

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

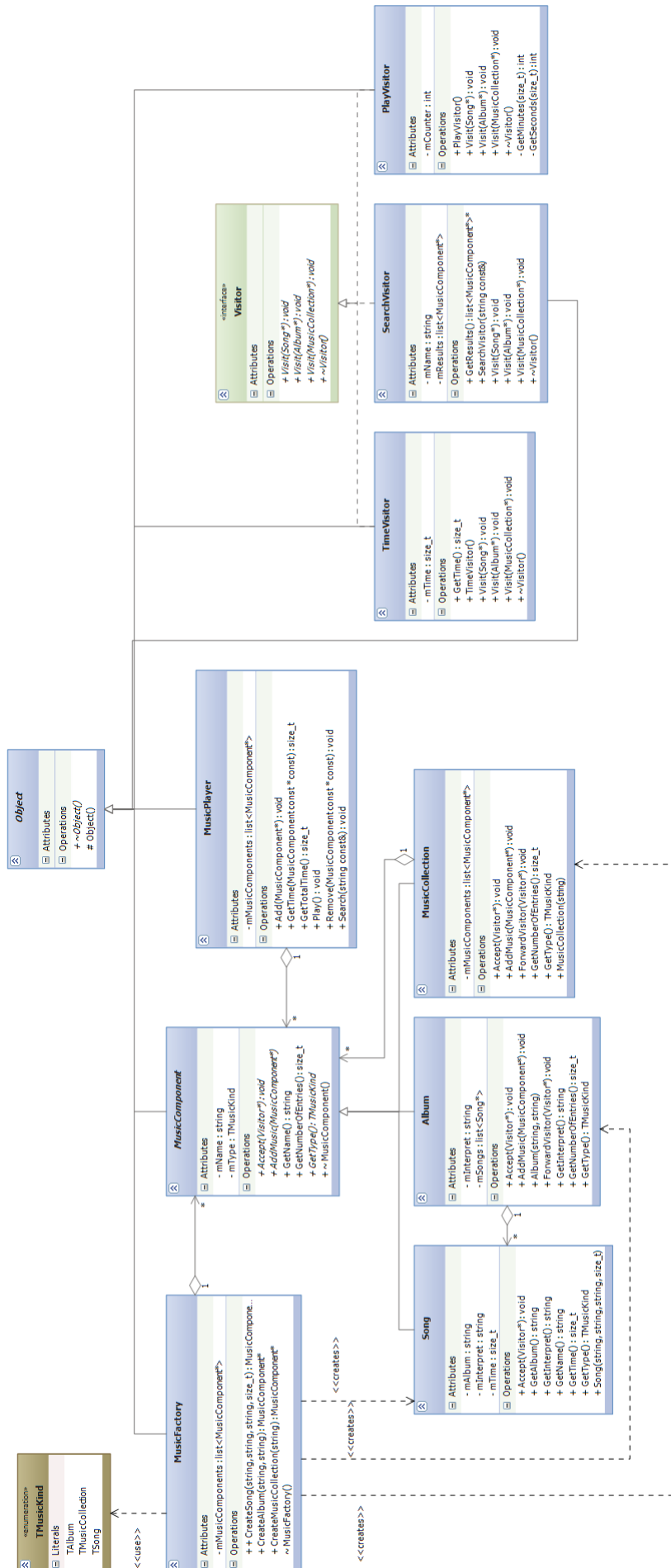
- geschätzte Mh: 14
- tatsächlich: Reinhard (7h), Bernhard (7h)

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 wird 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.

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:

Rückgabetyt: 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*

Rückgabetyt: 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*

Rückgabotyp: void

Ruft die Funktion Visit vom übergebenen Visitor mit sich selbst auf.

Methode "ForwardVisitor":

Schnittstelle:

Parameter: Visitor*

Rückgabotyp: void

Ruft für alle Lieder die Funktion "Accept" mit dem Visitor auf.

Methode "GetTime":

Schnittstelle:

Rückgabotyp: void

Ruft für alle Lieder die Methode "Accept" mit dem Visitor auf.

Methode "AddMusic":

Schnittstelle:

Parameter: MusicComponent*

Rückgabotyp: 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*

Rückgabotyp: void

Ruft die Funktion Visit vom übergebenen Visitor mit sich selbst auf.

