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

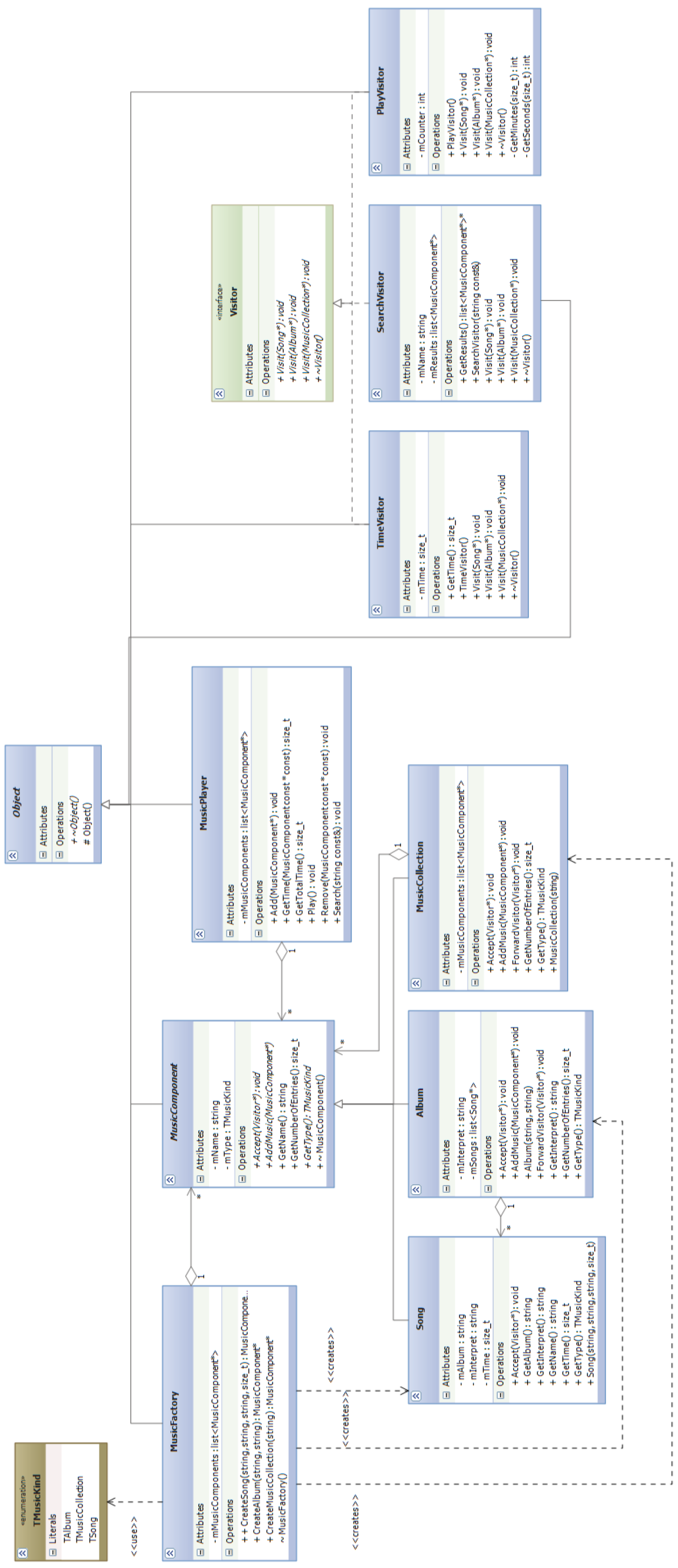
- geschätzte Mh: 14
- tatsächlich: Reinhard (7h), Bernhard (7h)

