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:

```
    //starts playing with the current song in list
    void Start();
    //stops playing
    void Stop();
    //switches to next song and starts at the end with first song
    void SwitchNext();
    //get index of current song
    size_t const GetCurIndex() const;
    //find a song by name in playlist
    bool Find(std::string const& name);
    //get count of songs in playlist
    size_t const GetCount() const;
    //increase the volume relative to the current volume
    void IncreaseVol(size_t const vol);
    //decrease the volume relative to the current volume
    void DecreaseVol(size_t const vol);
    //add a song to playlist
    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, 3. November 2025

Inhaltsverzeichnis

1 Organisatorisches

1.1 Team

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

1.2 Aufteilung der Verantwortlichkeitsbereiche

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

  - Implementierung des Testtreibers
  - Dokumentation

1.3 Aufwand

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

2 Anforderungsdefinition (Systemspezifikation)

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

2.1 IPlayer Interface Anforderung

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

2.2 VideoPlayer Anforderung

Folgende Anforderungen müssen die Methoden des VideoPlayers bereitstellen:

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

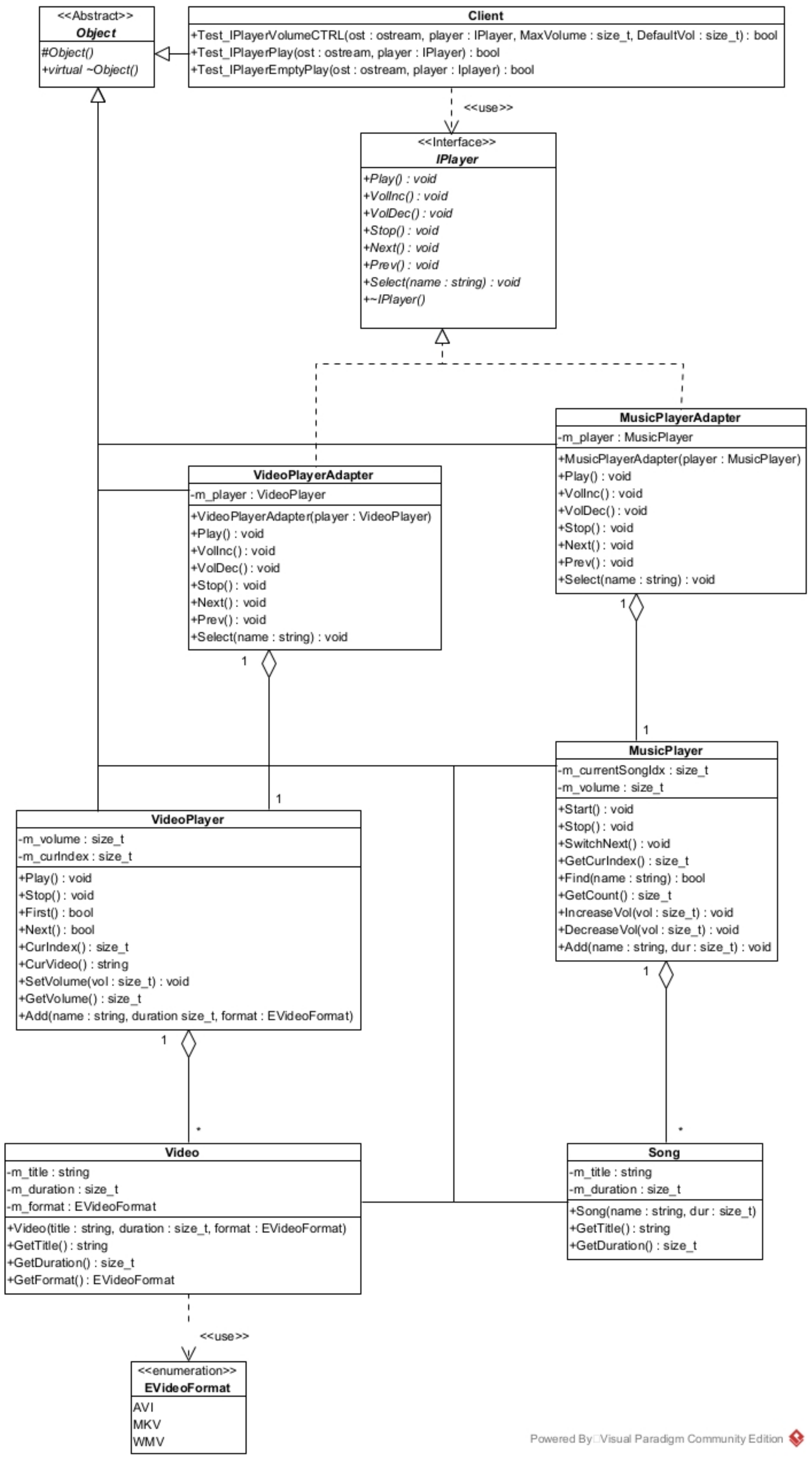
2.3 VideoPlayer Anforderung

Folgende Anforderungen müssen die Methoden des MusicPlayers bereitstellen:

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

3 Systementwurf

3.1 Klassendiagramm



3.2 Designentscheidungen

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