Methode "ForwardVisitor":

Schnittstelle:

Rückgabotyp: void

Ruft für alle Musikmedien die Methode "Accept" mit dem Visitor auf.

Methode "GetNumberOfEntries":

Schnittstelle:

Rückgabotyp: 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*

Rückgabotyp: void

Pure virtual function.

4.9 Klasse "TimeVisitor"

Hat einen Member der die gesamte Dauer speichert.

Methode "Visit":

Schnittstelle:

Parameter: Song*

Rückgabotyp: void

Addiert zum Gesamtdauermember die Dauer vom Lied.

Methode "Visit":

Schnittstelle:

Parameter: Album*

Rückgabotyp: void

Ruft die Funktion "ForwardVisitor" vom Album mit sich selbst (Visitor) auf.

Methode "Visit":

Schnittstelle:

Parameter: MusicCollection*

Rückgabotyp: 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*

Rückgabotyp: void

Schaut ob der gesuchte Name Teil des Namens vom Lied ist und speichert in ggf. in der Liste.

Methode "Visit":

Schnittstelle:

Parameter: Album*

Rückgabotyp: void

Ruft die Funktion "ForwardVisitor" vom Album mit sich selbst (Visitor) auf.

Methode "Visit":

Schnittstelle:

Parameter: MusicCollection*

Rückgabotyp: 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*

Rückgabotyp: void

Gibt die Daten des Liedes formatiert auf der Konsole aus und erhöht den Zähler.

Methode "Visit":

Schnittstelle:

Parameter: Album*

Rückgabotyp: 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*

Rückgabotyp: void

Legt einen TimeVisitor an. Dieser wird von der Kollektion akzeptiert. Die Informationen der Kollektion werden auf der Konsole ausgegeben. Danach wird der Visitor den Elementen in der Kollektion weitergegeben.

4.12 Klasse "MusicPlayer"

Hat eine Liste mit Musik-Medien.

Methode "GetTime":

Schnittstelle:

Parameter: MusicComponent* const

Rückgabotyp: size_t

Prüft zuerst ob das Element überhaupt vorhanden ist. Dann wird via eines TimeVisitors die Dauer des Elements ermittelt.

Methode "GetTotalTime":

Schnittstelle:

Rückgabotyp: size_t

Via eines TimeVisitors die Dauer aller Elemente ermittelt.

Methode "Play":

Schnittstelle:

Rückgabety: void

Via eines PlayVisitors wird die gesamte Abspielliste ausgegeben.

Methode "Search":

Schnittstelle:

Parameter: string const&

Rückgabety: 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
22
23  //////////////////////////////////////
24  // Workfile : Object.cpp
25  // Author : Reinhard Penn, Bernhard Selymes
26  // Date : 6.11.2012
27  // Description : Baseclass with protected constructor
28  //////////////////////////////////////
29
30  #include "Object.h"
31
32  Object::Object()
33  {}
34
35  Object::~~Object()
36  {}
```

```
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

```

```

1  //////////////////////////////////////
2  // Workfile : MusicFactory.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 16.12.2012
5  // Description : Implementation of class MusicFactory
6  //////////////////////////////////////
7
8  #include <algorithm>
9  #include "MusicFactory.h"
10 #include "MusicCollection.h"
11 #include "Album.h"
12 #include "Song.h"
13
14 MusicFactory::~MusicFactory()
15 {
16     std::for_each(mMusicComponents.begin(),mMusicComponents.end(), [&](
17         MusicComponent* m)
18     {
19         delete m;
20     });
21 }
22 MusicComponent* MusicFactory::CreateMusicCollection(std::string Name)
23 {
24     MusicComponent* m = new MusicCollection(Name);
25     mMusicComponents.push_back(m);
26     return m;
27 }
28
29 MusicComponent* MusicFactory::CreateAlbum(std::string Name, std::string
    Interpret)
30 {
31     MusicComponent* m = new Album(Name,Interpret);
32     mMusicComponents.push_back(m);
33     return m;
34 }
35
36 MusicComponent* MusicFactory::CreateSong(std::string Name, std::string
    Album, std::string Interpret, size_t time)
37 {
38     MusicComponent* m = new Song(Name,Album,Interpret,time);
39     mMusicComponents.push_back(m);
40     return m;
41 }

```

