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, 2. 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

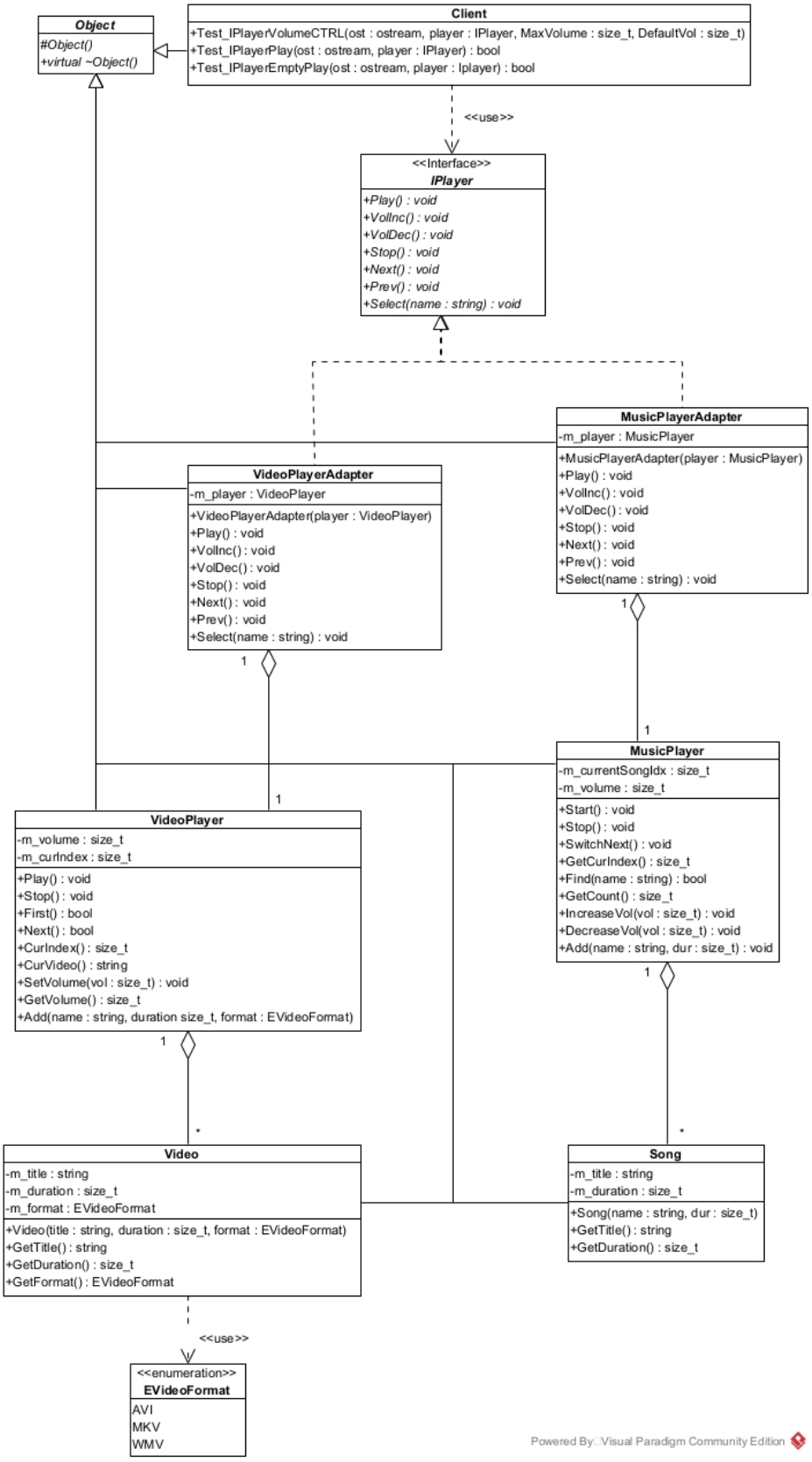
2.3 VideoPlayer Anforderung

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. Dieses 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 müssten die Funktion der Player so angewandt und kombiniert werden, dass für beide Player über das Interface diesselbe 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 FAILED] Result: (Expected: true != Result: false)

Test Volume Dec
[Test FAILED] Result: (Expected: true != Result: false)

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 FAILED] Result: (Expected: true != Result: false)

Test Next
[Test FAILED] Result: (Expected: true != Result: false)

