

**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:

```

1 //starts playing with the current song in list
2 void Play() const;
3 //stops playing
4 void Stop() const;
5 //switches to first video in playlist and returns true, otherwise false if list is empty.
6 bool First();
7 //switches to next video in playlist and returns true, otherwise false if last song is reached.
8 bool Next();
9 //returns index of current video
10 size_t CurIndex() const;
11 //returns name of current video
12 std::string const CurVideo() const;
13 //sets volume (min volume=0 and max volume=50)
14 void SetVolume(size_t const vol);
15 //gets current volume
16 size_t const GetVolume() const;
17 //adds a video to playlist
18 void Add(std::string const& name, size_t const dur, VideoFormat const& format);
19 }
```

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:

```

1 virtual void Play() = 0;
2 virtual void VolInc() = 0;
3 virtual void VolDec() = 0;
4 virtual void Stop() = 0;
5 virtual void Next() = 0;
6 virtual void Prev() = 0;
7 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 Fuhrpark**

**Version 1.0**

S. Offenberger, S. Vogelhuber

Hagenberg, 2. November 2025

## Inhaltsverzeichnis

# 1 Organisatorisches

## 1.1 Team

- Simon Offenberger, Matr.-Nr.: S2410306027, E-Mail: Simon.Offenberger@fh-hagenberg.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**

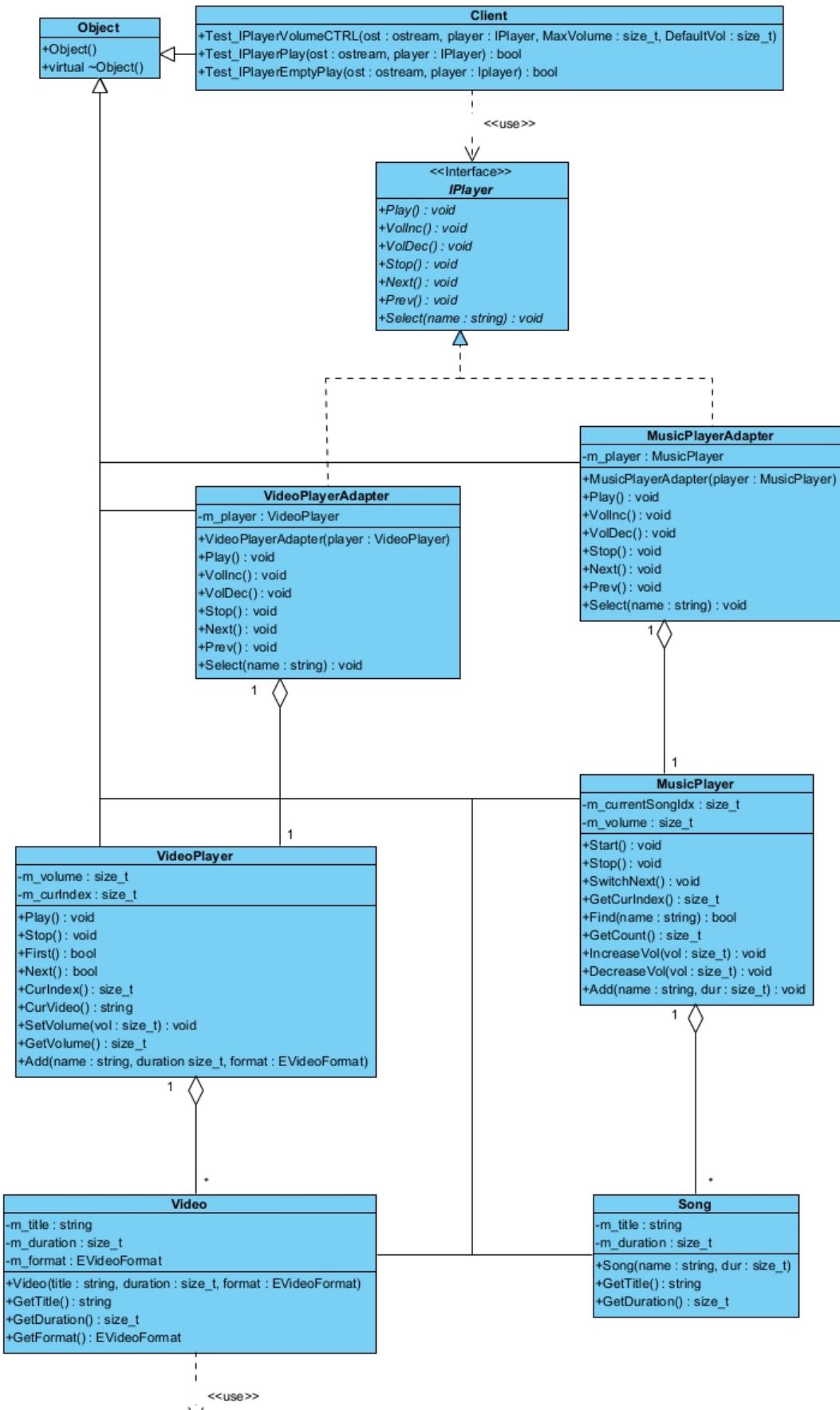
- Simon Offenberger: geschätzt 12 Ph / tatsächlich 11 Ph
- Simon Vogelhuber: geschätzt 9 Ph / tatsächlich 9 Ph

## **2 Anforderungsdefinition (Systemspezifikation)**



## 3 Systementwurf

### 3.1 Klassendiagramm



### **3.2 Designentscheidungen**

## **4 Dokumentation der Komponenten (Klassen)**

Die HTML-Startdatei befindet sich im Verzeichnis [\*./doxy/html/index.html\*](#)

## 5 Testprotokollierung