```

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

```

```
1  //////////////////////////////////////
2  // Workfile : MusicComponent.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 16.12.2012
5  // Description : Implementation of class MusicComponent
6  //////////////////////////////////////
7
8  #include "MusicComponent.h"
9
10 //virtual Destructor for baseclass
11 MusicComponent::~MusicComponent()
12 {}
13
14 std::string MusicComponent::GetName()
15 {
16     return mName;
17 }
```

```

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

```

```

1  //////////////////////////////////////
2  // Workfile : Song.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 16.12.2012
5  // Description : Implementation of class Song
6  //////////////////////////////////////
7
8  #include <iostream>
9  #include "Song.h"
10 #include "Visitor.h"
11
12 //CTor
13 Song::Song(std::string Name, std::string Album, std::string Interpret,
14             size_t time)
15     : mAlbum(Album), mInterpret(Interpret), mTime(time)
16 {
17     mName = Name;
18     mType = TSong;
19 }
20 //virtual Destructor
21 Song::~~Song()
22 {}
23
24 void Song::Accept(Visitor* visitor)
25 {
26     try
27     {
28         if(visitor == 0)
29         {
30             std::string error = "no valid pointer";
31             throw (error);
32         }
33         visitor->Visit(this);
34     }
35     catch (std::string const& error)
36     {
37         std::cerr << "Error in Song::Accept: " << error << std::endl;
38     }
39     catch(...)
40     {
41         std::cerr << "Song::Accept: Unknown Exception occured" << std::endl;
42     }
43 }
44
45 size_t Song::GetTime()
46 {
47     return mTime;
48 }
49
50 std::string Song::GetInterpret()
51 {
52     return mInterpret;
53 }
54
55 void Song::AddMusic(MusicComponent* m)
56 {
57     std::string error = "This functions is not implemented and should not be
58         used";
59     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 }
```

```

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

```

```

1  //////////////////////////////////////
2  // Workfile : Album.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 16.12.2012
5  // Description : Implementation of class Album
6  //////////////////////////////////////
7
8  #include <algorithm>
9  #include <iostream>
10 #include "Album.h"
11 #include "Visitor.h"
12
13 //CTOR
14 Album::Album(std::string Name, std::string Interpret)
15     : mInterpret(Interpret)
16 {
17     mName = Name;
18     mType = TAlbum;
19 }
20
21 //virtual Destructor
22 Album::~~Album()
23 {}
24
25 void Album::Accept(Visitor* visitor)
26 {
27     try
28     {
29         if(visitor == 0)
30         {
31             std::string error = "no valid pointer";
32             throw (error);
33         }
34         visitor->Visit(this);
35     }
36     catch (std::string const& error)
37     {
38         std::cerr << "Error in Album::Accept: " << error << std::endl;
39     }
40     catch(...)
41     {
42         std::cerr << "Album::Accept: Unknown Exception occured" << std::endl;
43     }
44 }
45
46 void Album::ForwardVisitor(Visitor* visitor)
47 {
48     try
49     {
50         if(visitor == 0)
51         {
52             std::string error = "no valid visitor";
53             throw (error);
54         }
55         std::for_each(mSongs.begin(), mSongs.end(), [=] (MusicComponent* s)
56         {
57             s->Accept(visitor);
58         });
59     }
60     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

```

```

1  //////////////////////////////////////
2  // Workfile : MusicCollection.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 16.12.2012
5  // Description : Implementation of class MusicCollection
6  //////////////////////////////////////
7
8  #include <iostream>
9  #include <algorithm>
10 #include "MusicCollection.h"
11 #include "Visitor.h"
12
13 //CTOR
14 MusicCollection::MusicCollection(std::string Name)
15 {
16     mName = Name;
17     mType = TMusicCollection;
18 }
19
20 //virtual Destructor
21 MusicCollection::~MusicCollection()
22 {}
23
24 void MusicCollection::Accept(Visitor* visitor)
25 {
26     try
27     {
28         if(visitor == 0)
29         {
30             std::string error = "no valid pointer";
31             throw (error);
32         }
33         visitor->Visit(this);
34     }
35     catch (std::string const& error)
36     {
37         std::cerr << "Error in MusicCollection::Accept: " << error << std::
            endl;
38     }
39     catch(...)
40     {
41         std::cerr << "MusicCollection::Accept: Unknown Exception occured" <<
            std::endl;
42     }
43 }
44
45 void MusicCollection::ForwardVisitor(Visitor* visitor)
46 {
47     try
48     {
49         if(visitor == 0)
50         {
51             std::string error = "no valid visitor";
52             throw (error);
53         }
54         std::for_each(mMusicComponents.begin(), mMusicComponents.end(), [=] (
            MusicComponent* m)
55         {
56             m->Accept(visitor);
57         });

```

