {
101       std::cerr << "error in SearchVisitor::Visit(MusicCollection*): " <<
              error << std::endl;
102    }
103    catch(...)
104    {
105       std::cerr << "SearchVisitor::Visit: Unknown Exception occured" << std
              ::endl;
106    }
107 }
108
109 TMusicComponents* SearchVisitor::GetResults()
110 {
111    return &mResults;
112 }
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : PlayVisitor.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 6.11.2012
5  // Description : Header of PlayVisitor.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef PLAYVISITOR_H
9  #define PLAYVISITOR_H
10
11 #include "Object.h"
12 #include "Visitor.h"
13 #include "Song.h"
14 #include "Album.h"
15 #include "MusicCollection.h"
16
17 class PlayVisitor :
18    public Visitor,
19    public Object
20 {
21 public:
22    virtual void Visit(Song* song);
23    virtual void Visit(Album* album);
24    virtual void Visit(MusicCollection* musicCollection);
25    PlayVisitor();
26 private:
27    int GetMinutes(size_t const seconds);
28    int GetSeconds(size_t const seconds);
29    int mCounter;
30 };
31
32 #endif
```

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

#include <string>
#include <iostream>
#include <iomanip>
#include "PlayVisitor.h"
#include "TimeVisitor.h"

PlayVisitor::PlayVisitor()
{
   mCounter = 0;
}

int PlayVisitor::GetMinutes(size_t const seconds)
{
   return seconds / 60;
}

int PlayVisitor::GetSeconds(size_t const seconds)
{
   return seconds % 60;
}

void PlayVisitor::Visit(Song* song)
{
   try
   {
      if(song == 0)
      {
         std::string error = "no valid pointer";
         throw (error);
      }
      mCounter++;
      std::cout << mCounter << ". " << song->GetName() << " " << " " << std::
         setfill('0') << std::setw(2)
              << GetMinutes(song->GetTime()) << ":" << GetSeconds(song->
                 GetTime()) << " " << "
              << song->GetInterpret() << std::endl;
   }
   catch (std::string const& error)
   {
      std::cerr << "error in PlayVisitor::Visit(Song*): " << error << std::
         endl;
   }
   catch(...)
   {
      std::cerr << "PlayVisitor::Visit: Unknown Exception occured" << std::
         endl;
   }
}

void PlayVisitor::Visit(Album* album)
{
   try
   {
```

```
57        if(album == 0)
58        {
59           std::string error = "no valid pointer";
60           throw (error);
61        }
62
63        TimeVisitor* timeVisitor = new TimeVisitor;
64
65        album->Accept(timeVisitor);
66
67        std::cout << "Album: " << album->GetName() << " (" << album->
             GetNumberOfEntries() << " Song(s)) " << " << "
68                << std::setfill('0') << std::setw(2) << GetMinutes(
                      timeVisitor->GetTime()) << ":"
69                << std::setfill('0') << std::setw(2) << GetSeconds(
                      timeVisitor->GetTime()) << std::endl;
70
71        album->ForwardVisitor(this);
72
73        delete timeVisitor; timeVisitor = 0;
74     }
75     catch (std::bad_alloc const& e)
76     {
77        std::cerr << e.what() << std::endl;
78     }
79     catch (std::string const& error)
80     {
81        std::cerr << "error in PlayVisitor::Visit(Album*): " << error << std
             ::endl;
82     }
83     catch(...)
84     {
85        std::cerr << "PlayVisitor::Visit: Unknown Exception occured" << std::
             endl;
86     }
87  }
88
89  void PlayVisitor::Visit(MusicCollection* musicCollection)
90  {
91     try
92     {
93        if(musicCollection == 0)
94        {
95           std::string error = "no valid pointer";
96           throw (error);
97        }
98
99        TimeVisitor* timeVisitor = new TimeVisitor;
100
101       musicCollection->Accept(timeVisitor);
102
103       std::cout << "Collection: " << musicCollection->GetName() << " (" <<
             musicCollection->GetNumberOfEntries() << " Song(s)) "
104               << " << " << std::setfill('0') << std::setw(2) << GetMinutes(
                     timeVisitor->GetTime()) << ":"
105               << std::setfill('0') << std::setw(2) << GetSeconds(
                     timeVisitor->GetTime()) << std::endl;
106
107       musicCollection->ForwardVisitor(this);
108
```

```
109         delete timeVisitor; timeVisitor = 0;
110      }
111      catch (std::bad_alloc const& e)
112      {
113         std::cerr << e.what() << std::endl;
114      }
115      catch (std::string const& error)
116      {
117         std::cerr << "error in PlayVisitor::Visit(MusicCollection*): " <<
               error << std::endl;
118      }
119      catch(...)
120      {
121         std::cerr << "PlayVisitor::Visit: Unknown Exception occured" << std::
               endl;
122      }
123   }
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : MusicPlayer.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 6.11.2012
5  // Description : Header of MusicPlayer.cpp
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef MUSICPLAYER_H
9  #define MUSICPLAYER_H
10
11 #include <string>
12 #include "Object.h"
13 #include "MusicComponent.h"
14
15 class MusicPlayer :
16    public Object
17 {
18 public:
19    void Add(MusicComponent* musicComponent);
20    size_t GetTime(MusicComponent * const musicComponent);
21    size_t GetTotalTime();
22    void Play();
23    void Remove(MusicComponent * const musicComponent);
24    void Search(std::string const& name);
25
26 private:
27    TMusicComponents mMusicComponents;
28 };
29
30 #endif
```