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


PlayVisitor
Attributes
- mCounter : int
Operations
+ PlayVisitor()
+ Visit(Song*) : void
+ Visit(Album*) : void
+ Visit(MusicCollection*) : void
+ ~Visitor()
- GetMinutes(size_t) : int
- GetSeconds(size_t) : int
«interface»
Visitor
Attributes
Operations
+ Visit(Song*) : void
+ Visit(Album*) : void
+ Visit(MusicCollection*) : void
+ ~Visitor()
SearchVisitor
Attributes
- mName : string
- mResults : list<MusicComponent*>
Operations
+ GetResults() : list<MusicComponent*>*
+ SearchVisitor(string const&)
+ Visit(Song*) : void
+ Visit(Album*) : void
+ Visit(MusicCollection*) : void
+ ~Visitor()
TimeVisitor
Attributes
- mTime : size_t
Operations
+ GetTime() : size_t
+ TimeVisitor()
+ Visit(Song*) : void
+ Visit(Album*) : void
+ Visit(MusicCollection*) : void
+ ~Visitor()
Object
Attributes
Operations
+ ~Object()
# Object()
MusicPlayer
Attributes
- mMusicComponents : list<MusicComponent*>
Operations
+ Add(MusicComponent*) : void
+ GetTime(MusicComponent const * const) : size_t
+ GetTotalTime() : size_t
+ Play() : void
+ Remove(MusicComponent const * const) : void
+ Search(string const&) : void
MusicCollection
Attributes
- mMusicComponents : list<MusicComponent*>
Operations
+ Accept(Visitor*) : void
+ AddMusic(MusicComponent*) : void
+ ForwardVisitor(Visitor*) : void
+ GetNumberOfEntries() : size_t
+ GetType() : TMusicKind
+ MusicCollection(string)
1
*
MusicComponent
Attributes
- mName : string
- mType : TMusicKind
Operations
+ Accept(Visitor*) : void
+ AddMusic(MusicComponent*)
+ GetName() : string
+ GetNumberOfEntries() : size_t
+ GetType() : TMusicKind
+ ~MusicComponent()
Album
Attributes
- mInterpret : string
- mSongs : list<Song*>
Operations
+ Accept(Visitor*) : void
+ AddMusic(MusicComponent*) : void
+ Album(string, string)
+ ForwardVisitor(Visitor*) : void
+ GetInterpret() : string
+ GetNumberOfEntries() : size_t
+ GetType() : TMusicKind
1
*
Song
Attributes
- mAlbum : string
- mInterpret : string
- mTime : size_t
Operations
+ Accept(Visitor*) : void
+ GetAlbum() : string
+ GetInterpret() : string
+ GetName() : string
+ GetTime() : size_t
+ GetType() : TMusicKind
+ Song(string, string, string, size_t)
MusicFactory
Attributes
- mMusicComponents : list<MusicComponent*>
Operations
+ + CreateSong(string, string, string, size_t) : MusicCompone...
+ CreateAlbum(string, string) : MusicComponent*
+ CreateMusicCollection(string) : MusicComponent*
~ MusicFactory()
«enumeration»
TMusicKind
Literals
TAlbum
TMusicCollection
TSong
<<use>>
<<creates>>
<<creates>>
<<creates>>

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.

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

```
///////////////////////////////////////////////////////////////////////////
// Workfile : Object.h
// Author : Reinhard Penn, Bernhard Selymes
// Date : 6.11.2012
// Description : Header for Object.cpp
///////////////////////////////////////////////////////////////////////////

#ifndef OBJECT_H
#define OBJECT_H

class Object
{
public:
   //virtual Destructor for baseclass
   virtual ~Object();
protected:
   //Default CTor for baseclass
   Object();
};

#endif
```

```
///////////////////////////////////////////////////////////////////////////
// Workfile : Object.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 6.11.2012
// Description : Baseclass with protected constructor
///////////////////////////////////////////////////////////////////////////

#include "Object.h"

Object::Object()
{}

Object::~Object()
{}
```

```
///////////////////////////////////////////////////////////////////////////
// Workfile : TMusicKind.h
// Author : Reinhard Penn, Bernhard Selymes
// Date : 16.12.2012
// Description : Enum for Musickind
///////////////////////////////////////////////////////////////////////////

#ifndef TMUSICKIND_H
#define TMUSICKIND_H

enum TMusicKind
{
   TMusicCollection,
   TAlbum,
   TSong
};

#endif
```

