FH-OÖ Hagenberg/HSD
SDP3, WS 2025
Übung 3

Name: Simon Offenberger/ Simon Vogelhuber Aufwand in h: siehe Doku

Mat.Nr: S2410306027/ S2410306014 Punkte:

Übungsgruppe: 1 korrigiert:

Beispiel 1 (24 Punkte) Player-Schnittstelle: Sie verwenden in Ihrer Firma HSDSoft einen MusicPlayer von der Firma MonkeySoft. Die öffentliche Schnittstelle des MusicPlayers sieht folgendermaßen aus und kann nicht verändert werden:

```
1     //starts playing with the current song in list
2     void Start();
3     //stops playing
4     void Stop();
5     //switches to next song and starts at the end with first song
6     void SwitchNext();
7     //get index of current song
8     size_t const GetCurIndex() const;
9     //find a song by name in playlist
10    bool Find(std::string const& name);
11    //get count of songs in playlist
12    size_t const GetCount() const;
13    //increase the volume relative to the current volume
14    void IncreaseVol(size_t const vol);
15    //decrease the volume relative to the current volume
16    void DecreaseVol(size_t const vol);
17    //add a song to playlist
18    void Add(std::string const& name, size_t const dur);
```

Der MusicPlayer verwaltet Lieder und speichert den Namen und die Dauer jedes Liedes in Sekunden. Er kann gestartet und gestoppt werden und erlaubt das Verändern der Lautstärke. Die Lautstärke ist begrenzt mit 0 und maximal 100. Der Defaultwert für die Lautstärke liegt bei 15.

Zur Simulation liefert der Player je nach Aktion folgende Ausgaben auf der Konsole:

```
playing song number 1: Hells Bells (256 sec)
...
playing song number 4: Hawaguck (129 sec)
...
volume is now -> 70
song: Pulp Fiction not found!
stop song: Hells Bells (256 sec)
...
no song in playlist!
```

In weiterer Folge kaufen Sie einen VideoPlayer der Firma DonkeySoft mit folgender vorgegebenen Schnittstelle:

```
    //starts playing with the current song in list
    void Play() const;
    //stops playing
    void Stop() const;
    //switches to first video in playlist and returns true, otherwise false if list is empty.
    bool First();
    //switches to next video in playlist and returns true, otherwise false if last song is reached.
    bool Next();
    //returns index of current video
    size_t CurIndex() const;
    //returns name of current video
    std::string const CurVideo() const;
    //sets volume (min volume=0 and max volume=50)
    void SetVolume(size_t const vol);
    //gets current volume
    size_t const GetVolume() const;
    //adds a vidoe to playlist
    void Add(std::string const& name, size_t const dur, VideoFormat const& format);
}
```

Der VideoPlayer kann die Formate WMV, AVI und MKV abspielen. Er verwaltet Videos und speichert den Namen und die Dauer in Minuten. Er kann gestartet und gestoppt werden und erlaubt das Verändern der Lautstärke. Die Lautstärke ist begrenzt mit 0 und maximal 50. Der Defaultwert für die Lautstärke liegt bei 8.

Zur Simulation liefert der Player je nach Aktion folgende Ausgaben auf der Konsole:

```
playing video number 1: Die Sendung mit der Maus [duration -> 55 min], AVI-Format
...
playing video number 3: Freitag der 13te [duration -> 95 min], WMV-Format
...
volume is now -> 30
video: Hells Bells not found!
stop video: Pulp Fiction [duration -> 126 min], MKV-Format
...
no video in playlist!
```

Für einen Klienten soll nun nach außen folgende, unabhängige Schnittstelle zur Verfügung gestellt werden:

```
virtual void Play() = 0;
virtual void VolInc() = 0;
virtual void VolDec() = 0;
virtual void Stop() = 0;
virtual void Next() = 0;
virtual void Prev() = 0;
virtual void Select(std::string const& name) = 0;
```

Mit dieser Schnittstelle kann der Klient sowohl den MusicPlayer als auch den VideoPlayer verwenden. Die Methoden `VolInc()` und `VolDec()` erhöhen bzw. erniedrigen die Lautstärke um den Wert 1. `Next()` und `Prev()` schalten vor und zurück. `Select(...)` wählt ein Lied oder ein Video aus der Playliste aus.

Achten Sie beim Design auf die Einhaltung der Design-Prinzipien und verwenden Sie ein entsprechendes Design-Pattern!

Implementieren Sie alle notwendigen Klassen (auch die Music/VideoPlayer-Klassen) und testen Sie diese entsprechend!

Treffen Sie für alle unzureichenden Angaben sinnvolle Annahmen und begründen Sie diese. Verfassen Sie weiters eine Systemdokumentation (entsprechend den Vorgaben aus Übung1)!

Allgemeine Hinweise: Legen Sie bei der Erstellung Ihrer Übung großen Wert auf eine **saubere Strukturierung** und auf eine **sorgfältige Ausarbeitung!** Dokumentieren Sie alle Schnittstellen und versehen Sie Ihre Algorithmen an entscheidenden Stellen ausführlich mit Kommentaren! Testen Sie ihre Implementierungen ausführlich! Geben Sie den **Testoutput** mit ab!

Systemdokumentation
Projekt Music/VideoPlayer Adapter

Version 1.0

S. Offenberger, S. Vogelhuber

Hagenberg, 4. November 2025

Inhaltsverzeichnis

1 Organisatorisches

1.1 Team

- Simon Offenberger, Matr.-Nr.: S2410306027, E-Mail: S2410306027@fhooe.at
- Simon Vogelhuber, Matr.-Nr.: S2410306014, E-Mail: s2410306014@fhooe.at

1.2 Aufteilung der Verantwortlichkeitsbereiche

- Simon Offenberger
  - Design Klassendiagramm
  - Implementierung und Test der Klassen:
    - Client,
    - VideoPlayerAdapter,
    - VideoPlayer,
    - Video,
    - EVideoFormat,
  - Implementierung des Testtreibers
  - Dokumentation
- Simon Vogelhuber
  - Design Klassendiagramm
  - Implementierung und Komponententest der Klassen:
    - IPlayer
    - MusicPlayerAdapter,
    - MusicPlayer,
    - Song

  - Implementierung des Testtreibers
  - Dokumentation

1.3 Aufwand

- Simon Offenberger: geschätzt 12 Ph / tatsächlich 11 Ph
- Simon Vogelhuber: geschätzt 9 Ph / tatsächlich 9 Ph

2 Anforderungsdefinition (Systemspezifikation)

Für die Implementierung wurden die Header von MusicPlayer, VideoPlayer und IPlayer Interface vorgegeben. Die Anforderung bestand darin einen Client eine gemeinsame Schnittstelle zum Ansprechen von MusicPlayer sowie VideoPlayer zu bieten. Die Schnittstelle soll folgende Funktionen bereitstellen.

2.1 IPlayer Interface Anforderung

- Play
  - Spielt das Video bzw. den Song des entsprechenden Players -> Ausgabe auf COUT
- VolInc
  - Diese Methode soll die Lautstärke des Players um 1 erhöhen.
- VolDec
  - Diese Methode soll die Lautstärke des Players um 1 verringern.
- Stop
  - Stoppt die Wiedergabe
- Next
  - Wechselt den aktuellen Titel auf den nächsten in der Liste
- Next
  - Wechselt den aktuellen Titel auf den vorherigen in der Liste
- Select
  - Wählt einen Titel über den Namen aus

2.2 VideoPlayer Anforderung

Folgende Anforderungen müssen die Methoden des VideoPlayers bereitstellen:

- Play
  - Spielt das Video ab -> Ausgabe auf COUT
- Stop
  - Stopt das Video -> Ausgabe auf COUT
- First
  - Wechsel auf den ersten Titel in der Playlist
  - gibt true zurück wenn dies erfogreich ist
  - gibt false wenn kein Titel in der Playlist ist
- Next
  - Wechsel auf den nächsten Titel in der Playlist
  - gibt true zurück wenn dies erfogreich ist
  - gibt false wenn kein weiterer Titel in der Playlist ist
- CurIndex
  - Liefert den aktuellen Index der Playlist
- CurVideo
  - Liefert den aktuellen Title als string
- SetVolume
  - Setzt die Lautstärke des Titles max 50 min 0
- GetVolume
  - Liefert die aktuelle Lautstärke
- Add
  - Fügt und erzeugt ein Video an die Playlist hinten an

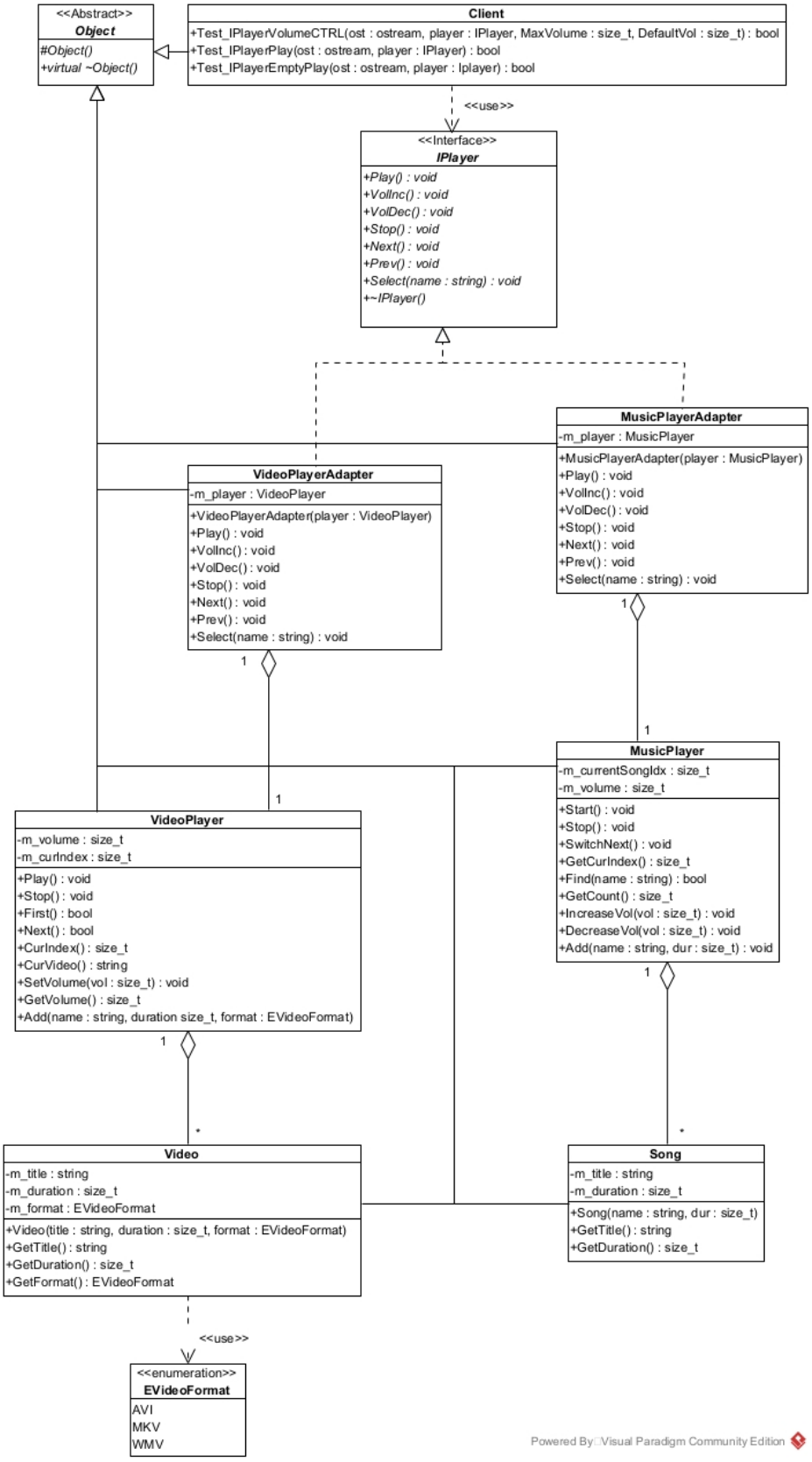
2.3 VideoPlayer Anforderung

Folgende Anforderungen müssen die Methoden des MusicPlayers bereitstellen:

- Start
  - Spielt den Song ab -> Ausgabe auf COUT
- Stop
  - Stopt den Song -> Ausgabe auf COUT
- SwitchNext
  - Wechsel auf den nächsten Titel in der Playlist am Ende wird mit den ersten fortgesetzt
- GetCurIndex
  - Liefert den aktuellen Index der Playlist
- Find
  - Sucht nach einem Titel und wählt ihn aus
  - gibt true wenn Titel gefunden wurde
  - gibt false wenn Titel nicht gefunden wurde
- GetCount
  - Gibt die Anzahl der Lieder in der Playlist zurück
- IncreaseVol
  - erhöht die Lautstärke um einen bestimmten Wert (max 100)
- DecreaseVol
  - reduziert die Lautstärke um einen bestimmten Wert (min 0)
- Add
  - Fügt und erzeugt ein Video an die Playlist hinten an

3 Systementwurf

3.1 Klassendiagramm



3.2 Designentscheidungen

Die Klassen Video und Song wurden so umgesetzt, dass diese für die Speicherung der spezifischen Daten eingesetzt werden. Hier wird in den Playerklassen ein Container von Videos bzw. Songs gespeichert. Für die Bereitstellung eines gemeinsamen Interfaces für den Client wurden Adapter für den Music- bzw. Video Player implementiert. Dieser Adapter speichern intern nur eine Referenz auf den tatsächlichen Players. Dies ermöglicht es den Player selbst als auch den Adapter simultan zu verwenden. Im Adapter mussten die Funktion der Player so angewandt und kombiniert werden, dass für beide Player über das Interface die selbe Funktionalität zur Verfügung steht.

Die gemeinsamen Funktionen des Interfaces wurde im Client getestet. Alle anderen Klassen wurden im main getestet.

In der Übung wurde nachgefragt ob die starre Ausgabe auf cout, über einen Parameter in der Methode ausgetauscht werden kann, aber nach Absprache mit Herrn Wiesinger dürfen keine Veränderungen vorgenommen werden. Somit musste im Testtreiber cout umgeleitet werden um einen sinnvollen Testtreiber zu schreiben.

4 Dokumentation der Komponenten (Klassen)

Die HTML-Startdatei befindet sich im Verzeichnis ./../doxy/html/index.html

5 Testprotokollierung

```
Test VideoPlayer Adapter in Client

****************************************
             TESTCASE START
****************************************

Test Volume Inc
[Test OK] Result: (Expected: true == Result: true)

Test Volume Dec
[Test OK] Result: (Expected: true == Result: true)

Test Lower Bound Volume 0
[Test OK] Result: (Expected: true == Result: true)

Test Upper Bound Volume
[Test OK] Result: (Expected: true == Result: true)

Test for Exceotion in Test Case
[Test OK] Result: (Expected: true == Result: true)


****************************************


****************************************
             TESTCASE START
****************************************

Test Play Contains Name
[Test OK] Result: (Expected: true == Result: true)

Test Next
[Test OK] Result: (Expected: true == Result: true)

Test Next
[Test OK] Result: (Expected: true == Result: true)

Test Next
[Test OK] Result: (Expected: true == Result: true)

Test Next
[Test OK] Result: (Expected: true == Result: true)

Test Next
[Test OK] Result: (Expected: true == Result: true)
```

```
47
48 Test Next Wrap around
49 [Test OK] Result: (Expected: true == Result: true)
50
51 Test Select Video by name
52 [Test OK] Result: (Expected: true == Result: true)
53
54 Test Select Video by name not found
55 [Test OK] Result: (Expected: true == Result: true)
56
57 Test Stop Player
58 [Test OK] Result: (Expected: true == Result: true)
59
60 Test for Exception in Test Case
61 [Test OK] Result: (Expected: true == Result: true)
62
63
64 ****************************************
65
66
67 ****************************************
68              TESTCASE START
69 ****************************************
70
71 Test for Message in Empty Player
72 [Test OK] Result: (Expected: true == Result: true)
73
74 Test for Exception in Testcase
75 [Test OK] Result: (Expected: true == Result: true)
76
77
78 ****************************************
79
80 Test MusicPlayer Adapter in Client
81
82 ****************************************
83              TESTCASE START
84 ****************************************
85
86 Test Volume Inc
87 [Test OK] Result: (Expected: true == Result: true)
88
89 Test Volume Dec
90 [Test OK] Result: (Expected: true == Result: true)
91
92 Test Lower Bound Volume 0
93 [Test OK] Result: (Expected: true == Result: true)
94
95 Test Upper Bound Volume
```

```
[Test OK] Result: (Expected: true == Result: true)

Test for Exceotion in Test Case
[Test OK] Result: (Expected: true == Result: true)


*****************************************


*****************************************
             TESTCASE START
*****************************************

Test Play Contains Name
[Test OK] Result: (Expected: true == Result: true)

Test Next
[Test OK] Result: (Expected: true == Result: true)

Test Next
[Test OK] Result: (Expected: true == Result: true)

Test Next
[Test OK] Result: (Expected: true == Result: true)

Test Next
[Test OK] Result: (Expected: true == Result: true)

Test Next
[Test OK] Result: (Expected: true == Result: true)

Test Next Wrap around
[Test OK] Result: (Expected: true == Result: true)

Test Select Video by name
[Test OK] Result: (Expected: true == Result: true)

Test Select Video by name not found
[Test OK] Result: (Expected: true == Result: true)

Test Stop Player
[Test OK] Result: (Expected: true == Result: true)

Test for Exception in Test Case
[Test OK] Result: (Expected: true == Result: true)


*****************************************

```

```
145
146 ****************************************
147              TESTCASE START
148 ****************************************
149
150 Test for Message in Empty Player
151 [Test OK] Result: (Expected: true == Result: true)
152
153 Test for Exception in Testcase
154 [Test OK] Result: (Expected: true == Result: true)
155
156
157 ****************************************
158
159
160 ****************************************
161              TESTCASE START
162 ****************************************
163
164 Test Song Getter Duration
165 [Test OK] Result: (Expected: 123 == Result: 123)
166
167 Test Song Getter Name
168 [Test OK] Result: (Expected: Hello World == Result: Hello World)
169
170 Check for Exception in Testcase
171 [Test OK] Result: (Expected: true == Result: true)
172
173 Test Exception in Song CTOR with duration 0
174 [Test OK] Result: (Expected: ERROR: Song with duration 0! == Result:
    ↪ ERROR: Song with duration 0!)
175
176 Test Exception in Song CTOR with empty string
177 [Test OK] Result: (Expected: ERROR: Song with empty Name! == Result:
    ↪ ERROR: Song with empty Name!)
178
179
180 ****************************************
181
182
183 ****************************************
184              TESTCASE START
185 ****************************************
186
187 Test Song Getter Duration
188 [Test OK] Result: (Expected: 123 == Result: 123)
189
190 Test Song Getter Name
191 [Test OK] Result: (Expected: Hello World == Result: Hello World)
```

```
192
193 Test Song Getter Format
194 [Test OK] Result: (Expected: AVI-Format == Result: AVI-Format)
195
196 Check for Exception in Testcase
197 [Test OK] Result: (Expected: true == Result: true)
198
199 Test Exception in Video CTOR with duration 0
200 [Test OK] Result: (Expected: ERROR: Video with duration 0! == Result:
    ↪  ERROR: Video with duration 0!)
201
202 Test Exception in Video CTOR with empty string
203 [Test OK] Result: (Expected: ERROR: Video with empty Name! == Result:
    ↪  ERROR: Video with empty Name!)
204
205
206 ****************************************
207
208
209 ****************************************
210              TESTCASE START
211 ****************************************
212
213 Test Videplayer Initial Index
214 [Test OK] Result: (Expected: 0 == Result: 0)
215
216 Test Videplayer Index after First
217 [Test OK] Result: (Expected: 0 == Result: 0)
218
219 Test Videplayer Index after Next
220 [Test OK] Result: (Expected: 1 == Result: 1)
221
222 Test Videplayer Index Upper Bound
223 [Test OK] Result: (Expected: 4 == Result: 4)
224
225 Test Videplayer Index after First
226 [Test OK] Result: (Expected: 0 == Result: 0)
227
228 Test Default Volume
229 [Test OK] Result: (Expected: 8 == Result: 8)
230
231 Test Set Volume
232 [Test OK] Result: (Expected: 25 == Result: 25)
233
234 Test Set Volume Max Volume
235 [Test OK] Result: (Expected: 50 == Result: 50)
236
237 Test Set Volume Min Volume
238 [Test OK] Result: (Expected: 0 == Result: 0)
```

```
239
240 Test Video Player Play
241 [Test OK] Result: (Expected: true == Result: true)
242
243 Test Video Player Stop
244 [Test OK] Result: (Expected: true == Result: true)
245
246 Check for Exception in Testcase
247 [Test OK] Result: (Expected: true == Result: true)
248
249 Test Exception in Add with empty string
250 [Test OK] Result: (Expected: ERROR: Video with empty Name! == Result:
    ↪  ERROR: Video with empty Name!)
251
252 Test Exception in Add with empty string
253 [Test OK] Result: (Expected: ERROR: Video with duration 0! == Result:
    ↪  ERROR: Video with duration 0!)
254
255
256 ****************************************
257
258
259 ****************************************
260              TESTCASE START
261 ****************************************
262
263 MusicPlayer - Basic Functionality - .GetCount()
264 [Test OK] Result: (Expected: 4 == Result: 4)
265
266 MusicPlayer - Basic Functionality - .GetIndex() initial
267 [Test OK] Result: (Expected: 0 == Result: 0)
268
269 MusicPlayer - Basic Functionality - .Find() unknown song
270 [Test OK] Result: (Expected: false == Result: false)
271
272 MusicPlayer - Basic Functionality - .Find() song that exists
273 [Test OK] Result: (Expected: true == Result: true)
274
275 MusicPlayer - Basic Functionality - Song name after initial .Start()
276 [Test OK] Result: (Expected: true == Result: true)
277
278 MusicPlayer - Basic Functionality - .GetIndex() after switching
279 [Test OK] Result: (Expected: 1 == Result: 1)
280
281 MusicPlayer - Basic Functionality - Song name switching
282 [Test OK] Result: (Expected: true == Result: true)
283
284 MusicPlayer - Basic Functionality - .GetIndex() wrap around
285 [Test OK] Result: (Expected: 1 == Result: 1)
```

```
286
287 MusicPlayer - Basic Functionality - Error Buffer
288 [Test OK] Result: (Expected: true == Result: true)
289
290 MusicPlayer - Add Song without title
291 [Test OK] Result: (Expected: ERROR: Song with empty Name! == Result:
    ↪ ERROR: Song with empty Name!)
292
293 MusicPlayer - Add Song without title
294 [Test OK] Result: (Expected: ERROR: Song with duration 0! == Result:
    ↪ ERROR: Song with duration 0!)
295
296 MusicPlayer - Add Song without title
297 [Test OK] Result: (Expected: ERROR: Song with empty Name! == Result:
    ↪ ERROR: Song with empty Name!)
298
299 TEST OK!!
```