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

#include <iostream>
#include <algorithm>
#include <iterator>
#include "MusicPlayer.h"
#include "TimeVisitor.h"
#include "SearchVisitor.h"
#include "PlayVisitor.h"

void MusicPlayer::Add(MusicComponent* musicComponent)
{
   try
   {
      if(musicComponent == 0)
      {
         std::string error = "no valid pointer";
         throw (error);
      }
      mMusicComponents.push_back(musicComponent);  //only adds a pointer
   }
   catch (std::string const& error)
   {
      std::cerr << "error in MusicPlayer::Add(): " << error << std::endl;
   }
   catch(...)
   {
      std::cerr << "MusicPlayer::Add: Unknown Exception occured" << std::
         endl;
   }
}

size_t MusicPlayer::GetTime(MusicComponent * const musicComponent)
{
   try
   {
      if(musicComponent == 0)
      {
         std::string error = "no valid pointer";
         throw (error);
      }

      //check if element is in list
      bool exists = false;

      std::for_each(mMusicComponents.begin(),mMusicComponents.end(),[=,&
         exists](MusicComponent* m)
      {
         if(m == musicComponent)
         {
            exists = true;
         }
      });
      if(!exists)
      {
```

```
59          std::string error = "component doesnt exist in list";
60          throw (error);
61       }
62
63       TimeVisitor* timeVisitor = new TimeVisitor;
64       musicComponent->Accept(timeVisitor);
65       size_t tmp = timeVisitor->GetTime();
66       delete timeVisitor;
67
68       return tmp;
69    }
70    catch (std::bad_alloc const& e)
71    {
72       std::cerr << e.what() << std::endl;
73       return 0;
74    }
75    catch (std::string const& error)
76    {
77       std::cout << "error in MusicPlayer::GetTime(): " << error << std::
             endl;
78       return 0;
79    }
80    catch(...)
81    {
82       std::cerr << "MusicPlayer::GetTime: Unknown Exception occured" << std
             ::endl;
83       return 0;
84    }
85 }
86
87 size_t MusicPlayer::GetTotalTime()
88 {
89    try
90    {
91       TimeVisitor* timeVisitor = new TimeVisitor;
92
93       std::for_each(mMusicComponents.begin(),mMusicComponents.end(),[=](
             MusicComponent* m)
94       {
95          m->Accept(timeVisitor);
96       });
97
98       size_t tmp = timeVisitor->GetTime();
99       delete timeVisitor;
100      return tmp;
101   }
102   catch (std::bad_alloc const& e)
103   {
104      std::cerr << e.what() << std::endl;
105      return 0;
106   }
107   catch(...)
108   {
109      std::cerr << "MusicPlayer::GetTotalTime: Unknown Exception occured"
             << std::endl;
110      return 0;
111   }
112 }
113
114 void MusicPlayer::Play()
```