```

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)


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


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

Test for Message in Empty Player
[Test OK] Result: (Expected: true == Result: true)

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


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

Test MusicPlayer 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)

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)


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

```

```

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

Test for Message in Empty Player
[Test OK] Result: (Expected: true == Result: true)

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


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


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

Test Song Getter Duration
[Test OK] Result: (Expected: 123 == Result: 123)

Test Song Getter Name
[Test OK] Result: (Expected: Hello World == Result: Hello World)

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

Test Exception in Song CTOR with duration 0
[Test OK] Result: (Expected: ERROR: Song with duration 0! == Result:
    ↪ ERROR: Song with duration 0!)

Test Exception in Song CTOR with empty string
[Test OK] Result: (Expected: ERROR: Song with empty Name! == Result:
    ↪ ERROR: Song with empty Name!)


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


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

Test Song Getter Duration
[Test OK] Result: (Expected: 123 == Result: 123)

Test Song Getter Name
[Test OK] Result: (Expected: Hello World == Result: Hello World)
```

```

Test Song Getter Format
[Test OK] Result: (Expected: AVI-Format == Result: AVI-Format)

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

Test Exception in Video CTOR with duration 0
[Test OK] Result: (Expected: ERROR: Video with duration 0! == Result:
   ↪ ERROR: Video with duration 0!)

Test Exception in Video CTOR with empty string
[Test OK] Result: (Expected: ERROR: Video with empty Name! == Result:
   ↪ ERROR: Video with empty Name!)


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


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

Test Videplayer Initial Index
[Test OK] Result: (Expected: 0 == Result: 0)

Test Videplayer Index after First
[Test OK] Result: (Expected: 0 == Result: 0)

Test Videplayer Index after Next
[Test OK] Result: (Expected: 1 == Result: 1)

Test Videplayer Index Upper Bound
[Test OK] Result: (Expected: 4 == Result: 4)

Test Videplayer Index after First
[Test OK] Result: (Expected: 0 == Result: 0)

Test Default Volume
[Test OK] Result: (Expected: 8 == Result: 8)

Test Set Volume
[Test OK] Result: (Expected: 25 == Result: 25)

Test Set Volume Max Volume
[Test OK] Result: (Expected: 50 == Result: 50)

Test Set Volume Min Volume
[Test OK] Result: (Expected: 0 == Result: 0)
```

```

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

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

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

Test Exception in Add with empty string
[Test OK] Result: (Expected: ERROR: Video with empty Name! == Result:
    ↪ ERROR: Video with empty Name!)