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 FAILED] Result: (Expected: true != Result: false)
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 FAILED] Result: (Expected: true != Result: false)
88
89 Test Volume Dec
90 [Test FAILED] Result: (Expected: true != Result: false)
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 FAILED] Result: (Expected: true != Result: false)

Test Next
[Test FAILED] Result: (Expected: true != Result: false)

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 FAILED] Result: (Expected: true != Result: false)

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

```
#ifndef OBJECT_HPP
#define OBJECT_HPP

#include <string>

class Object {
public:

   inline static const std::string ERROR_BAD_OSTREAM = "ERROR: Provided Ostream is bad";
   inline static const std::string ERROR_FAIL_WRITE = "ERROR: Fail to write on provided Ostream";
   inline static const std::string ERROR_NULLPTR = "ERROR: Passed in Nullptr!";

protected:
   Object() = default;

   virtual ~Object() = default;

};

#endif // !OBJECT_HPP
```

6.2 Client.hpp

```
/*****************************************************************//**
 * \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
```

```
45      * \param player Refernce 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 <sstream>
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
72          // cout redirect to stringstream
73          std::cout.rdbuf(result.rdbuf());
74
75          player.VollDec();
76
77          std::cout.rdbuf(coutbuf);
78
79          TestOK == TestOK && check_dump(ost, "Test␣Lower␣Bound␣Volume␣0", true, result.str().find("0") != std::string::npos);
80
81          // cout redirect to stringstream
82          std::cout.rdbuf(result.rdbuf());
83
84          for (int i = 0;i < 200; i++) player.VollInc();
85
86          std::cout.rdbuf(coutbuf);
87
88          result.clear();
89          result.str("");
90
91          result << MaxVolume;
92
93          string MaxVol;
94
95          result >> MaxVol;
96
97          result.clear();
98          result.str("");
99
100         // cout redirect to stringstream
101         std::cout.rdbuf(result.rdbuf());
102
103         player.VollInc();
104
105         std::cout.rdbuf(coutbuf);
106
107
108         TestOK == TestOK && check_dump(ost, "Test␣Upper␣Bound␣Volume", true, result.str().find(MaxVol) != std::string::npos);
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␣Exceotion␣in␣Test␣Case", 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␣Test␣Case", 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

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

#include <iostream>

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
 *
 * \author Simon
 * \date   November 2025
 *********************************************************************/
#ifndef MUSIC_PLAYER_ADAPTER_HPP
#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     // delete Copy Ctor and Assign Operator to prohibit untestet behaviour
58     MusicPlayerAdapter(MusicPlayerAdapter& Music) = delete;
59     void operator=(MusicPlayerAdapter Music) = delete;
60
61 private:
62     MusicPlayer & m_player;
63 };
64
65 #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()
```

```
{
    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

```
/*****************************************************************//**
 * \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>
#include <iostream>

using  SongCollection = std::vector<Song>;

class MusicPlayer : public Object
{
public:
    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;

    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
```

```
     * \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
     * \param size_t dur (duration)
     */
    void Add(std::string const& name, size_t const dur);

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

private:
    SongCollection m_songs;
    size_t m_currentSongIdx = 0;
    size_t m_volume = DEFAULT_VOLUME;
};


#endif // !MUSIC_PLAYER_HPP
```

6.8 MusicPlayer.cpp

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

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

    if (vol > m_volume)
        m_volume = MIN_VOLUME;
    else
        m_volume -= vol;

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

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

    m_songs.emplace_back(name, dur);
}
```

6.9 Song.hpp

```
/*****************************************************************//**
 * \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"
```

```
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 - duratoin 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
20 std::string const& Song::GetTitle() const
21 {
22     return m_name;
23 }
24
25 size_t const Song::GetDuration() const
26 {
27     return m_duration;
28 }
```