6 Quellcode

6.1 Object.hpp

```
1  /*****************************************************************//**
2   * \file   Object.hpp
3   * \brief  common ancestor for all objects
4   *
5   * \author Simon
6   * \date   November 2025
7   *********************************************************************/
8  #ifndef OBJECT_HPP
9  #define OBJECT_HPP
10
11 #include <string>
12
13 class Object {
14 public:
15
16         // Exceptions constants
17         inline static const std::string ERROR_BAD_OSTREAM = "ERROR: Provided Ostream is bad";
18         inline static const std::string ERROR_FAIL_WRITE = "ERROR: Fail to write on provided Ostream";
19         inline static const std::string ERROR_NULLPTR = "ERROR: Passed in Nullptr!";
20
21         // once virtual always virtual
22         virtual ~Object() = default;
23
24
25 protected:
26         Object() = default;
27 };
28
29 #endif // !OBJECT_HPP
```

6.2 Client.hpp

```cpp
/*****************************************************************//**
 * \file   Client.hpp
 * \brief  Client Class that uses a IPlayer Interface inorder to control
 * \brief  a Musicplayer or a Videoplayer via their adapter
 *
 * \author Simon
 * \date   November 2025
 *********************************************************************/
#ifndef CLIENT_HPP
#define CLIENT_HPP

#include "Object.hpp"
#include "IPlayer.hpp"
#include <iostream>

class Client : public Object
{
public:
        /**
         * \brief Test Function for the Volume Control of the IPlayer interface.
         *
         * \param ost Ostream
         * \param player Reference to the player
         * \param MaxVolume Maximum Volume of the player
         * \param DefaultVol Default Volume of the player
         * \return true -> tests OK
         * \return false -> tests failed
         */
        bool Test_IPlayerVolumeCTRL(std::ostream& ost, IPlayer& player, const size_t& MaxVolume, const size_t&
             DefaultVol) const;

        /**
         * \brief Test Play of the Player.
         *
         * \param ost Ostream for the Testoutput
         * \param player Refernce to player
         * \return true -> tests OK
         * \return false -> tests failed
         */
        bool Test_IPlayerPlay(std::ostream& ost, IPlayer& player) const;

        /**
         * \brief Test Play of an empty Player.
         *
         * \param ost Ostream for the Testoutput
         * \param player Refernce to player
         * \return true -> tests OK
         * \return false -> tests failed
         */
        bool Test_IPlayerEmptyPlay(std::ostream& ost, IPlayer& player) const;
};

#endif // !CLIENT_HPP
```

6.3 Client.cpp

```cpp
/*****************************************************************//**
 * \file   Client.cpp
 * \brief  Client Class that uses a IPlayer Interface inorder to control
 * \brief  a Musicplayer or a Videoplayer via their adapter
 *
 * \author Simon
 * \date   November 2025
 *********************************************************************/
#include "Client.hpp"
#include "Test.hpp"
#include <sstream>
#include <algorithm>

using namespace std;

bool Client::Test_IPlayerVolumeCTRL(std::ostream& ost, IPlayer& player, const size_t & MaxVolume, const size_t &
      DefaultVol) const
{
        if (!ost.good()) throw Client::ERROR_BAD_OSTREAM;

        TestStart(ost);

        bool TestOK = true;
        string error_msg = "";

        try {

                stringstream result;

                std::streambuf* coutbuf = std::cout.rdbuf();

                result << DefaultVol+1;
                string DVol;

                result >> DVol;

                result.clear();
                result.str("");

                // cout redirect to stringstream
                std::cout.rdbuf(result.rdbuf());

                player.VollInc();

                std::cout.rdbuf(coutbuf);

                TestOK == TestOK && check_dump(ost, "Test␣Volume␣Inc", true, result.str().find(DVol)!=std::string::
                      npos);

                result.clear();
                result.str("");

                result << DefaultVol;

                result >> DVol;

                result.clear();
                result.str("");

                // cout redirect to stringstream
                std::cout.rdbuf(result.rdbuf());

                player.VollDec();

                std::cout.rdbuf(coutbuf);

                TestOK == TestOK && check_dump(ost, "Test␣Volume␣Dec", true, result.str().find(DVol)!=std::string::
                      npos);

                // cout redirect to stringstream
                std::cout.rdbuf(result.rdbuf());

                for (int i = 0;i < 200; i++) player.VollDec();

                player.VollInc();

                std::cout.rdbuf(coutbuf);

                result.clear();
                result.str("");
```

```
78
79
80                  // cout redirect to stringstream
81                  std::cout.rdbuf(result.rdbuf());
82
83                  player.VollDec();
84
85                  std::cout.rdbuf(coutbuf);
86
87                  TestOK == TestOK && check_dump(ost, "Test␣Lower␣Bound␣Volume␣0", true, result.str().find("0") != std
                        ::string::npos);
88
89                  // cout redirect to stringstream
90                  std::cout.rdbuf(result.rdbuf());
91
92                  for (int i = 0;i < 200; i++) player.VollInc();
93
94                  std::cout.rdbuf(coutbuf);
95
96                  result.clear();
97                  result.str("");
98
99                  result << MaxVolume;
100
101                 string MaxVol;
102
103                 result >> MaxVol;
104
105                 result.clear();
106                 result.str("");
107
108                 // cout redirect to stringstream
109                 std::cout.rdbuf(result.rdbuf());
110
111                 player.VollInc();
112
113                 std::cout.rdbuf(coutbuf);
114
115
116                 TestOK == TestOK && check_dump(ost, "Test␣Upper␣Bound␣Volume", true, result.str().find(MaxVol) !=
                        std::string::npos);
117         }
118         catch (const string& err) {
119                 error_msg = err;
120                 TestOK = false;
121         }
122         catch (bad_alloc const& error) {
123                 error_msg = error.what();
124                 TestOK = false;
125         }
126         catch (const exception& err) {
127                 error_msg = err.what();
128                 TestOK = false;
129         }
130         catch (...) {
131                 error_msg = "Unhandelt␣Exception";
132                 TestOK = false;
133         }
134
135         TestOK == TestOK && check_dump(ost, "Test␣for␣Exceotion␣in␣Test␣Case", true,error_msg.empty());
136
137         TestEnd(ost);
138
139         if (ost.fail()) throw Client::ERROR_FAIL_WRITE;
140
141         return TestOK;
142 }
143
144 bool Client::Test_IPlayerPlay(std::ostream& ost, IPlayer& player) const
145 {
146         if (!ost.good()) throw Client::ERROR_BAD_OSTREAM;
147
148         TestStart(ost);
149
150         bool TestOK = true;
151         string error_msg = "";
152
153         try {
154
155                 stringstream result;
156                 std::streambuf* coutbuf = std::cout.rdbuf();
157
158                 // cout redirect to stringstream
```

```
159                 std::cout.rdbuf(result.rdbuf());
160
161                 player.Play();
162
163                 std::cout.rdbuf(coutbuf);
164
165                 TestOK == TestOK && check_dump(ost, "Test␣Play␣Contains␣Name", true, result.str().find("Harry␣
                          Potter1") != std::string::npos);
166
167                 player.Next();
168
169                 result.str("");
170                 result.clear();
171
172                 std::cout.rdbuf(result.rdbuf());
173
174                 player.Play();
175
176                 std::cout.rdbuf(coutbuf);
177
178                 TestOK == TestOK && check_dump(ost, "Test␣Next␣", true, result.str().find("Harry␣Potter2") != std::
                          string::npos);
179
180                 for (int i = 0; i < 4; i++) {
181
182                         player.Next();
183
184                         result.str("");
185                         result.clear();
186
187                         std::cout.rdbuf(result.rdbuf());
188
189                         player.Play();
190
191                         std::cout.rdbuf(coutbuf);
192
193                         TestOK == TestOK && check_dump(ost, "Test␣Next␣", true, result.str().find("Harry␣Potter" + 2
                                + i) != std::string::npos);
194
195                 }
196
197                 player.Next();
198
199                 result.str("");
200                 result.clear();
201
202                 std::cout.rdbuf(result.rdbuf());
203
204                 player.Play();
205
206                 std::cout.rdbuf(coutbuf);
207
208                 TestOK == TestOK && check_dump(ost, "Test␣Next␣Wrap␣around", true, result.str().find("Harry␣Potter1"
                          ) != std::string::npos);
209
210                 result.str("");
211                 result.clear();
212
213                 std::cout.rdbuf(result.rdbuf());
214
215                 player.Select("Harry␣Potter3");
216                 player.Play();
217
218                 std::cout.rdbuf(coutbuf);
219
220                 TestOK == TestOK && check_dump(ost, "Test␣Select␣Video␣by␣name␣", true, result.str().find("Harry␣
                          Potter3") != std::string::npos);
221
222                 result.str("");
223                 result.clear();
224
225                 std::cout.rdbuf(result.rdbuf());
226
227                 player.Select("Harry␣Potter14");
228                 player.Play();
229
230                 std::cout.rdbuf(coutbuf);
231
232                 TestOK == TestOK && check_dump(ost, "Test␣Select␣Video␣by␣name␣not␣found", true, result.str().find("
                          not␣found!") != std::string::npos);
233
234                 result.str("");
235                 result.clear();
```

```
236
237                 std::cout.rdbuf(result.rdbuf());
238
239                 player.Select("Harry␣Potter3");
240                 player.Stop();
241
242                 std::cout.rdbuf(coutbuf);
243
244                 TestOK == TestOK && check_dump(ost, "Test␣Stop␣Player",
245                                 true,
246                                 result.str().find("stop") != std::string::npos && result.str().find("Harry␣Potter3")
                                        != std::string::npos);
247
248
249         }
250         catch (const string& err) {
251                 error_msg = err;
252                 TestOK = false;
253         }
254         catch (bad_alloc const& error) {
255                 error_msg = error.what();
256                 TestOK = false;
257         }
258         catch (const exception& err) {
259                 error_msg = err.what();
260                 TestOK = false;
261         }
262         catch (...) {
263                 error_msg = "Unhandelt␣Exception";
264                 TestOK = false;
265         }
266
267         TestOK == TestOK && check_dump(ost, "Test␣for␣Exception␣in␣Test␣Case", true, error_msg.empty());
268
269         TestEnd(ost);
270
271         if (ost.fail()) throw Client::ERROR_FAIL_WRITE;
272
273         return TestOK;
274 }
275
276 bool Client::Test_IPlayerEmptyPlay(std::ostream& ost, IPlayer& player) const
277 {
278         if (!ost.good()) throw Client::ERROR_BAD_OSTREAM;
279
280         TestStart(ost);
281
282         bool TestOK = true;
283         string error_msg = "";
284
285         try {
286                 stringstream result;
287
288                 result.str("");
289                 result.clear();
290
291                 std::streambuf* coutbuf = std::cout.rdbuf();
292
293                 std::cout.rdbuf(result.rdbuf());
294
295                 player.Play();
296
297                 std::cout.rdbuf(coutbuf);
298
299                 TestOK == TestOK && check_dump(ost, "Test␣for␣Message␣in␣Empty␣Player", true, result.str().find("no"
                        )!=string::npos);
300
301         }
302         catch (const string& err) {
303                 error_msg = err;
304         }
305         catch (bad_alloc const& error) {
306                 error_msg = error.what();
307         }
308         catch (const exception& err) {
309                 error_msg = err.what();
310         }
311         catch (...) {
312                 error_msg = "Unhandelt␣Exception";
313         }
314
315         TestOK == TestOK && check_dump(ost, "Test␣for␣Exception␣in␣Testcase",true , error_msg.empty());
316
```

```
317         TestEnd(ost);
318
319         if (ost.fail()) throw Client::ERROR_FAIL_WRITE;
320
321         return TestOK;
322 }
```