```
1 Test VideoPlayer Adapter in Client
2
3 *****
4         TESTCASE START
5 *****
6
7 Test Volume Inc
8 [31m[Test FAILED] [37mResult: (Expected: true != Result: false)
9
10 Test Volume Dec
11 [31m[Test FAILED] [37mResult: (Expected: true != Result: false)
12
13 Test Lower Bound Volume 0
14 [32m[Test OK] [37mResult: (Expected: true == Result: true)
15
16 Test Upper Bound Volume
17 [32m[Test OK] [37mResult: (Expected: true == Result: true)
18
19 Test for Exception in Test Case
20 [32m[Test OK] [37mResult: (Expected: true == Result: true)
21
22
23 *****
24
25
26 *****
27         TESTCASE START
28 *****
29
30 Test Play Contains Name
31 [31m[Test FAILED] [37mResult: (Expected: true != Result: false)
32
33 Test Next
34 [31m[Test FAILED] [37mResult: (Expected: true != Result: false)
35
36 Test Next
37 [32m[Test OK] [37mResult: (Expected: true == Result: true)
38
39 Test Next
40 [32m[Test OK] [37mResult: (Expected: true == Result: true)
41
42 Test Next
```

```
43 [32m[Test OK] [37mResult: (Expected: true == Result: true)
44
45 Test Next
46 [32m[Test OK] [37mResult: (Expected: true == Result: true)
47
48 Test Next Wrap around
49 [31m[Test FAILED] [37mResult: (Expected: true != Result: false)
50
51 Test Select Video by name
52 [32m[Test OK] [37mResult: (Expected: true == Result: true)
53
54 Test Select Video by name not found
55 [32m[Test OK] [37mResult: (Expected: true == Result: true)
56
57 Test Stop Player
58 [32m[Test OK] [37mResult: (Expected: true == Result: true)
59
60 Test for Exception in Test Case
61 [32m[Test OK] [37mResult: (Expected: true == Result: true)
62
63
64 ****
65
66
67 **** TESTCASE START ****
68
69 ****
70
71 Test for Message in Empty Player
72 [32m[Test OK] [37mResult: (Expected: true == Result: true)
73
74 Test for Exception in Testcase
75 [32m[Test OK] [37mResult: (Expected: true == Result: true)
76
77
78 ****
79
80 Test MusicPlayer Adapter in Client
81
82 **** TESTCASE START ****
83
84 ****
85
86 Test Volume Inc
```

```
87 [31m[Test FAILED] [37mResult: (Expected: true != Result: false)
88
89 Test Volume Dec
90 [31m[Test FAILED] [37mResult: (Expected: true != Result: false)
91
92 Test Lower Bound Volume 0
93 [32m[Test OK] [37mResult: (Expected: true == Result: true)
94
95 Test Upper Bound Volume
96 [32m[Test OK] [37mResult: (Expected: true == Result: true)
97
98 Test for Exception in Test Case
99 [32m[Test OK] [37mResult: (Expected: true == Result: true)
100
101
102 ****
103
104
105 ****
106         TESTCASE START
107 ****
108
109 Test Play Contains Name
110 [31m[Test FAILED] [37mResult: (Expected: true != Result: false)
111
112 Test Next
113 [31m[Test FAILED] [37mResult: (Expected: true != Result: false)
114
115 Test Next
116 [32m[Test OK] [37mResult: (Expected: true == Result: true)
117
118 Test Next
119 [32m[Test OK] [37mResult: (Expected: true == Result: true)
120
121 Test Next
122 [32m[Test OK] [37mResult: (Expected: true == Result: true)
123
124 Test Next
125 [32m[Test OK] [37mResult: (Expected: true == Result: true)
126
127 Test Next Wrap around
128 [31m[Test FAILED] [37mResult: (Expected: true != Result: false)
129
130 Test Select Video by name
```

```
131 [32m[Test OK] [37mResult: (Expected: true == Result: true)
132
133 Test Select Video by name not found
134 [32m[Test OK] [37mResult: (Expected: true == Result: true)
135
136 Test Stop Player
137 [32m[Test OK] [37mResult: (Expected: true == Result: true)
138
139 Test for Exception in Test Case
140 [32m[Test OK] [37mResult: (Expected: true == Result: true)
141
142
143 ****
144
145
146 ****
147         TESTCASE START
148 ****
149
150 Test for Message in Empty Player
151 [32m[Test OK] [37mResult: (Expected: true == Result: true)
152
153 Test for Exception in Testcase
154 [32m[Test OK] [37mResult: (Expected: true == Result: true)
155
156
157 ****
158
159
160 ****
161         TESTCASE START
162 ****
163
164 Test Song Getter Duration
165 [32m[Test OK] [37mResult: (Expected: 123 == Result: 123)
166
167 Test Song Getter Name
168 [32m[Test OK] [37mResult: (Expected: Hello World == Result: Hello World)
169
170 Check for Exception in Testcase
171 [32m[Test OK] [37mResult: (Expected: true == Result: true)
172
173 Test Exception in Song CTOR with duration 0
```

```
174 [32m[Test OK] [37mResult: (Expected: ERROR: Song with duration 0! == Result
     ↪ : ERROR: Song with duration 0!)
175
176 Test Exception in Song CTOR with empty string
177 [32m[Test OK] [37mResult: (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 [32m[Test OK] [37mResult: (Expected: 123 == Result: 123)
189
190 Test Song Getter Name
191 [32m[Test OK] [37mResult: (Expected: Hello World == Result: Hello World)
192
193 Test Song Getter Format
194 [32m[Test OK] [37mResult: (Expected: AVI-Format == Result: AVI-Format)
195
196 Check for Exception in Testcase
197 [32m[Test OK] [37mResult: (Expected: true == Result: true)
198
199 Test Exception in Video CTOR with duration 0
200 [32m[Test OK] [37mResult: (Expected: ERROR: Video with duration 0! ==
     ↪ Result: ERROR: Video with duration 0!)
201
202 Test Exception in Video CTOR with empty string
203 [32m[Test OK] [37mResult: (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 Videoplayer Initial Index
```