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

#ifndef MUSICFACTORY_H
#define MUSICFACTORY_H

#include "Object.h"
#include "TMusicKind.h"
#include "MusicComponent.h"

class MusicFactory :
   public Object
{
public:
   //virtual Destructor
   virtual ~MusicFactory();

   MusicComponent* CreateMusicCollection(std::string Name);
   MusicComponent* CreateAlbum(std::string Name, std::string Interpret);
   MusicComponent* CreateSong(std::string Name, std::string Album, std::
      string Interpret, size_t time);

private:
   TMusicComponents mMusicComponents;
};

#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;
}
```

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

#ifndef MUSICCOMPONENT_H
#define MUSICCOMPONENT_H

#include <list>
#include <string>
#include "Object.h"
#include "TMusicKind.h"

class Visitor;

class MusicComponent :
   public Object
{
public:
   //virtual Destructor for baseclass
   virtual ~MusicComponent();

   virtual void Accept(Visitor* visitor) = 0;

   virtual TMusicKind GetType() = 0;
   std::string GetName();
   virtual size_t GetNumberOfEntries() = 0;

   virtual void AddMusic(MusicComponent* m) = 0;
protected:
   std::string mName;
   TMusicKind mType;
};

typedef std::list<MusicComponent*> TMusicComponents;
typedef TMusicComponents::iterator TMusicComponentsItor;

#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;
}
```

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

#ifndef SONG_H
#define SONG_H

#include "MusicComponent.h"

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

   //virtual Destructor
   virtual ~Song();

   virtual void Accept(Visitor* visitor);

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

   virtual void AddMusic(MusicComponent* m);
private:
   std::string mAlbum;
   std::string mInterpret;

   size_t mTime;
};

#endif
```

```
///////////////////////////////////////////////////////////////////////////
// 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;
}

TMusicKind Song::GetType()
{
   return mType;
}

size_t Song::GetNumberOfEntries()
{
   return 1;
}
```

```
////////////////////////////////////////////////////////////////////////////
// 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)
   {
      std::cerr << "Error in Album::ForwardVisitor: " << error << std::endl
          ;
   }
   catch(...)
   {
      std::cerr << "Album::ForwardVisitor: Unknown Exception occured" <<
          std::endl;
   }
}

void Album::AddMusic(MusicComponent* m)
{
   try
   {
      if(m == 0)
      {
         std::string error = "no valid pointer";
         throw (error);
      }
      if(m->GetType() != TSong)
      {
         std::string error = "Tried to add a wrong object type to the song
             list";
         throw (error);
      }
      mSongs.push_back(m);
   }
   catch (std::string const& error)
   {
      std::cerr << "Error in Album::AddMusic: " << error << std::endl;
   }
   catch(...)
   {
      std::cerr << "Album::Accept: Unknown Exception occured" << std::endl;
   }
}