Test Exception in Add with empty string
[Test OK] Result: (Expected: ERROR: Video with duration 0! == Result:
    ↪ ERROR: Video with duration 0!)


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


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

MusicPlayer - Basic Functionality - .GetCount()
[Test OK] Result: (Expected: 4 == Result: 4)

MusicPlayer - Basic Functionality - .GetIndex() initial
[Test OK] Result: (Expected: 0 == Result: 0)

MusicPlayer - Basic Functionality - .Find() unknown song
[Test OK] Result: (Expected: false == Result: false)

MusicPlayer - Basic Functionality - .Find() song that exists
[Test OK] Result: (Expected: true == Result: true)

MusicPlayer - Basic Functionality - Song name after initial .Start()
[Test OK] Result: (Expected: true == Result: true)

MusicPlayer - Basic Functionality - .GetIndex() after switching
[Test OK] Result: (Expected: 1 == Result: 1)

MusicPlayer - Basic Functionality - Song name switching
[Test OK] Result: (Expected: true == Result: true)

MusicPlayer - Basic Functionality - .GetIndex() wrap around
[Test OK] Result: (Expected: 1 == Result: 1)
```

```

MusicPlayer - Basic Functionality - Error Buffer
[Test OK] Result: (Expected: true == Result: true)

MusicPlayer - Add Song without title
[Test OK] Result: (Expected: ERROR: Song with empty Name! == Result:
    ↪ ERROR: Song with empty Name!)

MusicPlayer - Add Song without title
[Test OK] Result: (Expected: ERROR: Song with duration 0! == Result:
    ↪ ERROR: Song with duration 0!)

MusicPlayer - Add Song without title
[Test OK] Result: (Expected: ERROR: Song with empty Name! == Result:
    ↪ ERROR: Song with empty Name!)

TEST OK!!
```

6 Quellcode

6.1 Object.hpp

```cpp
/*******************************************************************//**
 * \file   Object.hpp
 * \brief  common ancestor for all objects
 *
 * \author Simon
 * \date   November 2025
 *********************************************************************/
#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!";

        // once virtual always virtual
        virtual ~Object() = default;


protected:
        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
         * \param player Refernce to player
         * \return true -> tests OK
         * \return false -> tests failed
         */
        bool Test_IPlayerEmptyPlay(std::ostream& ost, IPlayer& player) const;
};

#endif // !CLIENT_HPP
```

6.3 Client.cpp

```cpp
#include "Client.hpp"
#include "Test.hpp"
#include <sstream>
#include <algorithm>

using namespace std;

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

        TestStart(ost);

        bool TestOK = true;
        string error_msg = "";

        try {

                stringstream result;

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

                result << DefaultVol+1;
                string DVol;

                result >> DVol;

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

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

                player.VollInc();

                std::cout.rdbuf(coutbuf);

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

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

                result << DefaultVol;

                result >> DVol;

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

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

                player.VollDec();

                std::cout.rdbuf(coutbuf);

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

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

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

                player.VollInc();

                std::cout.rdbuf(coutbuf);

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


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

                player.VollDec();

                std::cout.rdbuf(coutbuf);

                TestOK == TestOK && check_dump(ost, "Test␣Lower␣Bound␣Volume␣0", true, result.str().find("0") != std::string::npos);

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

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

                std::cout.rdbuf(coutbuf);

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

                result << MaxVolume;

                string MaxVol;

                result >> MaxVol;

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

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

                player.VollInc();

                std::cout.rdbuf(coutbuf);


                TestOK == TestOK && check_dump(ost, "Test␣Upper␣Bound␣Volume", true, result.str().find(MaxVol) != std::string::npos);
        }
        catch (const string& err) {
                error_msg = err;
                TestOK = false;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
                TestOK = false;
        }
        catch (const exception& err) {
                error_msg = err.what();
                TestOK = false;
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
                TestOK = false;
        }