```
214 [32m[Test OK] [37mResult: (Expected: 0 == Result: 0)
215
216 Test Videoplayer Index after First
217 [32m[Test OK] [37mResult: (Expected: 0 == Result: 0)
218
219 Test Videoplayer Index after Next
220 [32m[Test OK] [37mResult: (Expected: 1 == Result: 1)
221
222 Test Videoplayer Index Upper Bound
223 [32m[Test OK] [37mResult: (Expected: 4 == Result: 4)
224
225 Test Videoplayer Index after First
226 [32m[Test OK] [37mResult: (Expected: 0 == Result: 0)
227
228 Test Default Volume
229 [32m[Test OK] [37mResult: (Expected: 8 == Result: 8)
230
231 Test Set Volume
232 [32m[Test OK] [37mResult: (Expected: 25 == Result: 25)
233
234 Test Set Volume Max Volume
235 [32m[Test OK] [37mResult: (Expected: 50 == Result: 50)
236
237 Test Set Volume Min Volume
238 [32m[Test OK] [37mResult: (Expected: 0 == Result: 0)
239
240 Test Video Player Play
241 [32m[Test OK] [37mResult: (Expected: true == Result: true)
242
243 Test Video Player Stop
244 [32m[Test OK] [37mResult: (Expected: true == Result: true)
245
246 Check for Exception in Testcase
247 [32m[Test OK] [37mResult: (Expected: true == Result: true)
248
249 Test Exception in Add with empty string
250 [32m[Test OK] [37mResult: (Expected: ERROR: Video with empty Name! ==
251     ↪ Result: ERROR: Video with empty Name!)
252
253 Test Exception in Add with empty string
254 [32m[Test OK] [37mResult: (Expected: ERROR: Video with duration 0! ==
255     ↪ Result: ERROR: Video with duration 0!)
```

```
256 *****  
257 [ 32mTEST OK!! [ 37m  
258 ]
```

## 6 Quellcode

### 6.1 Object.hpp

```

1 #ifndef OBJECT_HPP
2 #define OBJECT_HPP
3
4 #include <string>
5
6 class Object {
7 public:
8
9     inline static const std::string ERROR_BAD_OSTREAM = "ERROR:_Provided_Ostream_is_bad";
10    inline static const std::string ERROR_FAIL_WRITE = "ERROR:_Fail_to_write_on_Provided_Ostream";
11    inline static const std::string ERROR_NULLPTR = "ERROR:_Passed_in_Nullptr!";
12
13 protected:
14     Object() = default;
15
16     virtual ~Object() = default;
17
18 };
19
20 #endif // !OBJECT_HPP

```

### 6.2 Client.hpp

```

1 //*****
2 * \file Client.hpp
3 * \brief Client Class that uses a IPlayer Interface inorder to control
4 * \brief a Musicplayer or a Videoplayer via their adapter
5 *
6 * \author Simon
7 * \date November 2025
8 *****/
9 #ifndef CLIENT_HPP
10 #define CLIENT_HPP
11
12 #include "Object.hpp"
13 #include "IPlayer.hpp"
14 #include <iostream>
15
16 class Client : public Object
17 {
18 public:
19     /**
20      * \brief Test Function for the Volume Control of the IPlayer interface.
21      *
22      * \param ost Ostream
23      * \param player Reference to the player
24      * \param MaxVolume Maximum Volume of the player
25      * \param DefaultVol Default Volume of the player
26      * \return true -> tests OK
27      * \return false -> tests failed
28     */
29     bool Test_IPlayerVolumeCTRL(std::ostream& ost, IPlayer& player, const size_t& MaxVolume, const size_t& DefaultVol) const;
30
31     /**
32      * \brief Test Play of the Player.
33      *
34      * \param ost Ostream for the Testoutput
35      * \param player Refernce to player
36      * \return true -> tests OK

```

```

37     * \return false -> tests failed
38     */
39     bool Test_IPlayerPlay(std::ostream& ost, IPlayer& player) const;
40
41 /**
42  * \brief Test Play of an empty Player.
43  *
44  * \param ost Ostream for the Testoutput
45  * \param player Reference to player
46  * \return true -> tests OK
47  * \return false -> tests failed
48  */
49     bool Test_IPlayerEmptyPlay(std::ostream& ost, IPlayer& player) const;
50 };
51
52 #endif // !CLIENT_HPP

```

## 6.3 Client.cpp

```

1 #include "Client.hpp"
2 #include "Test.hpp"
3 #include <iostream>
4 #include <algorithm>
5
6 using namespace std;
7
8 bool Client::Test_IPlayerVolumeCTRL(std::ostream& ost, IPlayer& player, const size_t & MaxVolume, const size_t & DefaultVol) const
9 {
10     if (!ost.good()) throw Client::ERROR_BAD_OSTREAM;
11
12     TestStart(ost);
13
14     bool TestOK = true;
15     string error_msg = "";
16
17     try {
18
19         stringstream result;
20
21         std::streambuf* coutbuf = std::cout.rdbuf();
22
23         result << DefaultVol+1;
24         string DVol;
25
26         result >> DVol;
27
28         result.clear();
29         result.str("");
30
31         // cout redirect to stringstream
32         std::cout.rdbuf(result.rdbuf());
33
34         player.VollInc();
35
36         std::cout.rdbuf(coutbuf);
37
38         TestOK == TestOK && check_dump(ost, "Test_Volume_Inc", true, result.str().find(DVol)!=std::string::npos);
39
40         result.clear();
41         result.str("");
42
43         result << DefaultVol;
44
45         result >> DVol;
46
47         result.clear();
48         result.str("");
49

```