6.4 IPlayer.hpp

```cpp
/*****************************************************************//**
 * \file   IPlayer.hpp
 * \brief  Interface to interact with various Player (music, video)
 * \author Simon Vogelhuber
 * \date   October 2025
 *********************************************************************/
#ifndef IPLAYER_HPP
#define IPLAYER_HPP

#include <string>

class IPlayer
{
public:
    /**
     * \brief Play selected song
     */
    virtual void Play() = 0;

    /**
     * \brief increase volume by 1 (out of 100)
     */
    virtual void VollInc() = 0;

    /**
     * \brief decrease volume by 1 (out of 100)
     */
    virtual void VollDec() = 0;

    /**
     * \brief Stop playing Song
     */
    virtual void Stop() = 0;

    /**
     * \brief Skip to next song
     */
    virtual void Next() = 0;

    /**
     * \brief Skip to previous song
     */
    virtual void Prev() = 0;

    /**
     * \brief Selects a Video by Name.
     *
     * \param name
     */
    virtual void Select(std::string const& name) = 0;

    /**
     * \brief virtual Destructor for Interface.
     *
     */
    virtual ~IPlayer() = default;

};


#endif // !IPLAYER_HPP
```

6.5 MusicPlayerAdapter.hpp

```cpp
/*****************************************************************//**
 * \file   MusicPlayerAdapter.hpp
 * \brief  Adapter for a Musicplayer to comply with Interface IPlayer
 *
 * \author Simon
 * \date   November 2025
 *********************************************************************/
#ifndef MUSIC_PLAYER_ADAPTER_HPP
#define MUSIC_PLAYER_ADAPTER_HPP

#include "IPlayer.hpp"
#include "MusicPlayer.hpp"

class MusicPlayerAdapter :public Object, public IPlayer
{
public:

    /**
     * \brief Ctor for Adapter.
     *
     * \param player Reference to a MusicPlayer
     */
    MusicPlayerAdapter(MusicPlayer & player) : m_player{ player } {}

    /**
     * \brief Play selected song
     */
    virtual void Play() override;

    /**
     * \brief increase volume by 1 (out of 100)
     */
    virtual void VollInc() override;

    /**
     * \brief decrease volume by 1 (out of 100)
     */
    virtual void VollDec() override;

    /**
     * \brief Stop playing Song
     */
    virtual void Stop() override;

    /**
     * \Skip to next song
     */
    virtual void Next() override;

    /**
     * \brief Skip to previous song
     */
    virtual void Prev() override;

    /**
     * \brief Selects a Video by Name.
     *
     * \param name
     */
    virtual void Select(std::string const& name) override;

    // delete Copy Ctor and Assign Operator to prohibit untestet behaviour
    MusicPlayerAdapter(MusicPlayerAdapter& Music) = delete;
    void operator=(MusicPlayerAdapter Music) = delete;

private:
    MusicPlayer & m_player;
};

#endif // !MUSIC_PLAYER_ADAPTER_HPP
```

6.6 MusicPlayerAdapter.cpp

```cpp
/*****************************************************************//**
 * \file   MusicPlayerAdapter.cpp
 * \brief  Adapter for a Musicplayer to comply with Interface IPlayer
 *
 *
 * \author Simon
 * \date   November 2025
 *********************************************************************/

#include "MusicPlayerAdapter.hpp"
#include <iostream>

void MusicPlayerAdapter::Play()
{
    m_player.Start();
}

void MusicPlayerAdapter::VollInc()
{
    m_player.IncreaseVol(1);
}

void MusicPlayerAdapter::VollDec()
{
    m_player.DecreaseVol(1);
}

void MusicPlayerAdapter::Stop()
{
    m_player.Stop();
}

void MusicPlayerAdapter::Next()
{
    m_player.SwitchNext();
}

void MusicPlayerAdapter::Prev()
{
    // The MusicPlayer does not provide a prevSong
    // function - so we need to skip forward until
    // we hit the previus song.
    size_t skipSongs = m_player.GetCount() - 1;

    for (int i = 0; i < skipSongs; i++)
        m_player.SwitchNext();
}

void MusicPlayerAdapter::Select(std::string const& name)
{
    if (!m_player.Find(name)) std::cout << "song: " << name << " not found!" << std::endl;
}
```

6.7 MusicPlayer.hpp

```cpp
/*****************************************************************//**
 * \file   MusicPlayer.hpp
 * \brief  MusicPlayer - A player for music!
 * \author Simon Vogelhuber
 * \date   October 2025
 *********************************************************************/
#ifndef MUSIC_PLAYER_HPP
#define MUSIC_PLAYER_HPP

#include "Object.hpp"
#include "Song.hpp"
#include <vector>

using  SongCollection = std::vector<Song>;

class MusicPlayer : public Object
{
public:
    // Exception constants
    inline static const std::string ERROR_DURATION_NULL = "ERROR: Song with duration 0!";
    inline static const std::string ERROR_EMPTY_NAME = "ERROR: Song with empty Name!";

        inline static const std::size_t MAX_VOLUME = 100;
        inline static const std::size_t MIN_VOLUME = 0;
    inline static const std::size_t DEFAULT_VOLUME = 50;

    /**
     * Default CTOR.
     *
     */
    MusicPlayer() = default;

    /**
     * \brief Plays selected song
     */
    void Start();

    /**
     * \brief Stop playing Song
     */
    void Stop();

    /**
     * \brief Skip to next song
     */
    void SwitchNext();

    /**
     * \brief Get index of current song
     * \return size_t of current's song index
     */
    size_t const GetCurIndex() const;
    /**
     * \brief Find song by name and selcet it
     * \param string name name of the Song
     * \return true if song by that name exists
     */
    bool Find(std::string const& name);

    /**
     * \brief Get No. Songs inside the player
     * \return size_t count of songs inside player
     */
    size_t const GetCount() const;

    /**
     * \brief Increase volume by 'vol' amount
     * \param size_t vol (volume)
     */
    void IncreaseVol(size_t const vol);

    /**
     * \brief Decrease volume by 'vol' amount
     * \param size_t vol (volume)
     */
    void DecreaseVol(size_t const vol);

    /**
     * \brief Add song to player
     * \param string name
```

```
81          * \param size_t dur (duration)
82          */
83         void Add(std::string const& name, size_t const dur);
84
85         // delete Copy Ctor and Assign Operator to prohibit untestet behaviour
86         MusicPlayer(MusicPlayer& Music) = delete;
87         void operator=(MusicPlayer Music) = delete;
88
89     private:
90         SongCollection m_songs;
91         size_t m_currentSongIdx = 0;
92         size_t m_volume = DEFAULT_VOLUME;
93     };
94
95
96     #endif // !MUSIC_PLAYER_HPP
```

6.8 MusicPlayer.cpp

```cpp
/*****************************************************************//**
 * \file   MusicPlayer.cpp
 * \brief  MusicPlayer - A player for music!
 * \author Simon Vogelhuber
 * \date   October 2025
 *********************************************************************/
#include "MusicPlayer.hpp"
#include <iostream>

void MusicPlayer::Start()
{
    if (std::cout.bad())  throw Object::ERROR_BAD_OSTREAM;

    if (m_songs.empty())
    {
        std::cout << "no songs in playlist!" << std::endl;
        return;
    }

    std::cout
        << "playing song number " << m_currentSongIdx << ": "
        << m_songs.at(m_currentSongIdx).GetTitle()
        << " (" << m_songs.at(m_currentSongIdx).GetDuration() << ")\n";
}

void MusicPlayer::Stop()
{
    if (std::cout.bad())
        throw Object::ERROR_BAD_OSTREAM;

    std::cout
        << "stop song number " << m_currentSongIdx << ": "
        << m_songs.at(m_currentSongIdx).GetTitle()
        << " (" << m_songs.at(m_currentSongIdx).GetDuration() << ")\n";
}

void MusicPlayer::SwitchNext()
{
    // increase until end then wrap around
    m_currentSongIdx = (m_currentSongIdx + 1) % m_songs.size();
}

size_t const MusicPlayer::GetCurIndex() const
{
    return m_currentSongIdx;
}

bool MusicPlayer::Find(std::string const& name)
{
    if (name.empty()) throw MusicPlayer::ERROR_EMPTY_NAME;

    for (int i =0; i<m_songs.size(); i++)
    {
        if (m_songs.at(i).GetTitle() == name) {
            m_currentSongIdx = i;
            return true;
        }
    }
    return false;
}

size_t const MusicPlayer::GetCount() const
{
    return m_songs.size();
}

void MusicPlayer::IncreaseVol(size_t const vol)
{
    if (std::cout.bad())
        throw Object::ERROR_BAD_OSTREAM;

    m_volume += vol;
    if (m_volume > MAX_VOLUME)
        m_volume = MAX_VOLUME;

    std::cout << "volume is now -> " << m_volume << std::endl;
}

void MusicPlayer::DecreaseVol(size_t const vol)
{
```

```
81      if (std::cout.bad())
82          throw Object::ERROR_BAD_OSTREAM;
83
84      if (vol > m_volume)
85          m_volume = MIN_VOLUME;
86      else
87          m_volume -= vol;
88
89      std::cout << "volume is now -> " << m_volume << std::endl;
90  }
91
92  void MusicPlayer::Add(std::string const& name, size_t const dur)
93  {
94      if (name.empty()) throw MusicPlayer::ERROR_EMPTY_NAME;
95      if (dur == 0)     throw MusicPlayer::ERROR_DURATION_NULL;
96
97      m_songs.emplace_back(name, dur);
98  }
```