```

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

```

```

1  //////////////////////////////////////
2  // Workfile : TimeVisitor.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 6.11.2012
5  // Description : Implementation of class TimeVisitor
6  //////////////////////////////////////
7
8  #include <string>
9  #include <iostream>
10 #include "TimeVisitor.h"
11
12 void TimeVisitor::Visit(Song* song)
13 {
14     try
15     {
16         if(song == 0)
17         {
18             std::string error = "no valid pointer";
19             throw (error);
20         }
21         mTime = mTime + song->GetTime();
22     }
23     catch (std::string const& error)
24     {
25         std::cerr << "error in TimeVisitor::Visit(Song*): " << error << std::
26             endl;
27     }
28     catch(...)
29     {
30         std::cerr << "TimeVisitor::Visit: Unknown Exception occured" << std::
31             endl;
32     }
33 }
34
35 void TimeVisitor::Visit(Album* album)
36 {
37     try
38     {
39         if(album == 0)
40         {
41             std::string error = "no valid pointer";
42             throw (error);
43         }
44         album->ForwardVisitor(this);
45     }
46     catch (std::string const& error)
47     {
48         std::cerr << "error in TimeVisitor::Visit(Album*): " << error << std
49             ::endl;
50     }
51     catch(...)
52     {
53         std::cerr << "TimeVisitor::Visit: Unknown Exception occured" << std::
54             endl;
55     }
56 }
57
58 void TimeVisitor::Visit(MusicCollection* musicCollection)
59 {
60     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*): " <<
68             error << std::endl;
69     }
70     catch(...)
71     {
72         std::cerr << "TimeVisitor::Visit: Unknown Exception occured" << std::
73             endl;
74     }
75 }
76 size_t TimeVisitor::GetTime() const
77 {
78     return mTime;
79 }

```

```

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

```

```

1  //////////////////////////////////////
2  // Workfile : SearchVisitor.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 6.11.2012
5  // Description : Implementation of class SearchVisitor
6  //////////////////////////////////////
7
8  #include <iostream>
9  #include "SearchVisitor.h"
10
11 SearchVisitor::SearchVisitor(std::string const& name)
12 {
13     try
14     {
15         if(name == "")
16         {
17             std::string error = "no valid name";
18             throw (error);
19         }
20         mName = name;
21     }
22     catch (std::string const& error)
23     {
24         std::cerr << "error in SearchVisitor::SearchVisitor: " << error <<
                std::endl;
25     }
26     catch(...)
27     {
28         std::cerr << "SearchVisitor::Visit: Unknown Exception occured" << std
                ::endl;
29     }
30 }
31
32 void SearchVisitor::Visit(Song* song)
33 {
34     try
35     {
36         if(song == 0)
37         {
38             std::string error = "no valid pointer";
39             throw (error);
40         }
41         //check if searched name is part of name of the song
42         if((song->GetName()).find(mName, 0) != std::string::npos)
43         {
44             mResults.push_back(song);
45         }
46     }
47     catch (std::string const& error)
48     {
49         std::cerr << "error in SearchVisitor::Visit(Song*): " << error << std
                ::endl;
50     }
51     catch(...)
52     {
53         std::cerr << "SearchVisitor::Visit: Unknown Exception occured" << std
                ::endl;
54     }
55 }
56

```