```
50     // cout redirect to stringstream
51     std::cout.rdbuf(result.rdbuf());
52
53     player.VollDec();
54
55     std::cout.rdbuf(coutbuf);
56
57     TestOK == TestOK && check_dump(ost, "Test_Volume_Dec", true, result.str().find(DVol)!=std::string::npos);
58
59     // cout redirect to stringstream
60     std::cout.rdbuf(result.rdbuf());
61
62     for (int i = 0;i < 200; i++) player.VollDec();
63
64     player.VollInc();
65
66     std::cout.rdbuf(coutbuf);
67
68     result.clear();
69     result.str("");
70
71     // cout redirect to stringstream
72     std::cout.rdbuf(result.rdbuf());
73
74     player.VollDec();
75
76     std::cout.rdbuf(coutbuf);
77
78     TestOK == TestOK && check_dump(ost, "Test_Lower_Bound_Volume_0", true, result.str().find("0") != std::string::npos);
79
80     // cout redirect to stringstream
81     std::cout.rdbuf(result.rdbuf());
82
83     for (int i = 0;i < 200; i++) player.VollInc();
84
85     std::cout.rdbuf(coutbuf);
86
87     result.clear();
88     result.str("");
89
90     result << MaxVolume;
91
92     string MaxVol;
93
94     result >> MaxVol;
95
96     result.clear();
97     result.str("");
98
99     // cout redirect to stringstream
100    std::cout.rdbuf(result.rdbuf());
101
102    player.VollInc();
103
104    std::cout.rdbuf(coutbuf);
105
106
107    TestOK == TestOK && check_dump(ost, "Test_Upper_Bound_Volume", true, result.str().find(MaxVol) != std::string::npos);
108  }
109
110 catch (const string& err) {
111     error_msg = err;
112     TestOK = false;
113 }
114 catch (bad_alloc const& error) {
115     error_msg = error.what();
116     TestOK = false;
117 }
118 catch (const exception& err) {
119     error_msg = err.what();
120     TestOK = false;
121 }
122 catch (...) {
123     error_msg = "Unhandelt_Exception";
124     TestOK = false;
```

```
125    }
126
127    TestOK == TestOK && check_dump(ost, "Test_for_Exception_in_TestCase", true,error_msg.empty());
128
129    TestEnd(ost);
130
131    if (ost.fail()) throw Client::ERROR_FAIL_WRITE;
132
133    return TestOK;
134 }
135
136 bool Client::Test_IPlayerPlay(std::ostream& ost, IPlayer& player) const
137 {
138    if (!ost.good()) throw Client::ERROR_BAD_OSTREAM;
139
140    TestStart(ost);
141
142    bool TestOK = true;
143    string error_msg = "";
144
145    try {
146
147        stringstream result;
148        std::streambuf* coutbuf = std::cout.rdbuf();
149
150        // cout redirect to stringstream
151        std::cout.rdbuf(result.rdbuf());
152
153        player.Play();
154
155        std::cout.rdbuf(coutbuf);
156
157        TestOK == TestOK && check_dump(ost, "Test_Play_Contains_Name", true, result.str().find("Harry_Potter1") != std::string::npos);
158
159        player.Next();
160
161        result.str("");
162        result.clear();
163
164        std::cout.rdbuf(result.rdbuf());
165
166        player.Play();
167
168        std::cout.rdbuf(coutbuf);
169
170        TestOK == TestOK && check_dump(ost, "Test_Next_", true, result.str().find("Harry_Potter2") != std::string::npos);
171
172        for (int i = 0; i < 4; i++) {
173
174            player.Next();
175
176            result.str("");
177            result.clear();
178
179            std::cout.rdbuf(result.rdbuf());
180
181            player.Play();
182
183            std::cout.rdbuf(coutbuf);
184
185            TestOK == TestOK && check_dump(ost, "Test_Next_", true, result.str().find("Harry_Potter" + 2 + i) != std::string::npos);
186
187        }
188
189        player.Next();
190
191        result.str("");
192        result.clear();
193
194        std::cout.rdbuf(result.rdbuf());
195
196        player.Play();
197
198        std::cout.rdbuf(coutbuf);
199 }
```

```
200     TestOK == TestOK && check_dump(ost, "Test_Next_Wrap_around", true, result.str().find("Harry_Potter1") != std::string::npos);
201
202     result.str("");
203     result.clear();
204
205     std::cout.rdbuf(result.rdbuf());
206
207     player.Select("Harry_Potter3");
208     player.Play();
209
210     std::cout.rdbuf(coutbuf);
211
212     TestOK == TestOK && check_dump(ost, "Test_Select_Video_by_name", true, result.str().find("Harry_Potter3") != std::string::npos);
213
214     result.str("");
215     result.clear();
216
217     std::cout.rdbuf(result.rdbuf());
218
219     player.Select("Harry_Potter14");
220     player.Play();
221
222     std::cout.rdbuf(coutbuf);
223
224     TestOK == TestOK && check_dump(ost, "Test_Select_Video_by_name_not_found", true, result.str().find("not_found!") != std::string::npos);
225
226     result.str("");
227     result.clear();
228
229     std::cout.rdbuf(result.rdbuf());
230
231     player.Select("Harry_Potter3");
232     player.Stop();
233
234     std::cout.rdbuf(coutbuf);
235
236     TestOK == TestOK && check_dump(ost, "Test_Stop_Player",
237         true,
238         result.str().find("stop") != std::string::npos && result.str().find("Harry_Potter3") != std::string::npos);
239
240
241 }
242 catch (const string& err) {
243     error_msg = err;
244     TestOK = false;
245 }
246 catch (bad_alloc const& error) {
247     error_msg = error.what();
248     TestOK = false;
249 }
250 catch (const exception& err) {
251     error_msg = err.what();
252     TestOK = false;
253 }
254 catch (...) {
255     error_msg = "Unhandelt_Exception";
256     TestOK = false;
257 }
258
259 TestOK == TestOK && check_dump(ost, "Test_for_Exception_in_TestCase", true, error_msg.empty());
260
261 TestEnd(ost);
262
263 if (ost.fail()) throw Client::ERROR_FAIL_WRITE;
264
265 return TestOK;
266 }
267
268 bool Client::Test_IPlayerEmptyPlay(std::ostream& ost, IPlayer& player) const
269 {
270     if (!ost.good()) throw Client::ERROR_BAD_OSTREAM;
271
272     TestStart(ost);
273
274     bool TestOK = true;
```