6.9 Song.hpp

```cpp
/*****************************************************************//**
 * \file   Song.hpp
 * \brief  Atomic Class for saving information about a song
 * \author Simon Vogelhuber
 * \date   October 2025
 *********************************************************************/
#ifndef SONG_HPP
#define SONG_HPP

#include "Object.hpp"

class Song : public Object
{
public:

    // Exceptions
    inline static const std::string ERROR_DURATION_NULL = "ERROR: Song with duration 0!";
    inline static const std::string ERROR_EMPTY_NAME = "ERROR: Song with empty Name!";

    Song(const std::string& name, const size_t& dur);
    /**
     * \brief Get title of song
     * \return string - title of song
     * \throw ERROR_DURATION_NULL
     * \throw ERROR_EMPTY_NAME
     */
    std::string const& GetTitle() const;

    /**
     * \brief Get duration of song
     * \return size_t - duratoin of song
     */
    size_t const GetDuration() const;
private:
    std::string  m_name;
    size_t  m_duration;
};

#endif // !SONG_HPP
```

6.10 Song.cpp

```cpp
/*****************************************************************//**
 * \file   Song.cpp
 * \brief  Atomic Class for saving information about a song
 * \author Simon Vogelhuber
 * \date   October 2025
 *********************************************************************/

#include "Song.hpp"

Song::Song(const std::string& name, const size_t& dur)
{
    if (name.empty()) throw Song::ERROR_EMPTY_NAME;
    if (dur == 0)      throw Song::ERROR_DURATION_NULL;

    m_name = name;
    m_duration = dur;

}

std::string const& Song::GetTitle() const
{
    return m_name;
}

size_t const Song::GetDuration() const
{
    return m_duration;
}
```

6.11 VideoPlayerAdapter.hpp

```cpp
/*****************************************************************//**
 * \file   VideoPlayerAdapter.hpp
 * \brief  Adapter for the Video Player in order to Implement IPlayer Interface
 *
 * \author Simon
 * \date   November 2025
 *********************************************************************/
#ifndef VIDEO_PLAYER_ADAPTER_HPP
#define VIDEO_PLAYER_ADAPTER_HPP

#include "IPlayer.hpp"
#include "VideoPlayer.hpp"

class VideoPlayerAdapter : public Object, public IPlayer
{
public:

    /**
     * \brief Construct a VideoPlayer Adapter .
     *
     * \param VidPlayer Reference to the actual VideoPlayer
     */
    VideoPlayerAdapter(VideoPlayer & VidPlayer) : m_player(VidPlayer) {}

    /**
     * \brief Play selected song
     */
    virtual void Play() override;

    /**
     * \brief increase volume by 1
     */
    virtual void VollInc() override;

    /**
     * \brief decrease volume by 1
     */
    virtual void VollDec() override;

    /**
     * \brief Stop playing Song
     */
    virtual void Stop() override;

    /**
     * \Skip to next song
     */
    virtual void Next() override;

    /**
     * \brief Skip to previous song
     */
    virtual void Prev() override;

    /**
     * \brief Selects a Video by Name.
     *
     * \param name
     */
    virtual void Select(std::string const& name) override;

    // delete Copy Ctor and Assign Operator to prohibit untestet behaviour
    VideoPlayerAdapter(VideoPlayerAdapter& vid) = delete;
    void operator=(VideoPlayer vid) = delete;

private:
    VideoPlayer & m_player;
};

#endif // !MUSIC_PLAYER_ADAPTER_HPP
```

6.12 VideoPlayerAdapter.cpp

```cpp
/*****************************************************************//**
 * \file   VideoPlayerAdapter.cpp
 * \brief  Adapter for the Video Player in order to Implement IPlayer Interface
 *
 * \author Simon
 * \date   November 2025
 *********************************************************************/
#include "VideoPlayerAdapter.hpp"
#include <iostream>

void VideoPlayerAdapter::Play(){
        m_player.Play();
}

void VideoPlayerAdapter::VollInc()
{
        m_player.SetVolume(m_player.GetVolume() + 1);
}

void VideoPlayerAdapter::VollDec()
{
        if (m_player.GetVolume() != 0) {
                m_player.SetVolume(m_player.GetVolume() - 1);
        }
}

void VideoPlayerAdapter::Stop()
{
        m_player.Stop();
}

void VideoPlayerAdapter::Next()
{
        // wrap around if at the end
        if (!m_player.Next()) {
                m_player.First();
        }
}

void VideoPlayerAdapter::Prev()
{
        const size_t currIndex = m_player.CurIndex();

        if (currIndex == 0) return;

        m_player.First();

        while (m_player.CurIndex() < (currIndex-1)) m_player.Next();
}

void VideoPlayerAdapter::Select(std::string const& name)
{
        size_t prev_index = m_player.CurIndex();

        m_player.First();

        while (m_player.CurVideo() != name && m_player.Next());

        if (m_player.CurVideo() != name) {
                std::cout << "video: " << name << " not found!" << std::endl;
                // switch back to the previous Video
                m_player.First();
                while (prev_index != m_player.CurIndex())m_player.Next();
        }

}
```

6.13 VideoPlayer.hpp

```cpp
/*****************************************************************//**
 * \file   VideoPlayer.hpp
 * \brief  Implementation of Video Player of the Company DonkySoft
 *
 * \author Simon Offenberger
 * \date   November 2025
 *********************************************************************/
#ifndef VIDEO_PLAYER_HPP
#define VIDEO_PLAYER_HPP

#include "Object.hpp"
#include "Video.hpp"
#include <vector>

// Using definition of the container
using TContVids = std::vector<Video>;

class VideoPlayer : public Object {
public:
        // defintion of Error Messagnes and constance
        inline static const std::string ERROR_NO_VIDEO_IN_COLLECTION = "ERROR: No video in Player!";
        inline static const std::string ERROR_DURATION_NULL = "ERROR: Video with duration 0!";
        inline static const std::string ERROR_EMPTY_NAME = "ERROR: Video with empty Name!";

        inline static const std::size_t MAX_VOLUME = 50;
        inline static const std::size_t MIN_VOLUME = 0;
        inline static const std::size_t DEFAULT_VOLUME = 8;

        VideoPlayer() = default;

        /**
         * \brief Starts playing the selected Video.
         * \throw ERROR_BAD_OSTREAM
         * \throw ERROR_FAIL_WRITE
         */
        void Play() const;

        /**
         * \brief Stops the selected Video.
         * \throw ERROR_BAD_OSTREAM
         * \throw ERROR_FAIL_WRITE
         */
        void Stop() const;

        /**
         * \brief Switches to the first video in the collection.
         *
         * \return true -> if videos are in the playlist
         * \return false -> no video in the playlist
         */
        bool First();

        /**
         * \brief Switches to the next video.
         *
         * \return true -> switch was successful
         * \return false -> no switch possiple index at top of playlist
         */
        bool Next();

        /**
         * \brief returns the current index of the selected video.
         *
         * \return Index of the current video
         * \throw ERROR_NO_VIDEO_IN_COLLECTION
         */
        size_t CurIndex() const;

        /**
         * \brief Get the name of the current video.
         *
         * \return String identidier of the video
         * \throw ERROR_NO_VIDEO_IN_COLLECTION
         */
        std::string const CurVideo() const;

        /**
         * \brief sets the volume of the player to a specified value.

         * \param vol Volume is bond to VideoPlayer::MAX_VOLUME to VideoPlayer::MIN_VOLUME
```

```
81          * \throw ERROR_BAD_OSTREAM
82          * \throw ERROR_FAIL_WRITE
83          */
84         void SetVolume(const size_t vol);
85
86         /**
87          * \brief Returns the curreunt volume of the player.
88          *
89          * \return Volume of the player
90          */
91         size_t const GetVolume() const;
92
93         /**
94          * \brief Adds a Video to the VideoPlayer.
95          *
96          * \param name Name of the Video
97          * \param dur Duration of the Video in min
98          * \param format Video Format
99          * \throw ERROR_EMPTY_NAME
100         * \throw ERROR_DURATION_NULL
101         */
102        void Add(std::string const & name, size_t const dur,EVideoFormat const & format);
103
104        // delete Copy Ctor and Assign Operator to prohibit untestet behaviour
105        VideoPlayer(VideoPlayer& vid) = delete;
106        void operator=(VideoPlayer vid) = delete;
107
108 private:
109        size_t m_volume = DEFAULT_VOLUME;
110        TContVids m_Videos;
111        size_t m_curIndex = 0;
112 };
113
114 #endif // !VIDEO_PLAYER_HPP
```

6.14 VideoPlayer.cpp

```cpp
/*****************************************************************//**
 * \file   VideoPlayer.cpp
 * \brief  Implementation of Video Player of the Company DonkySoft
 *
 * \author Simon Offenberger
 * \date   November 2025
 *********************************************************************/
#include "VideoPlayer.hpp"
#include <iostream>

void VideoPlayer::Play() const {
	if (!std::cout.good()) throw VideoPlayer::ERROR_BAD_OSTREAM;
	if (m_Videos.empty()) {
		std::cout << "no video in playlist!" << std::endl;
		return;
	}

	std::cout << "playing video number" << CurIndex();
	std::cout << ": " << CurVideo();
	std::cout << " [" << m_Videos.at(m_curIndex).GetDuration() << "min]" << std::endl;

	if (std::cout.fail()) throw VideoPlayer::ERROR_FAIL_WRITE;
}

void VideoPlayer::Stop() const {
	if (!std::cout.good()) throw VideoPlayer::ERROR_BAD_OSTREAM;
	if (m_Videos.empty()) {
		std::cout << "no video in playlist!" << std::endl;
		return;
	}

	std::cout << "stop: video: " << CurVideo();
	std::cout << " [" << m_Videos.at(m_curIndex).GetDuration() << "min]" << std::endl;

	if (std::cout.fail()) throw VideoPlayer::ERROR_FAIL_WRITE;
}

bool VideoPlayer::First()
{
	if (m_Videos.empty()) return false;

	m_curIndex = 0;

	return true;
}

bool VideoPlayer::Next()
{
	m_curIndex++;

	if (m_curIndex >= m_Videos.size()) {
		m_curIndex = m_Videos.size() - 1;
		return false;
	}
	else {
		return true;
	}
}

size_t VideoPlayer::CurIndex() const
{
	if (m_Videos.size()==0) throw VideoPlayer::ERROR_NO_VIDEO_IN_COLLECTION;

	return  m_curIndex;
}

std::string const VideoPlayer::CurVideo() const
{
	if (m_Videos.size()==0) throw VideoPlayer::ERROR_NO_VIDEO_IN_COLLECTION;

	return m_Videos.at(m_curIndex).GetTitle();
}

void VideoPlayer::SetVolume(const size_t vol)
{
	if (!std::cout.good()) throw VideoPlayer::ERROR_BAD_OSTREAM;

	if (vol > MAX_VOLUME) m_volume = MAX_VOLUME;
	else                  m_volume = vol;