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

TMusicKind Album::GetType()
{
   return mType;
}

size_t Album::GetNumberOfEntries()
{
   size_t counter = 0;
   std::for_each(mSongs.begin(),mSongs.end(),[=, &counter](MusicComponent*
       m)
   {
      counter += m->GetNumberOfEntries();
   });
   return counter;
}
```

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

#ifndef MUSICCOLLECTION_H
#define MUSICCOLLECTION_H

#include "MusicComponent.h"

class MusicCollection :
   public MusicComponent
{
public:
   //CTor
   MusicCollection(std::string Name);

   //virtual Destructor
   virtual ~MusicCollection();

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

   virtual TMusicKind GetType();
   size_t GetNumberOfEntries();

   virtual void AddMusic(MusicComponent* m);
private:
   TMusicComponents mMusicComponents;
};
#endif
```

```
///////////////////////////////////////////////////////////////////////////
// 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);
      });
```

```cpp
    }
    catch (std::string const& error)
    {
        std::cerr << "Error in MusicCollection::ForwardVisitor: " << error <<
             std::endl;
    }
    catch(...)
    {
        std::cerr << "MusicCollection::ForwardVisitor: Unknown Exception
            occured" << std::endl;
    }
}

TMusicKind MusicCollection::GetType()
{
    return mType;
}

void MusicCollection::AddMusic(MusicComponent* m)
{
    try
    {
        //check for Null pointer and this pointer
        if(m == 0 || m == this)
        {
            std::string error = "no valid pointer";
            throw (error);
        }
        mMusicComponents.push_back(m);
    }
    catch (std::string const& error)
    {
        std::cerr << "Error in MusicCollection::AddMusic: " << error << std::
            endl;
    }
    catch(...)
    {
        std::cerr << "MusicCollection::AddMusic: Unknown Exception occured"
            << std::endl;
    }
}

size_t MusicCollection::GetNumberOfEntries()
{
    size_t counter = 0;
    std::for_each(mMusicComponents.begin(),mMusicComponents.end(),[=, &
        counter](MusicComponent* m)
    {
        counter += m->GetNumberOfEntries();
    });
    return counter;
}
```

```cpp
///////////////////////////////////////////////////////////////////////////
// Workfile : Visitor.h
// Author : Reinhard Penn, Bernhard Selymes
// Date : 6.11.2012
// Description : Interface for Visitors
///////////////////////////////////////////////////////////////////////////

#ifndef VISITOR_H
#define VISITOR_H

class Song;
class Album;
class MusicCollection;

class Visitor
{
public:
   //virtual Destructor
   virtual ~Visitor() {};

   virtual void Visit(Song* song) = 0;
   virtual void Visit(Album* album) = 0;
   virtual void Visit(MusicCollection* musicCollection) = 0;
};

#endif
```

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

#ifndef TIMEVISITOR_H
#define TIMEVISITOR_H

#include "Object.h"
#include "Visitor.h"
#include "Song.h"
#include "Album.h"
#include "MusicCollection.h"

class TimeVisitor :
   public Visitor,
   public Object
{
public:
   TimeVisitor() : mTime(0) {}

   void Visit(Song* song);
   void Visit(Album* album);
   void Visit(MusicCollection* musicCollection);

   size_t GetTime() const;

private:
   size_t mTime;
};

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

size_t TimeVisitor::GetTime() const
{
   return mTime;
}
```

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

#ifndef SEARCHVISITOR_H
#define SEARCHVISITOR_H

#include <string>
#include <list>
#include "Object.h"
#include "Visitor.h"
#include "Song.h"
#include "Album.h"
#include "MusicCollection.h"

class SearchVisitor :
   public Visitor,
   public Object
{
public:
   SearchVisitor(std::string const& name);

   virtual void Visit(Song* song);
   virtual void Visit(Album* album);
   virtual void Visit(MusicCollection* musicCollection);

   TMusicComponents* GetResults();

private:
   std::string mName;
   TMusicComponents mResults;
};

#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;
   }
}
```

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

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

TMusicComponents* SearchVisitor::GetResults()
{
   return &mResults;
}
```

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

#ifndef PLAYVISITOR_H
#define PLAYVISITOR_H

#include "Object.h"
#include "Visitor.h"
#include "Song.h"
#include "Album.h"
#include "MusicCollection.h"

class PlayVisitor :
   public Visitor,
   public Object
{
public:
   virtual void Visit(Song* song);
   virtual void Visit(Album* album);
   virtual void Visit(MusicCollection* musicCollection);
   PlayVisitor();
private:
   int GetMinutes(size_t const seconds);
   int GetSeconds(size_t const seconds);
   int mCounter;
};

#endif
```

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

#include <string>
#include <iostream>
#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() << " << " << 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
   {
      if(album == 0)
      {
         std::string error = "no valid pointer";
         throw (error);
      }