```

275     string error_msg = "";
276
277     try {
278         stringstream result;
279
280         result.str("");
281         result.clear();
282
283         std::streambuf* coutbuf = std::cout.rdbuf();
284
285         std::cout.rdbuf(result.rdbuf());
286
287         player.Play();
288
289         std::cout.rdbuf(coutbuf);
290
291         TestOK == TestOK && check_dump(ost, "Test_for_Message_in_Empty_Player", true, result.str().find("no")!=string::npos);
292
293     }
294     catch (const string& err) {
295         error_msg = err;
296     }
297     catch (bad_alloc const& error) {
298         error_msg = error.what();
299     }
300     catch (const exception& err) {
301         error_msg = err.what();
302     }
303     catch (...) {
304         error_msg = "Unhandelt_Exception";
305     }
306
307     TestOK == TestOK && check_dump(ost, "Test_for_Exception_in_Testcase",true , error_msg.empty());
308
309     TestEnd(ost);
310
311     if (ost.fail()) throw Client::ERROR_FAIL_WRITE;
312
313     return TestOK;
314 }
```

## 6.4 IPlayer.hpp

```

1 #ifndef IPLAYER_HPP
2 #define IPLAYER_HPP
3 //*****
4 * \file IPlayer.hpp
5 * \brief Interface to interact with various Player (music, video)
6 * \author Simon Vogelhuber
7 * \date October 2025
8 ****/
9
10 #include <iostream>
11
12 class IPlayer
13 {
14 public:
15     /**
16      * \brief Play selected song
17      */
18     virtual void Play() = 0;
19
20     /**
21      * \brief increase volume by 1 (out of 100)
22      */
23     virtual void VollInc() = 0;
24
25     /**

```

```

26     * \brief decrease volume by 1 (out of 100)
27     */
28     virtual void VollDec() = 0;
29
30     /**
31     * \brief Stop playing Song
32     */
33     virtual void Stop() = 0;
34
35     /**
36     * \brief Skip to next song
37     */
38     virtual void Next() = 0;
39
40     /**
41     * \brief Skip to previous song
42     */
43     virtual void Prev() = 0;
44
45     /**
46     * \brief Selects a Video by Name.
47     *
48     * \param name
49     */
50     virtual void Select(std::string const& name) = 0;
51
52     /**
53     * \brief virtual Destructor for Interface.
54     *
55     */
56     virtual ~IPlayer() = default;
57
58 };
59
60
61 #endif // !IPLAYER_HPP

```

## 6.5 MusicPlayerAdapter.hpp

```

1 /*****
2  * \file   MusicPlayerAdapter.hpp
3  * \brief
4  *
5  * \author Simon
6  * \date   November 2025
7 ****/
8 #ifndef MUSIC_PLAYER_ADAPTER_HPP
9 #define MUSIC_PLAYER_ADAPTER_HPP
10
11 #include "IPlayer.hpp"
12 #include "MusicPlayer.hpp"
13
14 class MusicPlayerAdapter : public Object, public IPlayer
15 {
16 public:
17
18     MusicPlayerAdapter(MusicPlayer & player) : m_player{ player } {}
19
20     /**
21     * \brief Play selected song
22     */
23     virtual void Play() override;
24
25     /**
26     * \brief increase volume by 1 (out of 100)
27     */
28     virtual void VollInc() override;
29

```

```

30     /**
31      * \brief decrease volume by 1 (out of 100)
32      */
33     virtual void VollDec() override;
34
35     /**
36      * \brief Stop playing Song
37      */
38     virtual void Stop() override;
39
40     /**
41      * \Skip to next song
42      */
43     virtual void Next() override;
44
45     /**
46      * \brief Skip to previous song
47      */
48     virtual void Prev() override;
49
50     /**
51      * \brief Selects a Video by Name.
52      *
53      * \param name
54      */
55     virtual void Select(std::string const& name) override;
56
57 private:
58     MusicPlayer & m_player;
59 };
60
61 #endif // !MUSIC_PLAYER_ADAPTER_HPP

```

## 6.6 MusicPlayerAdapter.cpp

```

1 #include "MusicPlayerAdapter.hpp"
2
3 void MusicPlayerAdapter::Play()
4 {
5     m_player.Start();
6 }
7
8 void MusicPlayerAdapter::VollInc()
9 {
10    m_player.IncreaseVol(1);
11 }
12
13 void MusicPlayerAdapter::VollDec()
14 {
15    m_player.DecreaseVol(1);
16 }
17
18 void MusicPlayerAdapter::Stop()
19 {
20    m_player.Stop();
21 }
22
23 void MusicPlayerAdapter::Next()
24 {
25    m_player.SwitchNext();
26 }
27
28 void MusicPlayerAdapter::Prev()
29 {
30    // The MusicPlayer does not provide a prevSong
31    // function - so we need to skip forward until
32    // we hit the previous song.
33    size_t skipSongs = m_player.GetCount() - 1;

```

```

34     for (int i = 0; i < skipSongs; i++)
35         m_player.SwitchNext();
36     }
37 }
38
39 void MusicPlayerAdapter::Select(std::string const& name)
40 {
41     if (!m_player.Find(name)) std::cout << "song:" << name << "not found!" << std::endl;
42 }
```

## 6.7 MusicPlayer.hpp

```

1 //*****
2 * \file MusicPlayer.hpp
3 * \brief MusicPlayer - A player for music!
4 * \author Simon Vogelhuber
5 * \date October 2025
6 ****
7 #ifndef MUSIC_PLAYER_HPP
8 #define MUSIC_PLAYER_HPP
9
10 #include "Object.hpp"
11 #include "Song.hpp"
12 #include <vector>
13 #include <iostream>
14
15 using SongCollection = std::vector<Song>;
16
17 class MusicPlayer : public Object
18 {
19 public:
20     inline static const std::string ERROR_DURATION_NULL = "ERROR: Song with duration 0!";
21     inline static const std::string ERROR_EMPTY_NAME = "ERROR: Song with empty Name!";
22
23     inline static const std::size_t MAX_VOLUME = 100;
24     inline static const std::size_t MIN_VOLUME = 0;
25     inline static const std::size_t DEFAULT_VOLUME = 50;
26
27     /**
28      * \brief Plays selected song
29      */
30     void Start();
31
32     /**
33      * \brief Stop playing Song
34      */
35     void Stop();
36
37     /**
38      * \brief Skip to next song
39      */
40     void SwitchNext();
41
42     /**
43      * \brief Get index of current song
44      * \return size_t of current's song index
45      */
46     size_t const GetCurIndex() const;
47
48     /**
49      * \brief Find song by name and select it
50      * \return true if song by that name exists
51      */
52     bool Find(std::string const& name);
53
54     /**
55      * \brief Get No. Songs inside the player
56      * \return size_t count of songs inside player
57      */
58 }
```