```

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

```

```

1  //////////////////////////////////////
2  // Workfile : PlayVisitor.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 6.11.2012
5  // Description : Implementation of class PlayVisitor
6  //////////////////////////////////////
7
8  #include <string>
9  #include <iostream>
10 #include "PlayVisitor.h"
11 #include "TimeVisitor.h"
12
13 PlayVisitor::PlayVisitor()
14 {
15     mCounter = 0;
16 }
17
18 int PlayVisitor::GetMinutes(size_t const seconds)
19 {
20     return seconds / 60;
21 }
22
23 int PlayVisitor::GetSeconds(size_t const seconds)
24 {
25     return seconds % 60;
26 }
27
28 void PlayVisitor::Visit(Song* song)
29 {
30     try
31     {
32         if(song == 0)
33         {
34             std::string error = "no valid pointer";
35             throw (error);
36         }
37         mCounter++;
38         std::cout << mCounter << ". " << song->GetName() << " << " <<
39             GetMinutes(song->GetTime()) << ":" <<
40             GetSeconds(song->GetTime()) << " << " << song->GetInterpret()
41             << std::endl;
42     }
43     catch (std::string const& error)
44     {
45         std::cerr << "error in PlayVisitor::Visit(Song*): " << error << std::
46             endl;
47     }
48     catch(...)
49     {
50         std::cerr << "PlayVisitor::Visit: Unknown Exception occured" << std::
51             endl;
52     }
53 }
54
55 void PlayVisitor::Visit(Album* album)
56 {
57     try
58     {
59         if(album == 0)
60         {

```



```

57         std::string error = "no valid pointer";
58         throw (error);
59     }
60
61     TimeVisitor* timeVisitor = new TimeVisitor;
62
63     album->Accept(timeVisitor);
64     timeVisitor->GetTime();
65
66     std::cout << "Album: " << album->GetName() << "(" << album->
        GetNumberOfEntries() << " Songs)"
67     << GetMinutes(timeVisitor->GetTime()) << ":" << GetSeconds(
        timeVisitor->GetTime()) << std::endl;
68     album->ForwardVisitor(this);
69
70     delete timeVisitor; timeVisitor = 0;
71 }
72 catch (std::bad_alloc const& e)
73 {
74     std::cerr << e.what() << std::endl;
75 }
76 catch (std::string const& error)
77 {
78     std::cerr << "error in PlayVisitor::Visit(Album*): " << error << std
        ::endl;
79 }
80 catch(...)
81 {
82     std::cerr << "PlayVisitor::Visit: Unknown Exception occured" << std::
        endl;
83 }
84 }
85
86 void PlayVisitor::Visit(MusicCollection* musicCollection)
87 {
88     try
89     {
90         if(musicCollection == 0)
91         {
92             std::string error = "no valid pointer";
93             throw (error);
94         }
95
96         TimeVisitor* timeVisitor = new TimeVisitor;
97
98         musicCollection->Accept(timeVisitor);
99         timeVisitor->GetTime();
100
101         std::cout << "Collection: " << musicCollection->GetName() << "(" <<
            musicCollection->GetNumberOfEntries()
102         << " Songs)" << GetMinutes(timeVisitor->GetTime()) << ":" <<
            GetSeconds(timeVisitor->GetTime()) << std::endl;
103         musicCollection->ForwardVisitor(this);
104
105         delete timeVisitor; timeVisitor = 0;
106     }
107     catch (std::bad_alloc const& e)
108     {
109         std::cerr << e.what() << std::endl;
110     }

```

```
111     catch (std::string const& error)
112     {
113         std::cerr << "error in PlayVisitor::Visit(MusicCollection*): " <<
            error << std::endl;
114     }
115     catch(...)
116     {
117         std::cerr << "PlayVisitor::Visit: Unknown Exception occured" << std:::
            endl;
118     }
119 }
```