6.11 VideoPlayerAdapter.hpp

```
1  /*****************************************************************//**
2   * \file   VideoPlayerAdapter.hpp
3   * \brief  Adapter for the Video Player in order to Implement IPlayer Interface
4   *
5   * \author Simon
6   * \date   November 2025
7   *********************************************************************/
```

```
8  #ifndef VIDEO_PLAYER_ADAPTER_HPP
9  #define VIDEO_PLAYER_ADAPTER_HPP
10
11 #include "IPlayer.hpp"
12 #include "VideoPlayer.hpp"
13
14 class VideoPlayerAdapter : public Object, public IPlayer
15 {
16 public:
17
18     /**
19      * \brief Construct a VideoPlayer Adapter .
20      *
21      * \param VidPlayer Reference to the actual VideoPlayer
22      */
23     VideoPlayerAdapter(VideoPlayer & VidPlayer) : m_player(VidPlayer) {}
24
25     /**
26      * \brief Play selected song
27      */
28     virtual void Play() override;
29
30     /**
31      * \brief increase volume by 1
32      */
33     virtual void VollInc() override;
34
35     /**
36      * \brief decrease volume by 1
37      */
38     virtual void VollDec() override;
39
40     /**
41      * \brief Stop playing Song
42      */
43     virtual void Stop() override;
44
45     /**
46      * \Skip to next song
47      */
48     virtual void Next() override;
49
50     /**
51      * \brief Skip to previous song
52      */
53     virtual void Prev() override;
54
55     /**
56      * \brief Selects a Video by Name.
57      *
58      * \param name
59      */
60     virtual void Select(std::string const& name) override;
61
62     // delete Copy Ctor and Assign Operator to prohibit untestet behaviour
63     VideoPlayerAdapter(VideoPlayerAdapter& vid) = delete;
64     void operator=(VideoPlayer vid) = delete;
65
66 private:
67     VideoPlayer & m_player;
68 };
69
70 #endif // !MUSIC_PLAYER_ADAPTER_HPP
```

6.12 VideoPlayerAdapter.cpp

```
1  /*****************************************************************//**
2   * \file   VideoPlayerAdapter.hpp
3   * \brief  Adapter for the Video Player in order to Implement IPlayer Interface
4   *
5   * \author Simon
6   * \date   November 2025
7   *********************************************************************/
8  #include "VideoPlayerAdapter.hpp"
9
10 void VideoPlayerAdapter::Play(){
11    m_player.Play();
```

```
12 }
13
14 void VideoPlayerAdapter::VollInc()
15 {
16    m_player.SetVolume(m_player.GetVolume() + 1);
17 }
18
19 void VideoPlayerAdapter::VollDec()
20 {
21    if (m_player.GetVolume() != 0) {
22       m_player.SetVolume(m_player.GetVolume() - 1);
23    }
24 }
25
26 void VideoPlayerAdapter::Stop()
27 {
28    m_player.Stop();
29 }
30
31 void VideoPlayerAdapter::Next()
32 {
33    // wrap around if at the end
34    if (!m_player.Next()) {
35       m_player.First();
36    }
37 }
38
39 void VideoPlayerAdapter::Prev()
40 {
41    const size_t currIndex = m_player.CurIndex();
42
43    if (currIndex == 0) return;
44
45    m_player.First();
46
47    while (m_player.CurIndex() < (currIndex-1)) m_player.Next();
48 }
49
50 void VideoPlayerAdapter::Select(std::string const& name)
51 {
52    size_t prev_index = m_player.CurIndex();
53
54    m_player.First();
55
56    while (m_player.CurVideo() != name && m_player.Next());
57
58    if (m_player.CurVideo() != name) {
59       std::cout << "video:␣" << name << "␣not␣found!" << std::endl;
60       // switch back to the previous Video
61       m_player.First();
62       while (prev_index != m_player.CurIndex())m_player.Next();
63    }
64
65 }
```