```

57     size_t const GetCount() const;
58
59     /**
60      * \brief Increase volume by 'vol' amount
61      * \param size_t vol (volume)
62      */
63     void IncreaseVol(size_t const vol);
64
65     /**
66      * \brief Decrease volume by 'vol' amount
67      * \param size_t vol (volume)
68      */
69     void DecreaseVol(size_t const vol);
70
71     /**
72      * \brief Add song to player
73      * \param string name
74      * \param size_t dur (duration)
75      */
76     void Add(std::string const& name, size_t const dur);
77 private:
78     SongCollection m_songs;
79     size_t m_currentSongIdx = 0;
80     size_t m_volume = DEFAULT_VOLUME;
81 };
82
83
84 #endif // !MUSIC_PLAYER_HPP

```

## 6.8 MusicPlayer.cpp

```

1  /**************************************************************************//**
2   * \file MusicPlayer.hpp
3   * \brief MusicPlayer - A player for music!
4   * \author Simon Vogelhuber
5   * \date October 2025
6   *****/
7  #include "MusicPlayer.hpp"
8
9  void MusicPlayer::Start()
10 {
11     if (std::cout.bad()) throw Object::ERROR_BAD_OSTREAM;
12
13     if (m_songs.empty())
14     {
15         std::cout << "no songs in playlist!" << std::endl;
16         return;
17     }
18
19     std::cout
20         << "playing song number " << m_currentSongIdx << ":"_"
21         << m_songs.at(m_currentSongIdx).GetTitle()
22         << "(" << m_songs.at(m_currentSongIdx).GetDuration() << ")" << "\n";
23 }
24
25 void MusicPlayer::Stop()
26 {
27     if (std::cout.bad())
28         throw Object::ERROR_BAD_OSTREAM;
29
30     std::cout
31         << "stop song number " << m_currentSongIdx << ":"_"
32         << m_songs.at(m_currentSongIdx).GetTitle()
33         << "(" << m_songs.at(m_currentSongIdx).GetDuration() << ")" << "\n";
34 }
35
36 void MusicPlayer::SwitchNext()
37 {

```

```

38     // increase until end then wrap around
39     m_currentSongIdx = (m_currentSongIdx + 1) % m_songs.size();
40 }
41
42 size_t const MusicPlayer::GetCurIndex() const
43 {
44     return m_currentSongIdx;
45 }
46
47 bool MusicPlayer::Find(std::string const& name)
48 {
49     if (name.empty()) throw MusicPlayer::ERROR_EMPTY_NAME;
50
51     for (int i = 0; i < m_songs.size(); i++)
52     {
53         if (m_songs.at(i).GetTitle() == name) {
54             m_currentSongIdx = i;
55             return true;
56         }
57     }
58     return false;
59 }
60
61 size_t const MusicPlayer::GetCount() const
62 {
63     return m_songs.size();
64 }
65
66 void MusicPlayer::IncreaseVol(size_t const vol)
67 {
68     if (std::cout.bad())
69         throw Object::ERROR_BAD_OSTREAM;
70
71     m_volume += vol;
72     if (m_volume > MAX_VOLUME)
73         m_volume = MAX_VOLUME;
74
75     std::cout << "volume_is_now->" << m_volume << std::endl;
76 }
77
78 void MusicPlayer::DecreaseVol(size_t const vol)
79 {
80     if (std::cout.bad())
81         throw Object::ERROR_BAD_OSTREAM;
82
83     if (vol > m_volume)
84         m_volume = MIN_VOLUME;
85     else
86         m_volume -= vol;
87
88     std::cout << "volume_is_now->" << m_volume << std::endl;
89 }
90
91 void MusicPlayer::Add(std::string const& name, size_t const dur)
92 {
93     if (name.empty()) throw MusicPlayer::ERROR_EMPTY_NAME;
94     if (dur == 0)    throw MusicPlayer::ERROR_DURATION_NULL;
95
96     m_songs.emplace_back(name, dur);
97 }
```

## 6.9 Song.hpp

```

1 /*****//**
2 * \file Song.hpp
3 * \brief Atomic Class for saving information about a song
4 * \author Simon Vogelhuber
5 * \date October 2025
```

```

6  *****/
7 #ifndef SONG_HPP
8 #define SONG_HPP
9
10 #include "Object.hpp"
11
12 class Song : public Object
13 {
14 public:
15
16     // Exceptions
17     inline static const std::string ERROR_DURATION_NULL = "ERROR: Song with duration 0!";
18     inline static const std::string ERROR_EMPTY_NAME = "ERROR: Song with empty Name!";
19
20     Song(const std::string& name, const size_t& dur);
21     /**
22      * \brief Get title of song
23      * \return string - title of song
24      * \throw ERROR_DURATION_NULL
25      * \throw ERROR_EMPTY_NAME
26      */
27     std::string const& GetTitle() const;
28
29     /**
30      * \brief Get duration of song
31      * \return size_t - duration of song
32      */
33     size_t const GetDuration() const;
34 private:
35     std::string m_name;
36     size_t m_duration;
37 };
38
39 #endif // !SONG_HPP

```

## 6.10 Song.cpp