```
115 {
116    try
117    {
118       PlayVisitor* playVisitor = new PlayVisitor;
119 
120       std::for_each(mMusicComponents.begin(),mMusicComponents.end(),[=](
             MusicComponent* m)
121       {
122          m->Accept(playVisitor);
123       });
124 
125       delete playVisitor; playVisitor = 0;
126    }
127    catch (std::bad_alloc const& e)
128    {
129       std::cerr << e.what() << std::endl;
130    }
131    catch(...)
132    {
133       std::cerr << "MusicPlayer::Play: Unknown Exception occured" << std::
             endl;
134    }
135 }
136 
137 void MusicPlayer::Remove(MusicComponent * const musicComponent)
138 {
139    try
140    {
141       if(musicComponent == 0)
142       {
143          std::string error = "no valid pointer";
144          throw (error);
145       }
146       mMusicComponents.remove(musicComponent);  //only deletes the pointer,
              not the object
147    }
148    catch (std::string const& error)
149    {
150       std::cerr << "error in MusicPlayer::Remove(): " << error << std::endl
             ;
151    }
152    catch(...)
153    {
154       std::cerr << "MusicPlayer::Remove: Unknown Exception occured" << std
             ::endl;
155    }
156 }
157 
158 void MusicPlayer::Search(std::string const& name)
159 {
160    try
161    {
162       if(name == "")
163       {
164          std::string error = "no valid name";
165          throw (error);
166       }
167 
168       std::cout << "found medias: (search for \"" << name << "\")" << std::
             endl;
```

```
169
170         SearchVisitor* searchVisitor = new SearchVisitor(name);
171
172         std::for_each(mMusicComponents.begin(),mMusicComponents.end(),[=](
                MusicComponent* m)
173         {
174            m->Accept(searchVisitor);
175         });
176
177         TMusicComponents* tmp;
178         tmp = searchVisitor->GetResults();
179
180         std::for_each(tmp->begin(),tmp->end(),[=](MusicComponent* m)
181         {
182            std::cout << m->GetName() << std::endl;
183         });
184
185         delete searchVisitor; searchVisitor = 0;
186      }
187      catch (std::bad_alloc const& e)
188      {
189         std::cerr << e.what() << std::endl;
190      }
191      catch (std::string const& error)
192      {
193         std::cerr << "error in MusicPlayer::Search(): " << error << std::endl
                ;
194      }
195      catch(...)
196      {
197         std::cerr << "MusicPlayer::Search: Unknown Exception occured" << std
                ::endl;
198      }
199   }
```

```cpp
////////////////////////////////////////////////////////////////////////////
// Workfile : Main.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 17.11.2012
// Description : Testdriver for MusicPlayer
////////////////////////////////////////////////////////////////////////////

#include <iostream>
#include "MusicPlayer.h"
#include "MusicFactory.h"
#include "MusicComponent.h"

using namespace std;


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

   MusicPlayer* player = new MusicPlayer;
   MusicFactory* fact = new MusicFactory;

   cout << "Add: ";
   player->Add(0);
   cout << " ...done" << endl;

   cout << "GetTime: ";
   player->GetTime(0);
   cout << " ...done" << endl;

   cout << "GetTotalTime: ";
   player->GetTotalTime();
   cout << " ...done" << endl;

   cout << "Play: " << endl;
   player->Play();
   cout << " ...done" << endl;

   cout << "Remove: ";
   player->Remove(0);
   cout << " ...done" << endl;

   cout << "Search: ";
   player->Search("");
   cout << " ...done" << endl;

   cout << "Delete MusicPlayer: ";
   delete player; player=0;
   cout << " ...done" << endl;

   cout << "Delete MusicFactory: ";
   delete fact; fact=0;
   cout << " ...done" << endl;
   cout << endl << endl;
}

void NormalTestCase()
{
   cout << "Testcase1: Normal testcase with valid objects." << endl;

```