      TimeVisitor* timeVisitor = new TimeVisitor;

      album->Accept(timeVisitor);
      timeVisitor->GetTime();

      std::cout << "Album: " << album->GetName() << "(" << album->GetNumberOfEntries() << " Songs)"
         << GetMinutes(timeVisitor->GetTime()) << ":" << GetSeconds(timeVisitor->GetTime()) << std::endl;
      album->ForwardVisitor(this);

      delete timeVisitor; timeVisitor = 0;
   }
   catch (std::bad_alloc const& e)
   {
      std::cerr << e.what() << std::endl;
   }
   catch (std::string const& error)
   {
      std::cerr << "error in PlayVisitor::Visit(Album*): " << error << std::endl;
   }
   catch(...)
   {
      std::cerr << "PlayVisitor::Visit: Unknown Exception occured" << std::endl;
   }
}

void PlayVisitor::Visit(MusicCollection* musicCollection)
{
   try
   {
      if(musicCollection == 0)
      {
         std::string error = "no valid pointer";
         throw (error);
      }

      TimeVisitor* timeVisitor = new TimeVisitor;

      musicCollection->Accept(timeVisitor);
      timeVisitor->GetTime();

      std::cout << "Collection: " << musicCollection->GetName() << "(" << musicCollection->GetNumberOfEntries()
         << " Songs)" << GetMinutes(timeVisitor->GetTime()) << ":" << GetSeconds(timeVisitor->GetTime()) << std::endl;
      musicCollection->ForwardVisitor(this);

      delete timeVisitor; timeVisitor = 0;
   }
   catch (std::bad_alloc const& e)
   {
      std::cerr << e.what() << std::endl;
   }
   catch (std::string const& error)
   {
      std::cerr << "error in PlayVisitor::Visit(MusicCollection*): " <<
         error << std::endl;
   }
   catch(...)
   {
      std::cerr << "PlayVisitor::Visit: Unknown Exception occured" << std::
         endl;
   }
}
```

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

#ifndef MUSICPLAYER_H
#define MUSICPLAYER_H

#include <string>
#include "Object.h"
#include "MusicComponent.h"

class MusicPlayer :
   public Object
{
public:
   void Add(MusicComponent* musicComponent);
   size_t GetTime(MusicComponent * const musicComponent);
   size_t GetTotalTime();
   void Play();
   void Remove(MusicComponent * const musicComponent);
   void Search(std::string const& name);

private:
   TMusicComponents mMusicComponents;
};

#endif
```

```
///////////////////////////////////////////////////////////////////////////
// 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)
      {
```

```cpp
         std::string error = "component doesnt exist in list";
         throw (error);
      }

      TimeVisitor* timeVisitor = new TimeVisitor;
      musicComponent->Accept(timeVisitor);
      size_t tmp = timeVisitor->GetTime();
      delete timeVisitor;

      return tmp;
   }
   catch (std::bad_alloc const& e)
   {
      std::cerr << e.what() << std::endl;
      return 0;
   }
   catch (std::string const& error)
   {
      std::cout << "error in MusicPlayer::GetTime(): " << error << std::endl;
      return 0;
   }
   catch(...)
   {
      std::cerr << "MusicPlayer::GetTime: Unknown Exception occured" << std::endl;
      return 0;
   }
}

size_t MusicPlayer::GetTotalTime()
{
   try
   {
      TimeVisitor* timeVisitor = new TimeVisitor;

      std::for_each(mMusicComponents.begin(),mMusicComponents.end(),[=](MusicComponent* m)
      {
         m->Accept(timeVisitor);
      });

      size_t tmp = timeVisitor->GetTime();
      delete timeVisitor;
      return tmp;
   }
   catch (std::bad_alloc const& e)
   {
      std::cerr << e.what() << std::endl;
      return 0;
   }
   catch(...)
   {
      std::cerr << "MusicPlayer::GetTotalTime: Unknown Exception occured" << std::endl;
      return 0;
   }
}

void MusicPlayer::Play()
```

```cpp
{
   try
   {
      PlayVisitor* playVisitor = new PlayVisitor;

      std::for_each(mMusicComponents.begin(),mMusicComponents.end(),[=](MusicComponent* m)
      {
         m->Accept(playVisitor);
      });

      delete playVisitor; playVisitor = 0;
   }
   catch (std::bad_alloc const& e)
   {
      std::cerr << e.what() << std::endl;
   }
   catch(...)
   {
      std::cerr << "MusicPlayer::Play: Unknown Exception occured" << std::endl;
   }
}

void MusicPlayer::Remove(MusicComponent * const musicComponent)
{
   try
   {
      if(musicComponent == 0)
      {
         std::string error = "no valid pointer";
         throw (error);
      }
      mMusicComponents.remove(musicComponent);  //only deletes the pointer, not the object
   }
   catch (std::string const& error)
   {
      std::cerr << "error in MusicPlayer::Remove(): " << error << std::endl;
   }
   catch(...)
   {
      std::cerr << "MusicPlayer::Remove: Unknown Exception occured" << std::endl;
   }
}

void MusicPlayer::Search(std::string const& name)
{
   try
   {
      if(name == "")
      {
         std::string error = "no valid name";
         throw (error);
      }

      std::cout << "found medias: (search for \"" << name << "\")" << std::endl;

      SearchVisitor* searchVisitor = new SearchVisitor(name);

      std::for_each(mMusicComponents.begin(),mMusicComponents.end(),[=](MusicComponent* m)
      {
         m->Accept(searchVisitor);
      });

      TMusicComponents* tmp;
      tmp = searchVisitor->GetResults();

      std::for_each(tmp->begin(),tmp->end(),[=](MusicComponent* m)
      {
         std::cout << m->GetName() << std::endl;
      });

      delete searchVisitor; searchVisitor = 0;
   }
   catch (std::bad_alloc const& e)
   {
      std::cerr << e.what() << std::endl;
   }
   catch (std::string const& error)
   {
      std::cerr << "error in MusicPlayer::Search(): " << error << std::endl;
   }
   catch(...)
   {
      std::cerr << "MusicPlayer::Search: Unknown Exception occured" << std::endl;
   }
}
```

```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: ";
   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;