```

1  *****/
2  * \file Song.cpp
3  * \brief Atomic Class for saving information about a song
4  * \author Simon Vogelhuber
5  * \date October 2025
6  *****/
7
8 #include "Song.hpp"
9
10 Song::Song(const std::string& name, const size_t& dur)
11 {
12     if (name.empty()) throw Song::ERROR_EMPTY_NAME;
13     if (dur == 0)     throw Song::ERROR_DURATION_NULL;
14
15     m_name = name;
16     m_duration = dur;
17 }
18
19 std::string const& Song::GetTitle() const
20 {
21     return m_name;
22 }
23
24 size_t const Song::GetDuration() const
25 {
26     return m_duration;
27 }

```

## 6.11 VideoPlayerAdapter.hpp

## 6.12 VideoPlayerAdapter.cpp

```
1 //*****\n2 * \file VideoPlayerAdapter.hpp\n3 * \brief Adapter for the Video Player in order to Implement IPlayer Interface\n4 *\n5 * \author Simon\n6 * \date November 2025\n7 *****\n8 #include "VideoPlayerAdapter.hpp"\n9\n10 void VideoPlayerAdapter::Play()\n11 {\n12     m_player.Play();\n13 }\n14\n15 void VideoPlayerAdapter::VollInc()\n16 {\n17     m_player.SetVolume(m_player.GetVolume() + 1);\n18 }\n19\n20 void VideoPlayerAdapter::VollDec()\n21 {\n22     if (m_player.GetVolume() != 0) {\n23         m_player.SetVolume(m_player.GetVolume() - 1);\n24     }\n25 }\n26\n27 void VideoPlayerAdapter::Stop()\n28 {\n29     m_player.Stop();\n30 }\n31\n32 void VideoPlayerAdapter::Next()\n33 {\n34     // wrap around if at the end\n35     if (!m_player.Next()) {\n36         m_player.First();\n37     }\n38 }\n39\n40 void VideoPlayerAdapter::Prev()\n41 {\n42     const size_t currIndex = m_player.CurIndex();\n43\n44     if (currIndex == 0) return;\n45\n46     m_player.First();\n47\n48     while (m_player.CurIndex() < (currIndex-1)) m_player.Next();\n49 }\n50\n51 void VideoPlayerAdapter::Select(std::string const& name)\n52 {\n53     size_t prev_index = m_player.CurIndex();\n54\n55     m_player.First();\n56\n57     while (m_player.CurVideo() != name && m_player.Next());\n58\n59     if (m_player.CurVideo() != name) {\n60         std::cout << "video:" << name << "not found!" << std::endl;\n61         // switch back to the previous Video\n62         m_player.First();\n63         while (prev_index != m_player.CurIndex()) m_player.Next();\n64     }\n65 }
```

## 6.13 VideoPlayer.hpp

```
1 //*****
2 * \file VideoPlayer.hpp
3 * \brief Implementation of Video Player of the Company DonkySoft
4 *
5 * \author Simon Offenberger
6 * \date November 2025
7 *****/
8 #ifndef VIDEO_PLAYER_HPP
9 #define VIDEO_PLAYER_HPP
10
11 #include "Object.hpp"
12 #include "Video.hpp"
13 #include <vector>
14 #include <memory>
15 #include <iostream>
16
17 // Using definition of the container
18 using TContVids = std::vector<Video>;
19
20 class VideoPlayer : public Object {
21 public:
22     // defintion of Error Messagges and constance
23     inline static const std::string ERROR_NO_VIDEO_IN_COLLECTION = "ERROR: No video in Player!";
24     inline static const std::string ERROR_DURATION_NULL = "ERROR: Video with duration 0!";
25     inline static const std::string ERROR_EMPTY_NAME = "ERROR: Video with empty Name!";
26
27     inline static const std::size_t MAX_VOLUME = 50;
28     inline static const std::size_t MIN_VOLUME = 0;
29     inline static const std::size_t DEFAULT_VOLUME = 8;
30
31     /**
32      * \brief Starts playing the selected Video.
33      * \throw ERROR_BAD_OSTREAM
34      * \throw ERROR_FAIL_WRITE
35      */
36     void Play() const;
37
38     /**
39      * \brief Stops the selected Video.
40      * \throw ERROR_BAD_OSTREAM
41      * \throw ERROR_FAIL_WRITE
42      */
43     void Stop() const;
44
45     /**
46      * \brief Switches to the first video in the collection.
47      *
48      * \return true -> if videos are in the playlist
49      * \return false -> no video in the playlist
50      */
51     bool First();
52
53     /**
54      * \brief Switches to the next video.
55      *
56      * \return true -> switch was successful
57      * \return false -> no switch possiple index at top of playlist
58      */
59     bool Next();
60
61     /**
62      * \brief returns the current index of the selected video.
63      *
64      * \return Index of the current video
65      * \throw ERROR_NO_VIDEO_IN_COLLECTION
66      */
67     size_t CurIndex() const;
68
69     /**
70      * \brief Get the name of the current video.
71      *
72      * \return String identidier of the video
73
```

```

73     * \throw ERROR_NO_VIDEO_IN_COLLECTION
74     */
75     std::string const CurVideo() const;
76
77 /**
78  * \brief Sets the volume of the player to a specified value.
79
80  * \param vol Volume is bound 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 current 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 private:
105     size_t m_volume = DEFAULT_VOLUME;
106     TContVids m_Videos;
107     size_t m_curIndex = 0;
108 };
110 #endif // !VIDEO_PLAYER_HPP

```

## 6.14 VideoPlayer.cpp

```

1 ****
2 * \file VideoPlayer.cpp
3 * \brief Implementation of Video Player of the Company DonkySoft
4 *
5 * \author Simon Offenberger
6 * \date November 2025
7 ****
8 #include "VideoPlayer.hpp"
9
10 void VideoPlayer::Play() const {
11     if (!std::cout.good()) throw VideoPlayer::ERROR_BAD_OSTREAM;
12     if (m_Videos.empty()) {
13         std::cout << "no video in playlist!" << std::endl;
14         return;
15     }
16
17     std::cout << "playing video number" << CurIndex();
18     std::cout << ":" << CurVideo();
19     std::cout << "[" << m_Videos.at(m_curIndex).GetDuration() << "min]" << std::endl;
20
21     if (std::cout.fail()) throw VideoPlayer::ERROR_FAIL_WRITE;
22 }
23
24 void VideoPlayer::Stop() const {
25     if (!std::cout.good()) throw VideoPlayer::ERROR_BAD_OSTREAM;
26     if (m_Videos.empty()) {
27         std::cout << "no video in playlist!" << std::endl;

```