```
61      cout << "Create Objects: ";
62      MusicPlayer* player = new MusicPlayer;
63      MusicFactory* fact = new MusicFactory;
64
65      MusicComponent* Song1 = fact->CreateSong("Staring At The Sun","Americana
            ","The Offspring",132);
66      MusicComponent* Song2 = fact->CreateSong("Have You Ever","Americana","
            The Offspring",236);
67      MusicComponent* Song3 = fact->CreateSong("Living In Chaos","Conspiracy
            Of One","The Offspring",208);
68      MusicComponent* Song4 = fact->CreateSong("Psychosocial","All Hope Is
            Gone","Slipknot",283);
69
70      MusicComponent* Album1 = fact->CreateAlbum("Americana","The Offspring");
71      MusicComponent* Album2 = fact->CreateAlbum("Conspiracy Of One","The
            Offspring");
72
73      MusicComponent* Collection1 = fact->CreateMusicCollection("MyPlayList");
74      cout << " ...done" << endl;
75
76      cout << "Put Albums together: ";
77      Album1->AddMusic(Song1);
78      Album1->AddMusic(Song2);
79      Album2->AddMusic(Song3);
80      cout << " ...done" << endl;
81
82      cout << "Put an album into an album: ";
83      Album1->AddMusic(Album2);
84      cout << " ...done" << endl;
85
86      cout << "Put Collection together: ";
87      Collection1->AddMusic(Album1);
88      Collection1->AddMusic(Song4);
89      cout << " ...done" << endl;
90
91      cout << "Put a collection into itself: ";
92      Collection1->AddMusic(Collection1);
93      cout << " ...done" << endl;
94
95
96      cout << "Add stuff to the player: ";
97      player->Add(Song1);
98      player->Add(Collection1);
99      player->Add(Album2);
100     player->Add(Album1);
101     cout << " ...done" << endl;
102
103     cout << "GetTime: ";
104     cout << player->GetTime(Song4) << " seconds";
105     cout << " ...done" << endl;
106
107     cout << "GetTotalTime: ";
108     cout << player->GetTotalTime() << " seconds in player";
109     cout << " ...done" << endl;
110
111     cout << "Play: " << endl;
112     player->Play();
113     cout << " ...done" << endl;
114
115     cout << "Search: ";
```

```
116     player->Search("in");
117     cout << " ...done" << endl;
118 
119     cout << "Remove: ";
120     player->Remove(Collection1);
121     player->Remove(Album1);
122     cout << " ...done" << endl;
123 
124     cout << "Play after remove: " << endl;
125     player->Play();
126     cout << " ...done" << endl;
127 
128     cout << "Delete MusicPlayer: ";
129     delete player; player=0;
130     cout << " ...done" << endl;
131 
132     cout << "Delete MusicFactory: ";
133     delete fact; fact=0;
134     cout << " ...done" << endl;
135     cout << endl << endl;
136 }
137 
138 int main()
139 {
140     EmptyTestCase();
141     NormalTestCase();
142 
143     return 0;
144 }
```

6 Testausgaben

```
Testcase0: Empty testcase with NULL pointer.
Add: error in MusicPlayer::Add(): no valid pointer
 ...done
GetTime: error in MusicPlayer::GetTime(): no valid pointer
 ...done
GetTotalTime:  ...done
Play:
 ...done
Remove: error in MusicPlayer::Remove(): no valid pointer
 ...done
Search: error in MusicPlayer::Search(): no valid name
 ...done
Delete MusicPlayer:  ...done
Delete MusicFactory:  ...done


Testcase1: Normal testcase with valid objects.
Create Objects:  ...done
Put Albums together:  ...done
Put an album into an album: Error in Album::AddMusic:
Tried to add a wrong object type to the song list
 ...done
Put Collection together:  ...done
Put a collection into itself: Error in MusicCollection::AddMusic:
no valid pointer
 ...done
Add stuff to the player:  ...done
GetTime: error in MusicPlayer::GetTime(): component doesnt exist in list
0 seconds ...done
GetTotalTime: 1359 seconds in player ...done
Play:
1. Staring At The Sun << 02:12 << The Offspring
Collection: MyPlayList (3 Song(s))  << 10:51
Album: Americana (2 Song(s))  << 06:08
2. Staring At The Sun << 02:12 << The Offspring
3. Have You Ever << 03:56 << The Offspring
4. Psychosocial << 04:43 << Slipknot
Album: Conspiracy Of One (1 Song(s))  << 03:28
5. Living In Chaos << 03:28 << The Offspring
Album: Americana (2 Song(s))  << 06:08
6. Staring At The Sun << 02:12 << The Offspring
7. Have You Ever << 03:56 << The Offspring
 ...done
Search: found medias: (search for "in")
Staring At The Sun
Staring At The Sun
Living In Chaos
Staring At The Sun
```

```
 ...done
Remove:  ...done
Play after remove:
1. Staring At The Sun << 02:12 << The Offspring
Album: Conspiracy Of One (1 Song(s))  << 03:28
2. Living In Chaos << 03:28 << The Offspring
 ...done
Delete MusicPlayer:  ...done
Delete MusicFactory:  ...done
```