```

```
81          std::cout << "volume is now -> " << m_volume;
82
83          if (std::cout.fail()) throw VideoPlayer::ERROR_FAIL_WRITE;
84  }
85
86  size_t const VideoPlayer::GetVolume() const
87  {
88          return m_volume;
89  }
90
91  void VideoPlayer::Add(std::string const& name, size_t const dur, EVideoFormat const & format)
92  {
93          if (name.empty()) throw VideoPlayer::ERROR_EMPTY_NAME;
94          if (dur == 0) throw VideoPlayer::ERROR_DURATION_NULL;
95
96          m_Videos.emplace_back(name,dur,format);
97  }
```

6.15 Video.hpp

```cpp
/*****************************************************************//**
 * \file   Video.hpp
 * \brief  Implementation of a Video
 *
 * \author Simon
 * \date   November 2025
 *********************************************************************/
#ifndef VIDEO_HPP
#define VIDEO_HPP

#include "Object.hpp"
#include "EVideoFormat.hpp"

class Video : public Object
{
public:

        // Exceptions
        inline static const std::string ERROR_DURATION_NULL = "ERROR: Video with duration 0!";
        inline static const std::string ERROR_EMPTY_NAME = "ERROR: Video with empty Name!";

        /**
         * \brief CTOR of a Video.
         *
         * \param title Tilte of the Video
         * \param duration Duration of the Video in min
         * \param format Video Format can be of Type EVideoFormat
         * \throw ERROR_DURATION_NULL
         * \throw ERROR_EMPTY_NAME
         */
        Video(const std::string& title, const size_t& duration, const EVideoFormat& format);

        /**
         * \brief Getter of the Video Title.
         *
         * \return Video Title
         */
        const std::string & GetTitle() const;

        /**
         * \brief Getter of the Video duration
         *
         * \return duration of the video
         */
        size_t GetDuration() const;

        /**
         * \brief Getter for the String Identifier of the Format.
         *
         * \return String of the Video Format
         */
        const std::string GetFormatID() const;

private:
        std::string m_title;
        size_t m_duration;
        EVideoFormat m_format;
};


#endif // !VIDEO_HPP
```

6.16 Video.cpp

```cpp
/*****************************************************************//**
 * \file   Video.cpp
 * \brief  Implementation of a Video
 *
 * \author Simon
 * \date   November 2025
 *********************************************************************/
#include "Video.hpp"

Video::Video(const std::string& title, const size_t& duration, const EVideoFormat& format)
{
    if (title.empty()) throw Video::ERROR_EMPTY_NAME;
    if (duration == 0) throw Video::ERROR_DURATION_NULL;

    m_title = title;
    m_duration = duration;
    m_format = format;
}

const std::string & Video::GetTitle() const
{
    return m_title;
}

size_t Video::GetDuration() const
{
    return m_duration;
}

const std::string Video::GetFormatID() const
{
    switch (m_format) {
    case (EVideoFormat::AVI): return "AVI-Format";
    case (EVideoFormat::MKV): return "MKV-Format";
    case (EVideoFormat::WMV): return "WMV-Format";
    default: return "unkown Format";
    }
}
```

6.17 EVideoFormat.hpp

```
/*****************************************************************//**
 * \file   EVideoFormat.hpp
 * \brief  provides an enum for the Video formats
 *
 * \author Simon
 * \date   November 2025
 *********************************************************************/
#ifndef EVIDEO_FORMAT_HPP
#define EVIDEO_FORMAT_HPP

enum class EVideoFormat
{
        AVI,
        MKV,
        WMV
};


#endif // !EVIDEO_FORMAT_HPP
```