```
28     return;
29 }
30 std::cout << "stop:video:" << CurVideo();
31 std::cout << "[" << m_Videos.at(m_curIndex).GetDuration() << "min]" << std::endl;
32 if (std::cout.fail()) throw VideoPlayer::ERROR_FAIL_WRITE;
33 }
34
35 bool VideoPlayer::First()
36 {
37     if (m_Videos.empty()) return false;
38     m_curIndex = 0;
39     return true;
40 }
41
42 bool VideoPlayer::Next()
43 {
44     m_curIndex++;
45
46     if (m_curIndex >= m_Videos.size())
47         m_curIndex = m_Videos.size() - 1;
48     return false;
49 }
50 else {
51     return true;
52 }
53
54 }
55
56
57 size_t VideoPlayer::CurIndex() const
58 {
59     if (m_Videos.size()==0) throw VideoPlayer::ERROR_NO_VIDEO_IN_COLLECTION;
60     return m_curIndex;
61 }
62
63 std::string const VideoPlayer::CurVideo() const
64 {
65     if (m_Videos.size()==0) throw VideoPlayer::ERROR_NO_VIDEO_IN_COLLECTION;
66     return m_Videos.at(m_curIndex).GetTitle();
67 }
68
69 void VideoPlayer::SetVolume(const size_t vol)
70 {
71     if (!std::cout.good()) throw VideoPlayer::ERROR_BAD_OSTREAM;
72
73     if (vol > MAX_VOLUME) m_volume = MAX_VOLUME;
74     else m_volume = vol;
75
76     std::cout << "volume_is_now->" << m_volume;
77
78     if (std::cout.fail()) throw VideoPlayer::ERROR_FAIL_WRITE;
79 }
80
81 size_t const VideoPlayer::GetVolume() const
82 {
83     return m_volume;
84 }
85
86 void VideoPlayer::Add(std::string const& name, size_t const dur, EVideoFormat const & format)
87 {
88     if (name.empty()) throw VideoPlayer::ERROR_EMPTY_NAME;
89     if (dur == 0) throw VideoPlayer::ERROR_DURATION_NULL;
90
91     m_Videos.emplace_back(name,dur,format);
92 }
```

## 6.15 Video.hpp

```

1  /*****\file Video.hpp
2  * \brief Implementation of a Video
3  *
4  * \author Simon
5  * \date November 2025
6  *****/
7  #ifndef VIDEO_HPP
8  #define VIDEO_HPP
9
10
11 #include "Object.hpp"
12 #include "EVideoFormat.hpp"
13
14 class Video : public Object
15 {
16 public:
17
18     // Exceptions
19     inline static const std::string ERROR_DURATION_NULL = "ERROR: Video with duration 0!";
20     inline static const std::string ERROR_EMPTY_NAME = "ERROR: Video with empty Name!";
21
22     /**
23      * \brief CTOR of a Video.
24      *
25      * \param title Title of the Video
26      * \param duration Duration of the Video in min
27      * \param format Video Format can be of Type EVideoformat
28      * \throw ERROR_DURATION_NULL
29      * \throw ERROR_EMPTY_NAME
30      */
31     Video(const std::string& title, const size_t& duration, const EVideoFormat& format);
32
33     /**
34      * \brief Getter of the Video Title.
35      *
36      * \return Video Title
37      */
38     const std::string & GetTitle() const;
39
40     /**
41      * \brief Getter of the Video duration
42      *
43      * \return duration of the video
44      */
45     size_t GetDuration() const;
46
47     /**
48      * \brief Getter for the String Identifier of the Format.
49      *
50      * \return String of the Video Format
51      */
52     const std::string GetFormatID() const;
53
54 private:
55     std::string m_title;
56     size_t m_duration;
57     EVideoFormat m_format;
58 };
59
60
61 #endif // !VIDEO_HPP

```

## 6.16 Video.cpp

```

1  /*****\file Video.cpp
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61

```

```

2 * \file Video.hpp
3 * \brief Implementation of a Video
4 *
5 * \author Simon
6 * \date November 2025
7 *****/
8 #include "Video.hpp"
9
10 Video::Video(const std::string& title, const size_t& duration, const EVideoFormat& format)
11 {
12     if (title.empty()) throw Video::ERROR_EMPTY_NAME;
13     if (duration == 0) throw Video::ERROR_DURATION_NULL;
14
15     m_title = title;
16     m_duration = duration;
17     m_format = format;
18 }
19
20 const std::string & Video::GetTitle() const
21 {
22     return m_title;
23 }
24
25 size_t Video::GetDuration() const
26 {
27     return m_duration;
28 }
29
30 const std::string Video::GetFormatID() const
31 {
32     switch (m_format) {
33     case (EVideoFormat::AVI): return "AVI-Format";
34     case (EVideoFormat::MKV): return "MKV-Format";
35     case (EVideoFormat::WMV): return "WMV-Format";
36     default: return "unknownFormat";
37     }
38 }
```

## 6.17 EVideoFormat.hpp

```

1 *****/
2 * \file EVideoFormat.hpp
3 * \brief provides an enum for the Video formats
4 *
5 * \author Simon
6 * \date November 2025
7 *****/
8 #ifndef EVIDEO_FORMAT_HPP
9 #define EVIDEO_FORMAT_HPP
10
11 enum class EVideoFormat
12 {
13     AVI,
14     MKV,
15     WMV
16 };
17
18
19 #endif // !EVIDEO_FORMAT_HPP
```