```

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

```

```

1  //////////////////////////////////////
2  // Workfile : MusicPlayer.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 6.11.2012
5  // Description : Implementation of class MusicPlayer
6  //////////////////////////////////////
7
8  #include <iostream>
9  #include <algorithm>
10 #include <iterator>
11 #include "MusicPlayer.h"
12 #include "TimeVisitor.h"
13 #include "SearchVisitor.h"
14 #include "PlayVisitor.h"
15
16 void MusicPlayer::Add(MusicComponent* musicComponent)
17 {
18     try
19     {
20         if(musicComponent == 0)
21         {
22             std::string error = "no valid pointer";
23             throw (error);
24         }
25         mMusicComponents.push_back(musicComponent); //only adds a pointer
26     }
27     catch (std::string const& error)
28     {
29         std::cerr << "error in MusicPlayer::Add(): " << error << std::endl;
30     }
31     catch(...)
32     {
33         std::cerr << "MusicPlayer::Add: Unknown Exception occured" << std::endl;
34     }
35 }
36
37 size_t MusicPlayer::GetTime(MusicComponent * const musicComponent)
38 {
39     try
40     {
41         if(musicComponent == 0)
42         {
43             std::string error = "no valid pointer";
44             throw (error);
45         }
46
47         //check if element is in list
48         bool exists = false;
49
50         std::for_each(mMusicComponents.begin(), mMusicComponents.end(), [=, &
51             exists](MusicComponent* m)
52         {
53             if(m == musicComponent)
54             {
55                 exists = true;
56             }
57         });
58         if(!exists)
59         {

```

```

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(), [=] (
121             MusicComponent* m)
122         {
123             m->Accept(playVisitor);
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::
134             endl;
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,
147             not the object
148     }
149     catch (std::string const& error)
150     {
151         std::cerr << "error in MusicPlayer::Remove(): " << error << std::endl
152             ;
153     }
154     catch(...)
155     {
156         std::cerr << "MusicPlayer::Remove: Unknown Exception occured" << std
157             ::endl;
158     }
159 }
160
161 void MusicPlayer::Search(std::string const& name)
162 {
163     try
164     {
165         if(name == "")
166         {
167             std::string error = "no valid name";
168             throw (error);
169         }
170
171         std::cout << "found medias: (search for \"" << name << "\")" << std::
172             endl;

```

```

169
170     SearchVisitor* searchVisitor = new SearchVisitor(name);
171
172     std::for_each(mMusicComponents.begin(), mMusicComponents.end(), [=] (
173         MusicComponent* m)
174     {
175         m->Accept(searchVisitor);
176     });
177
178     TMusicComponents* tmp;
179     tmp = searchVisitor->GetResults();
180
181     std::for_each(tmp->begin(), tmp->end(), [=] (MusicComponent* m)
182     {
183         std::cout << m->GetName() << std::endl;
184     });
185
186     delete searchVisitor; searchVisitor = 0;
187
188     catch (std::bad_alloc const& e)
189     {
190         std::cerr << e.what() << std::endl;
191     }
192
193     catch (std::string const& error)
194     {
195         std::cerr << "error in MusicPlayer::Search(): " << error << std::endl
196             << std::endl;
197     }
198
199     catch (...)
200     {
201         std::cerr << "MusicPlayer::Search: Unknown Exception occurred" << std
202             << std::endl;
203     }
204 }

```

```

1  //////////////////////////////////////
2  // Workfile : Main.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 17.11.2012
5  // Description : Testdriver for MusicPlayer
6  //////////////////////////////////////
7
8  #include <iostream>
9  #include "MusicPlayer.h"
10 #include "MusicFactory.h"
11 #include "MusicComponent.h"
12
13 using namespace std;
14
15
16 void EmptyTestCase()
17 {
18     cout << "Testcase0: Empty testcase with NULL pointer." << endl;
19
20     MusicPlayer* player = new MusicPlayer;
21     MusicFactory* fact = new MusicFactory;
22
23     cout << "Add: ";
24     player->Add(0);
25     cout << " ...done" << endl;
26
27     cout << "GetTime: ";
28     player->GetTime(0);
29     cout << " ...done" << endl;
30
31     cout << "GetTotalTime: ";
32     player->GetTotalTime();
33     cout << " ...done" << endl;
34
35     cout << "Play: ";
36     player->Play();
37     cout << " ...done" << endl;
38
39     cout << "Remove: ";
40     player->Remove(0);
41     cout << " ...done" << endl;
42
43     cout << "Search: ";
44     player->Search("");
45     cout << " ...done" << endl;
46
47     cout << "Delete MusicPlayer: ";
48     delete player; player=0;
49     cout << " ...done" << endl;
50
51     cout << "Delete MusicFactory: ";
52     delete fact; fact=0;
53     cout << " ...done" << endl;
54     cout << endl << endl;
55 }
56
57 void NormalTestCase()
58 {
59     cout << "Testcase1: Normal testcase with valid objects." << endl;
60

```



```

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: ";
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: ";
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