```

```cpp
    cout << "Create Objects: ";
    MusicPlayer* player = new MusicPlayer;
    MusicFactory* fact = new MusicFactory;

    MusicComponent* Song1 = fact->CreateSong("Staring At The Sun","Americana","The Offspring",132);
    MusicComponent* Song2 = fact->CreateSong("Have You Ever","Americana","The Offspring",236);
    MusicComponent* Song3 = fact->CreateSong("Living In Chaos","Conspiracy Of One","The Offspring",208);
    MusicComponent* Song4 = fact->CreateSong("Psychosocial","All Hope Is Gone","Slipknot",283);

    MusicComponent* Album1 = fact->CreateAlbum("Americana","The Offspring");
    MusicComponent* Album2 = fact->CreateAlbum("Conspiracy Of One","The Offspring");

    MusicComponent* Collection1 = fact->CreateMusicCollection("MyPlayList");
    cout << " ...done" << endl;

    cout << "Put Albums together: ";
    Album1->AddMusic(Song1);
    Album1->AddMusic(Song2);
    Album2->AddMusic(Song3);
    cout << " ...done" << endl;

    cout << "Put an album into an album: ";
    Album1->AddMusic(Album2);
    cout << " ...done" << endl;

    cout << "Put Collection together: ";
    Collection1->AddMusic(Album1);
    Collection1->AddMusic(Song4);
    cout << " ...done" << endl;

    cout << "Put a collection into itself: ";
    Collection1->AddMusic(Collection1);
    cout << " ...done" << endl;


    cout << "Add stuff to the player: ";
    player->Add(Song1);
    player->Add(Collection1);
    player->Add(Album2);
    player->Add(Album1);
    cout << " ...done" << endl;

    cout << "GetTime: ";
    cout << player->GetTime(Song4) << " seconds";
    cout << " ...done" << endl;

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

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

    cout << "Search: ";
```

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

   cout << "Remove: ";
   player->Remove(Collection1);
   player->Remove(Album1);
   cout << " ...done" << endl;

   cout << "Play after remove: ";
   player->Play();
   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;
}

int main()
{
   EmptyTestCase();
   NormalTestCase();

   return 0;
}
```

6 Testausgaben