6.18 main.cpp

```
1  #include "vld.h"
2  #include "Video.hpp"
3  #include "VideoPlayer.hpp"
4  #include "VideoPlayerAdapter.hpp"
5  #include "MusicPlayer.hpp"
6  #include "MusicPlayerAdapter.hpp"
7  #include "Client.hpp"
8  #include <iostream>
9  #include <fstream>
10 #include <cassert>
11 #include <sstream>
12 #include "Test.hpp"
13
14 using namespace std;
15
16 #define WRITE_OUTPUT true
17
18 static bool TestSong(ostream& ost);
19 static bool TestVideo(ostream& ost);
20 static bool TestVideoPlayer(ostream& ost);
21 static bool TestMusicPlayer(ostream& ost);
22
23 int main(void){
24
25         ofstream testoutput;
26         bool TestOK = true;
27
28         try {
29
30                 if (WRITE_OUTPUT == true) {
31                         testoutput.open("TestOutput.txt");
32                 }
33
34                 VideoPlayer VPlayer;
35
36                 VPlayer.Add("Harry␣Potter1", 160, EVideoFormat::AVI);
37                 VPlayer.Add("Harry␣Potter2", 160, EVideoFormat::AVI);
38                 VPlayer.Add("Harry␣Potter3", 160, EVideoFormat::AVI);
39                 VPlayer.Add("Harry␣Potter4", 160, EVideoFormat::AVI);
40                 VPlayer.Add("Harry␣Potter5", 160, EVideoFormat::AVI);
41                 VPlayer.Add("Harry␣Potter6", 160, EVideoFormat::AVI);
42
43                 VideoPlayerAdapter VidAdapter{ VPlayer };
44
45                 Client client;
46
47                 cout << "Test␣VideoPlayer␣Adapter␣in␣Client" << endl;
48                 TestOK = TestOK && client.Test_IPlayerVolumeCTRL(cout,VidAdapter,VideoPlayer::MAX_VOLUME,VideoPlayer
                       ::DEFAULT_VOLUME);
49                 TestOK = TestOK && client.Test_IPlayerPlay(cout,VidAdapter);
50
51                 VideoPlayer VPlayer2;
52
53                 VPlayer2.Add("Harry␣Potter1", 160, EVideoFormat::AVI);
54                 VPlayer2.Add("Harry␣Potter2", 160, EVideoFormat::AVI);
55                 VPlayer2.Add("Harry␣Potter3", 160, EVideoFormat::AVI);
56                 VPlayer2.Add("Harry␣Potter4", 160, EVideoFormat::AVI);
57                 VPlayer2.Add("Harry␣Potter5", 160, EVideoFormat::AVI);
58                 VPlayer2.Add("Harry␣Potter6", 160, EVideoFormat::AVI);
59
60                 VideoPlayerAdapter VidAdapter2{ VPlayer2 };
61
62                 if (WRITE_OUTPUT) {
63                         testoutput << "Test␣VideoPlayer␣Adapter␣in␣Client" << endl;
64                         TestOK = TestOK && client.Test_IPlayerVolumeCTRL(testoutput, VidAdapter2, VideoPlayer::
                               MAX_VOLUME, VideoPlayer::DEFAULT_VOLUME);
65                         TestOK = TestOK && client.Test_IPlayerPlay(testoutput, VidAdapter2);
66                 }
67
68                 VideoPlayer EmptyPlayer;
69                 VideoPlayerAdapter EmptyAdapter { EmptyPlayer };
70
71                 TestOK = TestOK && client.Test_IPlayerEmptyPlay(cout, EmptyAdapter);
72                 if (WRITE_OUTPUT) TestOK = TestOK && client.Test_IPlayerEmptyPlay(testoutput, EmptyAdapter);
73
74
75                 MusicPlayer MPlayer;
76
77                 MPlayer.Add("Harry␣Potter1", 160);
78                 MPlayer.Add("Harry␣Potter2", 160);
```

```
79                  MPlayer.Add("Harry Potter3", 160);
80                  MPlayer.Add("Harry Potter4", 160);
81                  MPlayer.Add("Harry Potter5", 160);
82                  MPlayer.Add("Harry Potter6", 160);
83
84                  MusicPlayerAdapter MusAdapter{ MPlayer };
85
86                  cout << "Test MusicPlayer Adapter in Client" << endl;
87                  TestOK = TestOK && client.Test_IPlayerVolumeCTRL(cout, MusAdapter,MusicPlayer::MAX_VOLUME,
                          MusicPlayer::DEFAULT_VOLUME);
88                  TestOK = TestOK && client.Test_IPlayerPlay(cout, MusAdapter);
89
90
91                  MusicPlayer MPlayer2;
92
93                  MPlayer2.Add("Harry Potter1", 160);
94                  MPlayer2.Add("Harry Potter2", 160);
95                  MPlayer2.Add("Harry Potter3", 160);
96                  MPlayer2.Add("Harry Potter4", 160);
97                  MPlayer2.Add("Harry Potter5", 160);
98                  MPlayer2.Add("Harry Potter6", 160);
99
100                 MusicPlayerAdapter MusAdapter2{ MPlayer2 };
101
102
103                 if (WRITE_OUTPUT) {
104                         testoutput << "Test MusicPlayer Adapter in Client" << endl;
105                         TestOK = TestOK && client.Test_IPlayerVolumeCTRL(testoutput, MusAdapter2, MusicPlayer::
                                MAX_VOLUME, MusicPlayer::DEFAULT_VOLUME);
106                         TestOK = TestOK && client.Test_IPlayerPlay(testoutput, MusAdapter2);
107                 }
108
109                 MusicPlayer EmptyMPlayer;
110                 MusicPlayerAdapter EmptyMAdapter{ EmptyMPlayer };
111
112                 TestOK = TestOK && client.Test_IPlayerEmptyPlay(cout, EmptyMAdapter);
113                 if (WRITE_OUTPUT) TestOK = TestOK && client.Test_IPlayerEmptyPlay(testoutput, EmptyMAdapter);
114
115
116                 TestOK = TestOK && TestSong(cout);
117                 if (WRITE_OUTPUT) TestOK = TestOK && TestSong(testoutput);
118
119                 TestOK = TestOK && TestVideo(cout);
120                 if (WRITE_OUTPUT) TestOK = TestOK && TestVideo(testoutput);
121
122                 TestOK = TestOK && TestVideoPlayer(cout);
123                 if (WRITE_OUTPUT) TestOK = TestOK && TestVideoPlayer(testoutput);
124
125                 TestOK = TestOK && TestMusicPlayer(cout);
126                 if (WRITE_OUTPUT) TestOK = TestOK && TestMusicPlayer(testoutput);
127
128
129                 if (WRITE_OUTPUT) {
130                         if (TestOK) TestCaseOK(testoutput);
131                         else TestCaseFail(testoutput);
132
133                         testoutput.close();
134                 }
135
136                 if (TestOK) TestCaseOK(cout);
137                 else TestCaseFail(cout);
138
139         }
140         catch (const string& err) {
141                 cerr << err;
142         }
143         catch (bad_alloc const& error) {
144                 cerr << error.what();
145         }
146         catch (const exception& err) {
147                 cerr << err.what();
148         }
149         catch (...) {
150                 cerr << "Unhandelt Exception";
151         }
152
153         if (testoutput.is_open()) testoutput.close();
154
155         return 0;
156 }
157
158 bool TestSong(ostream& ost)
159 {
```

```
160         assert(ost.good());
161
162         TestStart(ost);
163
164         bool TestOK = true;
165         string error_msg = "";
166
167         try {
168
169                 Song HelloWorld("Hello World", 123);
170
171                 TestOK = TestOK && check_dump(ost,"Test Song Getter Duration", static_cast<size_t>(123), HelloWorld.
                        GetDuration());
172
173                 TestOK = TestOK && check_dump(ost,"Test Song Getter Name",static_cast<string>( "Hello World"),
                        HelloWorld.GetTitle());
174
175         }
176         catch (const string& err) {
177                 error_msg = err;
178         }
179         catch (bad_alloc const& error) {
180                 error_msg = error.what();
181         }
182         catch (const exception& err) {
183                 error_msg = err.what();
184         }
185         catch (...) {
186                 error_msg = "Unhandelt Exception";
187         }
188
189         TestOK = TestOK && check_dump(ost, "Check for Exception in Testcase", true, error_msg.empty());
190         error_msg.clear();
191
192         try {
193                 Song song{ "Hello World",0};
194         }
195         catch (const string& err) {
196                 error_msg = err;
197         }
198         catch (bad_alloc const& error) {
199                 error_msg = error.what();
200         }
201         catch (const exception& err) {
202                 error_msg = err.what();
203         }
204         catch (...) {
205                 error_msg = "Unhandelt Exception";
206         }
207
208         TestOK = TestOK && check_dump(ost, "Test Exception in Song CTOR with duration 0", error_msg, Song::
              ERROR_DURATION_NULL);
209         error_msg.clear();
210
211         try {
212
213                 Song song{ "",12};
214
215         }
216         catch (const string& err) {
217                 error_msg = err;
218         }
219         catch (bad_alloc const& error) {
220                 error_msg = error.what();
221         }
222         catch (const exception& err) {
223                 error_msg = err.what();
224         }
225         catch (...) {
226                 error_msg = "Unhandelt Exception";
227         }
228
229         TestOK = TestOK && check_dump(ost, "Test Exception in Song CTOR with empty string", error_msg, Song::
              ERROR_EMPTY_NAME);
230         error_msg.clear();
231
232         TestEnd(ost);
233         return TestOK;
234 }
235
236
237 bool TestVideo(ostream& ost)
238 {
```

```
239         assert(ost.good());
240
241         TestStart(ost);
242
243         bool TestOK = true;
244         string error_msg = "";
245
246         try {
247
248                 Video HelloWorld("Hello World", 123,EVideoFormat::AVI);
249
250                 TestOK = TestOK && check_dump(ost,"Test Song Getter Duration", static_cast<size_t>(123), HelloWorld.
                        GetDuration());
251
252                 TestOK = TestOK && check_dump(ost,"Test Song Getter Name",static_cast<string>( "Hello World"),
                        HelloWorld.GetTitle());
253
254                 TestOK = TestOK && check_dump(ost,"Test Song Getter Format", static_cast<string>("AVI-Format"),
                        HelloWorld.GetFormatID());
255         }
256         catch (const string& err) {
257                 error_msg = err;
258         }
259         catch (bad_alloc const& error) {
260                 error_msg = error.what();
261         }
262         catch (const exception& err) {
263                 error_msg = err.what();
264         }
265         catch (...) {
266                 error_msg = "Unhandelt Exception";
267         }
268
269         TestOK = TestOK && check_dump(ost, "Check for Exception in Testcase", true, error_msg.empty());
270         error_msg.clear();
271
272         try {
273                 Video vid{ "Hello World",0, EVideoFormat::AVI };
274         }
275         catch (const string& err) {
276                 error_msg = err;
277         }
278         catch (bad_alloc const& error) {
279                 error_msg = error.what();
280         }
281         catch (const exception& err) {
282                 error_msg = err.what();
283         }
284         catch (...) {
285                 error_msg = "Unhandelt Exception";
286         }
287
288         TestOK = TestOK && check_dump(ost, "Test Exception in Video CTOR with duration 0", error_msg, Video::
              ERROR_DURATION_NULL);
289         error_msg.clear();
290
291         try {
292
293                 Video vid{ "",12,EVideoFormat::MKV };
294
295         }
296         catch (const string& err) {
297                 error_msg = err;
298         }
299         catch (bad_alloc const& error) {
300                 error_msg = error.what();
301         }
302         catch (const exception& err) {
303                 error_msg = err.what();
304         }
305         catch (...) {
306                 error_msg = "Unhandelt Exception";
307         }
308
309         TestOK = TestOK && check_dump(ost, "Test Exception in Video CTOR with empty string", error_msg, Video::
              ERROR_EMPTY_NAME);
310         error_msg.clear();
311
312
313         TestEnd(ost);
314         return TestOK;
315 }
316
```

```
317 bool TestVideoPlayer(ostream& ost)
318 {
319         assert(ost.good());
320
321         TestStart(ost);
322
323         bool TestOK = true;
324         string error_msg = "";
325
326         try {
327
328                 Video HelloWorld("Hello World", 123, EVideoFormat::AVI);
329
330                 VideoPlayer VPlayer;
331
332                 VPlayer.Add("Hello World1",123,EVideoFormat::AVI);
333                 VPlayer.Add("Hello World2",124,EVideoFormat::MKV);
334                 VPlayer.Add("Hello World3",125,EVideoFormat::WMV);
335                 VPlayer.Add("Hello World4",126,EVideoFormat::AVI);
336                 VPlayer.Add("Hello World5",127,EVideoFormat::MKV);
337
338                 TestOK = TestOK && check_dump(ost, "Test Videplayer Initial Index", static_cast<size_t>(0), VPlayer.
                        CurIndex());
339
340                 VPlayer.First();
341
342                 TestOK = TestOK && check_dump(ost, "Test Videplayer Index after First", static_cast<size_t>(0),
                        VPlayer.CurIndex());
343
344                 VPlayer.Next();
345
346                 TestOK = TestOK && check_dump(ost, "Test Videplayer Index after Next", static_cast<size_t>(1),
                        VPlayer.CurIndex());
347
348                 for (int i = 0; i < 100;i++) VPlayer.Next();
349
350                 TestOK = TestOK && check_dump(ost, "Test Videplayer Index Upper Bound", static_cast<size_t>(4),
                        VPlayer.CurIndex());
351
352                 VPlayer.First();
353
354                 TestOK = TestOK && check_dump(ost, "Test Videplayer Index after First", static_cast<size_t>(0),
                        VPlayer.CurIndex());
355
356                 TestOK = TestOK && check_dump(ost, "Test Default Volume", static_cast<size_t>(8), VPlayer.GetVolume
                        ());
357
358                 std::streambuf* coutbuf = std::cout.rdbuf();
359
360                 stringstream result;
361
362                 // cout redirect to stringstream
363                 std::cout.rdbuf(result.rdbuf());
364
365                 VPlayer.SetVolume(25);
366
367                 std::cout.rdbuf(coutbuf);
368
369                 TestOK = TestOK && check_dump(ost, "Test Set Volume", static_cast<size_t>(25), VPlayer.GetVolume());
370
371                 // cout redirect to stringstream
372                 std::cout.rdbuf(result.rdbuf());
373
374                 VPlayer.SetVolume(300);
375
376                 std::cout.rdbuf(coutbuf);
377
378                 TestOK = TestOK && check_dump(ost, "Test Set Volume Max Volume", static_cast<size_t>(VideoPlayer::
                        MAX_VOLUME), VPlayer.GetVolume());
379
380                 // cout redirect to stringstream
381                 std::cout.rdbuf(result.rdbuf());
382
383                 VPlayer.SetVolume(0);
384
385                 std::cout.rdbuf(coutbuf);
386
387                 TestOK = TestOK && check_dump(ost, "Test Set Volume Min Volume", static_cast<size_t>(VideoPlayer::
                        MIN_VOLUME), VPlayer.GetVolume());
388
389
390                 result.str("");
391                 result.clear();
```

```
392                 // cout redirect to stringstream
393                 std::cout.rdbuf(result.rdbuf());
394
395                 VPlayer.Play();
396
397                 std::cout.rdbuf(coutbuf);
398
399                 TestOK = TestOK && check_dump(ost, "Test Video Player Play", true, result.str().find(VPlayer.
                        CurVideo()) != string::npos);
400
401                 result.str("");
402                 result.clear();
403                 // cout redirect to stringstream
404                 std::cout.rdbuf(result.rdbuf());
405
406                 VPlayer.Stop();
407
408                 std::cout.rdbuf(coutbuf);
409
410                 TestOK = TestOK && check_dump(ost, "Test Video Player Stop", true, result.str().find("stop") !=
                        string::npos);
411         }
412         catch (const string& err) {
413                 error_msg = err;
414         }
415         catch (bad_alloc const& error) {
416                 error_msg = error.what();
417         }
418         catch (const exception& err) {
419                 error_msg = err.what();
420         }
421         catch (...) {
422                 error_msg = "Unhandelt Exception";
423         }
424
425         TestOK = TestOK && check_dump(ost, "Check for Exception in Testcase", true, error_msg.empty());
426         error_msg.clear();
427
428         try{
429                 VideoPlayer VidPlayer;
430                 VidPlayer.Add("", 123, EVideoFormat::AVI);
431
432         }
433         catch (const string& err) {
434                 error_msg = err;
435         }
436         catch (bad_alloc const& error) {
437                 error_msg = error.what();
438         }
439         catch (const exception& err) {
440                 error_msg = err.what();
441         }
442         catch (...) {
443                 error_msg = "Unhandelt Exception";
444         }
445
446         TestOK = TestOK && check_dump(ost, "Test Exception in Add with empty string", error_msg, VideoPlayer::
               ERROR_EMPTY_NAME);
447         error_msg.clear();
448
449         try{
450                 VideoPlayer VidPlayer;
451                 VidPlayer.Add("234", 0, EVideoFormat::AVI);
452
453         }
454         catch (const string& err) {
455                 error_msg = err;
456         }
457         catch (bad_alloc const& error) {
458                 error_msg = error.what();
459         }
460         catch (const exception& err) {
461                 error_msg = err.what();
462         }
463         catch (...) {
464                 error_msg = "Unhandelt Exception";
465         }
466
467         TestOK = TestOK && check_dump(ost, "Test Exception in Add with empty string", error_msg, VideoPlayer::
               ERROR_DURATION_NULL);
468         error_msg.clear();
469
470
```

```
471         TestEnd(ost);
472         return TestOK;
473 }
474
475 bool TestMusicPlayer(ostream& ost)
476 {
477         assert(ost.good());
478
479         TestStart(ost);
480
481         bool TestOK = true;
482         string error_msg = "";
483
484         // test basic functionality
485         try {
486                 MusicPlayer music;
487                 std::string const song1 = "How much is the Fish - Scooter";
488                 std::string const song2 = "Die Blume aus dem Gemeindebau - Wolfgang Ambros";
489                 std::string const song3 = "Red Sun in the Sky - MaoZe";
490                 std::string const song4 = "Ski-Twist - Hansi Hinterseer";
491                 size_t const dur1 = 300;
492                 size_t const dur2 = 240;
493                 size_t const dur3 = 180;
494                 size_t const dur4 = 110;
495                 size_t const songCount = 4;
496                 music.Add(song1, dur1);
497                 music.Add(song2, dur2);
498                 music.Add(song3, dur3);
499                 music.Add(song4, dur4);
500
501                 TestOK = TestOK && check_dump(ost, "MusicPlayer - Basic Functionality - .GetCount()", music.GetCount
                        (), songCount);
502                 TestOK = TestOK && check_dump(ost, "MusicPlayer - Basic Functionality - .GetIndex() initial", music.
                        GetCurIndex(), static_cast<size_t>(0));
503                 TestOK = TestOK && check_dump(ost, "MusicPlayer - Basic Functionality - .Find() unknown song", music
                        .Find("not a real song"), false);
504                 TestOK = TestOK && check_dump(ost, "MusicPlayer - Basic Functionality - .Find() song that exists",
                        music.Find(song1), true);
505
506                 // for checking cout
507                 std::streambuf* coutbuf = std::cout.rdbuf();
508                 stringstream result;
509
510                 // cout redirect to stringstream
511                 std::cout.rdbuf(result.rdbuf());
512                 music.Start();
513                 std::cout.rdbuf(coutbuf);
514
515                 TestOK = TestOK && check_dump(ost, "MusicPlayer - Basic Functionality - Song name after initial -
                        Start()", true, result.str().find(song1) != string::npos);
516                 result.str("");
517                 result.clear();
518
519                 music.SwitchNext();
520                 std::cout.rdbuf(result.rdbuf());
521                 music.Start();
522                 std::cout.rdbuf(coutbuf);
523
524                 TestOK = TestOK && check_dump(ost, "MusicPlayer - Basic Functionality - .GetIndex() after switching"
                        , static_cast<size_t>(1), music.GetCurIndex());
525                 TestOK = TestOK && check_dump(ost, "MusicPlayer - Basic Functionality - Song name switching", true,
                        result.str().find(song2) != string::npos);
526                 result.str("");
527                 result.clear();
528
529                 // wrap around
530                 for (int i = 0; i < music.GetCount(); i++)
531                 {
532                         music.SwitchNext();
533                 }
534
535                 std::cout.rdbuf(result.rdbuf());
536                 music.Stop();
537                 std::cout.rdbuf(coutbuf);
538
539                 TestOK = TestOK && check_dump(ost, "MusicPlayer - Basic Functionality - .GetIndex() wrap around",
                        static_cast<size_t>(1), music.GetCurIndex());
540         }
541         catch (const string& err) {
542                 error_msg = err;
543         }
544         catch (bad_alloc const& error) {
545                 error_msg = error.what();
```

```
546         }
547         catch (const exception& err) {
548                 error_msg = err.what();
549         }
550         catch (...) {
551                 error_msg = "Unhandelt Exception";
552         }
553 
554         TestOK = TestOK && check_dump(ost, "MusicPlayer - Basic Functionality - Error Buffer", true, error_msg.empty
                ());
555         error_msg.clear();
556 
557         // Add empty song
558         try {
559                 MusicPlayer music;
560                 std::string const song = "";
561                 size_t const dur = 1;
562                 music.Add(song, dur);
563         }
564         catch (const string& err) {
565                 error_msg = err;
566         }
567         catch (bad_alloc const& error) {
568                 error_msg = error.what();
569         }
570         catch (const exception& err) {
571                 error_msg = err.what();
572         }
573         catch (...) {
574                 error_msg = "Unhandelt Exception";
575         }
576 
577         TestOK = TestOK && check_dump(ost, "MusicPlayer - Add Song without title", MusicPlayer::ERROR_EMPTY_NAME,
              error_msg);
578         error_msg.clear();
579 
580         // Add song with 0 duration
581         try {
582                 MusicPlayer music;
583                 std::string const song = "This is a legit song";
584                 size_t const dur = 0;
585                 music.Add(song, dur);
586         }
587         catch (const string& err) {
588                 error_msg = err;
589         }
590         catch (bad_alloc const& error) {
591                 error_msg = error.what();
592         }
593         catch (const exception& err) {
594                 error_msg = err.what();
595         }
596         catch (...) {
597                 error_msg = "Unhandelt Exception";
598         }
599 
600         TestOK = TestOK && check_dump(ost, "MusicPlayer - Add Song without title", MusicPlayer::ERROR_DURATION_NULL,
              error_msg);
601         error_msg.clear();
602 
603         // find empty name
604         try {
605                 MusicPlayer music;
606                 music.Find("");
607         }
608         catch (const string& err) {
609                 error_msg = err;
610         }
611         catch (bad_alloc const& error) {
612                 error_msg = error.what();
613         }
614         catch (const exception& err) {
615                 error_msg = err.what();
616         }
617         catch (...) {
618                 error_msg = "Unhandelt Exception";
619         }
620 
621         TestOK = TestOK && check_dump(ost, "MusicPlayer - Add Song without title", MusicPlayer::ERROR_EMPTY_NAME,
              error_msg);
622         error_msg.clear();
623 
624         return TestOK;
```

```
625 }
```