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 Messagnes 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    VideoPlayer() = default;
32
33    /**
34     * \brief Starts playing the selected Video.
35     * \throw ERROR_BAD_OSTREAM
36     * \throw ERROR_FAIL_WRITE
37     */
38    void Play() const;
39
40    /**
41     * \brief Stops the selected Video.
42     * \throw ERROR_BAD_OSTREAM
43     * \throw ERROR_FAIL_WRITE
44     */
45    void Stop() const;
46
47    /**
48     * \brief Switches to the first video in the collection.
49     *
50     * \return true -> if videos are in the playlist
51     * \return false -> no video in the playlist
52     */
53    bool First();
54
55    /**
56     * \brief Switches to the next video.
57     *
58     * \return true -> switch was successful
59     * \return false -> no switch possiple index at top of playlist
60     */
61    bool Next();
62
63    /**
64     * \brief returns the current index of the selected video.
65     *
66     * \return Index of the current video
67     * \throw ERROR_NO_VIDEO_IN_COLLECTION
68     */
69    size_t CurIndex() const;
70
71    /**
72     * \brief Get the name of the current video.
73     *
74     * \return String identidier of the video
75     * \throw ERROR_NO_VIDEO_IN_COLLECTION
76     */
77    std::string const CurVideo() const;
78
79    /**
80     * \brief sets the volume of the player to a specified value.
81
82     * \param vol Volume is bond to VideoPlayer::MAX_VOLUME to VideoPlayer::MIN_VOLUME
83     * \throw ERROR_BAD_OSTREAM
84     * \throw ERROR_FAIL_WRITE
85     */
86    void SetVolume(const size_t vol);
87
88    /**
89     * \brief Returns the curreunt volume of the player.
90     *
91     * \return Volume of the player
92     */
93    size_t const GetVolume() const;
94
95    /**
96     * \brief Adds a Video to the VideoPlayer.
97     *
98     * \param name Name of the Video
99     * \param dur Duration of the Video in min
100    * \param format Video Format
101    * \throw ERROR_EMPTY_NAME
102    * \throw ERROR_DURATION_NULL
103    */
```

```
104     void Add(std::string const & name, size_t const dur,EVideoFormat const & format);
105
106     // delete Copy Ctor and Assign Operator to prohibit untestet behaviour
107     VideoPlayer(VideoPlayer& vid) = delete;
108     void operator=(VideoPlayer vid) = delete;
109
110 private:
111     size_t m_volume = DEFAULT_VOLUME;
112     TContVids m_Videos;
113     size_t m_curIndex = 0;
114 };
115
116 #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
31     std::cout << "stop:␣video:␣" << CurVideo();
32     std::cout << "␣[" << m_Videos.at(m_curIndex).GetDuration() << "min]" << std::endl;
33
34     if (std::cout.fail()) throw VideoPlayer::ERROR_FAIL_WRITE;
35 }
36
37 bool VideoPlayer::First()
38 {
39     if (m_Videos.empty()) return false;
40
41     m_curIndex = 0;
42
43     return true;
44 }
45
46 bool VideoPlayer::Next()
47 {
48     m_curIndex++;
49
50     if (m_curIndex >= m_Videos.size()) {
51         m_curIndex = m_Videos.size() - 1;
52         return false;
53     }
54     else {
55         return true;
56     }
57 }
58
59 size_t VideoPlayer::CurIndex() const
60 {
61     if (m_Videos.size()==0) throw VideoPlayer::ERROR_NO_VIDEO_IN_COLLECTION;
```

```
62
63     return  m_curIndex;
64 }
65
66 std::string const VideoPlayer::CurVideo() const
67 {
68     if (m_Videos.size()==0) throw VideoPlayer::ERROR_NO_VIDEO_IN_COLLECTION;
69
70     return m_Videos.at(m_curIndex).GetTitle();
71 }
72
73 void VideoPlayer::SetVolume(const size_t vol)
74 {
75     if (!std::cout.good()) throw VideoPlayer::ERROR_BAD_OSTREAM;
76
77     if (vol > MAX_VOLUME) m_volume = MAX_VOLUME;
78     else              m_volume = vol;
79
80     std::cout << "volume␣is␣now␣->␣" << m_volume;
81
82     if (std::cout.fail()) throw VideoPlayer::ERROR_FAIL_WRITE;
83 }
84
85 size_t const VideoPlayer::GetVolume() const
86 {
87     return m_volume;
88 }
89
90 void VideoPlayer::Add(std::string const& name, size_t const dur, EVideoFormat const & format)
91 {
92     if (name.empty()) throw VideoPlayer::ERROR_EMPTY_NAME;
93     if (dur == 0) throw VideoPlayer::ERROR_DURATION_NULL;
94
95     m_Videos.emplace_back(name,dur,format);
96 }
```

6.15 Video.hpp

```
1  /*****************************************************************//**
2   * \file   Video.hpp
3   * \brief  Implementation of a Video
4   *
5   * \author Simon
6   * \date   November 2025
7   *********************************************************************/
8  #ifndef VIDEO_HPP
9  #define VIDEO_HPP
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 Tilte 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  /*****************************************************************//**
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 "unkown Format";
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
```