6.19 Test.hpp

```
/*****************************************************************//**
 * \file   Test.hpp
 * \brief  File that provides a Test Function with a formated output
 *
 * \author Simon
 * \date   April 2025
 *********************************************************************/
#ifndef TEST_HPP
#define TEST_HPP

#include <string>
#include <iostream>
#include <vector>
#include <list>
#include <queue>
#include <forward_list>

#define ON 1
#define OFF 0
#define COLOR_OUTPUT OFF

// Definitions of colors in order to change the color of the output stream.
const std::string colorRed = "\x1B[31m";
const std::string colorGreen = "\x1B[32m";
const std::string colorWhite = "\x1B[37m";

inline std::ostream& RED(std::ostream& ost) {
	if (ost.good()) {
		ost << colorRed;
	}
	return ost;
}
inline std::ostream& GREEN(std::ostream& ost) {
	if (ost.good()) {
		ost << colorGreen;
	}
	return ost;
}
inline std::ostream& WHITE(std::ostream& ost) {
	if (ost.good()) {
		ost << colorWhite;
	}
	return ost;
}

inline std::ostream& TestStart(std::ostream& ost) {
	if (ost.good()) {
		ost << std::endl;
		ost << "****************************************" << std::endl;
		ost << "__________TESTCASE_START__________" << std::endl;
		ost << "****************************************" << std::endl;
		ost << std::endl;
	}
	return ost;
}

inline std::ostream& TestEnd(std::ostream& ost) {
	if (ost.good()) {
		ost << std::endl;
		ost << "****************************************" << std::endl;
		ost << std::endl;
	}
	return ost;
}

inline std::ostream& TestCaseOK(std::ostream& ost) {

#if COLOR_OUTPUT
	if (ost.good()) {
		ost << colorGreen << "TEST_OK!!" << colorWhite << std::endl;
	}
#else
	if (ost.good()) {
		ost <<  "TEST_OK!!" <<  std::endl;
	}
#endif // COLOR_OUTPUT

	return ost;
}

```

```
81  inline std::ostream& TestCaseFail(std::ostream& ost) {
82
83  #if COLOR_OUTPUT
84          if (ost.good()) {
85                  ost << colorRed << "TEST␣FAILED␣!!" << colorWhite << std::endl;
86
87          }
88  #else
89          if (ost.good()) {
90                  ost << "TEST␣FAILED␣!!" << std::endl;
91
92          }
93  #endif // COLOR_OUTPUT
94
95          return ost;
96  }
97
98  /**
99          * \brief function that reports if the testcase was successful.
100         *
101         * \param testcase       String that indicates the testcase
102         * \param succsessful true -> reports to cout test OK
103         * \param succsessful false -> reports test failed
104         */
105 template <typename T>
106 bool check_dump(std::ostream& ostr, const std::string& testcase, const T& expected, const T& result) {
107         if (ostr.good()) {
108 #if COLOR_OUTPUT
109                 if (expected == result) {
110                         ostr << testcase << std::endl <<  colorGreen << "[Test␣OK]␣" << colorWhite <<"Result:␣(
                                Expected:␣" << std::boolalpha << expected << "␣==" << "␣Result:␣" << result << ")"  <<
                                 std::noboolalpha << std::endl << std::endl;
111                 }
112                 else {
113                         ostr << testcase << std::endl << colorRed << "[Test␣FAILED]␣" << colorWhite << "Result:␣(
                                Expected:␣" << std::boolalpha << expected << "␣!=" << "␣Result:␣" << result << ")"  <<
                                 std::noboolalpha << std::endl << std::endl;
114                 }
115 #else
116                 if (expected == result) {
117                         ostr << testcase << std::endl << "[Test␣OK]␣"  << "Result:␣(Expected:␣" << std::boolalpha <<
                                 expected << "␣==" << "␣Result:␣" << result << ")" << std::noboolalpha << std::endl <<
                                 std::endl;
118                 }
119                 else {
120                         ostr << testcase << std::endl  << "[Test␣FAILED]␣"  << "Result:␣(Expected:␣" << std::
                                boolalpha << expected << "␣!=" << "␣Result:␣" << result << ")" << std::noboolalpha <<
                                std::endl << std::endl;
121                 }
122 #endif
123                 if (ostr.fail()) {
124                         std::cerr << "Error:␣Write␣Ostream" << std::endl;
125                 }
126         }
127         else {
128                 std::cerr << "Error:␣Bad␣Ostream" << std::endl;
129         }
130         return expected == result;
131 }
132
133 template <typename T1, typename T2>
134 std::ostream& operator<< (std::ostream& ost,const std::pair<T1,T2> & p) {
135         if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
136         ost << "(" << p.first << "," << p.second << ")";
137         return ost;
138 }
139
140 template <typename T>
141 std::ostream& operator<< (std::ostream& ost,const std::vector<T> & cont) {
142         if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
143         std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>{ost, "␣"});
144         return ost;
145 }
146
147 template <typename T>
148 std::ostream& operator<< (std::ostream& ost,const std::list<T> & cont) {
149         if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
150         std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>{ost, "␣"});
151         return ost;
152 }
153
154 template <typename T>
155 std::ostream& operator<< (std::ostream& ost,const std::deque<T> & cont) {
```

```
156         if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
157         std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>{ost, "␣"});
158         return ost;
159 }
160
161 template <typename T>
162 std::ostream& operator<< (std::ostream& ost,const std::forward_list<T> & cont) {
163         if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
164         std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>{ost, "␣"});
165         return ost;
166 }
167
168
169 #endif // !TEST_HPP
```