        TestOK == TestOK && check_dump(ost, "Test␣for␣Exceotion␣in␣Test␣Case", true,error_msg.empty());

        TestEnd(ost);

        if (ost.fail()) throw Client::ERROR_FAIL_WRITE;

        return TestOK;
}

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

        TestStart(ost);

        bool TestOK = true;
        string error_msg = "";

        try {

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

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

                player.Play();

                std::cout.rdbuf(coutbuf);

                TestOK == TestOK && check_dump(ost, "Test␣Play␣Contains␣Name", true, result.str().find("Harry␣Potter1") != std::string::npos);

                player.Next();

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

                std::cout.rdbuf(result.rdbuf());

                player.Play();

                std::cout.rdbuf(coutbuf);

                TestOK == TestOK && check_dump(ost, "Test␣Next␣", true, result.str().find("Harry␣Potter2") != std::string::npos);

                for (int i = 0; i < 4; i++) {

                        player.Next();

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

                        std::cout.rdbuf(result.rdbuf());

                        player.Play();

                        std::cout.rdbuf(coutbuf);

                        TestOK == TestOK && check_dump(ost, "Test␣Next␣", true, result.str().find("Harry␣Potter" + 2 + i) != std::string::npos);

                }

                player.Next();

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

                std::cout.rdbuf(result.rdbuf());

                player.Play();

                std::cout.rdbuf(coutbuf);

                TestOK == TestOK && check_dump(ost, "Test␣Next␣Wrap␣around", true, result.str().find("Harry␣Potter1") != std::string::npos);

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

                std::cout.rdbuf(result.rdbuf());

                player.Select("Harry␣Potter3");
                player.Play();

                std::cout.rdbuf(coutbuf);

                TestOK == TestOK && check_dump(ost, "Test␣Select␣Video␣by␣name␣", true, result.str().find("Harry␣Potter3") != std::string::npos);

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

                std::cout.rdbuf(result.rdbuf());

                player.Select("Harry␣Potter14");
                player.Play();

                std::cout.rdbuf(coutbuf);

                TestOK == TestOK && check_dump(ost, "Test␣Select␣Video␣by␣name␣not␣found", true, result.str().find("not␣found!") != std::string::npos);

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

                std::cout.rdbuf(result.rdbuf());

                player.Select("Harry␣Potter3");
                player.Stop();

                std::cout.rdbuf(coutbuf);

```

```
                TestOK == TestOK && check_dump(ost, "Test␣Stop␣Player",
                                    true,
                              result.str().find("stop") != std::string::npos && result.str().find("Harry␣Potter3")
                                    != std::string::npos);


        }
        catch (const string& err) {
                error_msg = err;
                TestOK = false;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
                TestOK = false;
        }
        catch (const exception& err) {
                error_msg = err.what();
                TestOK = false;
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
                TestOK = false;
        }

        TestOK == TestOK && check_dump(ost, "Test␣for␣Exception␣in␣Test␣Case", true, error_msg.empty());

        TestEnd(ost);

        if (ost.fail()) throw Client::ERROR_FAIL_WRITE;

        return TestOK;
}

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

        TestStart(ost);

        bool TestOK = true;
        string error_msg = "";

        try {
                stringstream result;

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

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

                std::cout.rdbuf(result.rdbuf());

                player.Play();

                std::cout.rdbuf(coutbuf);

                TestOK == TestOK && check_dump(ost, "Test␣for␣Message␣in␣Empty␣Player", true, result.str().find("no"
                     )!=string::npos);

        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK == TestOK && check_dump(ost, "Test␣for␣Exception␣in␣Testcase",true , error_msg.empty());

        TestEnd(ost);

        if (ost.fail()) throw Client::ERROR_FAIL_WRITE;

        return TestOK;
}
```

6.4 IPlayer.hpp

```
#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

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

class MusicPlayerAdapter :public Object, public IPlayer
{
public:

    MusicPlayerAdapter(MusicPlayer & player) : m_player{ player } {}

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

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

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

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

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

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

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

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

private:
    MusicPlayer & m_player;
};

#endif // !MUSIC_PLAYER_ADAPTER_HPP
```

6.6 MusicPlayerAdapter.cpp

```
#include "MusicPlayerAdapter.hpp"

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

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

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

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

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

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

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

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

6.7 MusicPlayer.hpp

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

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

using  SongCollection = std::vector<Song>;

class MusicPlayer : public Object
{
public:
    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
     * \param string name name of the Song
     * \return true if song by that name exists
     */
    bool Find(std::string const& name);

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

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

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

    /**
     * \brief Add song to player
     * \param string name
     * \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"
#include <iostream>

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

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

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

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

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

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

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

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

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

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

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

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

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

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

class Song : public Object
{
public:

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

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

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

#endif // !SONG_HPP
```

6.10 Song.cpp

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

#include "Song.hpp"

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

    m_name = name;
    m_duration = dur;

}

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

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

6.11 VideoPlayerAdapter.hpp

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

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

class VideoPlayerAdapter : public Object, public IPlayer
{
public:

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

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

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

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

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

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

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

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

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

private:
    VideoPlayer & m_player;
};

#endif // !MUSIC_PLAYER_ADAPTER_HPP
```

6.12 VideoPlayerAdapter.cpp

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

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

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

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

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

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

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

        if (currIndex == 0) return;

        m_player.First();

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

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

        m_player.First();

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

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

}
```

6.13 VideoPlayer.hpp

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

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

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

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

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

	VideoPlayer() = default;

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

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

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

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

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

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

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

```

         * \param vol Volume is bond to VideoPlayer::MAX_VOLUME to VideoPlayer::MIN_VOLUME
         * \throw ERROR_BAD_OSTREAM
         * \throw ERROR_FAIL_WRITE
         */
        void SetVolume(const size_t vol);

        /**
         * \brief Returns the curreunt volume of the player.
         *
         * \return Volume of the player
         */
        size_t const GetVolume() const;

        /**
         * \brief Adds a Video to the VideoPlayer.
         *
         * \param name Name of the Video
         * \param dur Duration of the Video in min
         * \param format Video Format
         * \throw ERROR_EMPTY_NAME
         * \throw ERROR_DURATION_NULL
         */
        void Add(std::string const & name, size_t const dur,EVideoFormat const & format);

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

private:
        size_t m_volume = DEFAULT_VOLUME;
        TContVids m_Videos;
        size_t m_curIndex = 0;
};

#endif // !VIDEO_PLAYER_HPP
```

6.14 VideoPlayer.cpp

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

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

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

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

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

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

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

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

	m_curIndex = 0;

	return true;
}

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

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

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

	return  m_curIndex;
}

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

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

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

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

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

```

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

size_t const VideoPlayer::GetVolume() const
{
        return m_volume;
}

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

        m_Videos.emplace_back(name,dur,format);
}
```

6.15 Video.hpp

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

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

class Video : public Object
{
public:

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

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

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

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

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

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


#endif // !VIDEO_HPP
```

6.16 Video.cpp

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

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

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

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

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

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

6.17 EVideoFormat.hpp

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

enum class EVideoFormat
{
        AVI,
        MKV,
        WMV
};


#endif // !EVIDEO_FORMAT_HPP
```

6.18 main.cpp

```
#include "vld.h"
#include "Video.hpp"
#include "VideoPlayer.hpp"
#include "VideoPlayerAdapter.hpp"
#include "MusicPlayer.hpp"
#include "MusicPlayerAdapter.hpp"
#include "Client.hpp"
#include <iostream>
#include <fstream>
#include <cassert>
#include <sstream>
#include "Test.hpp"

using namespace std;

#define WRITE_OUTPUT true

static bool TestSong(ostream& ost);
static bool TestVideo(ostream& ost);
static bool TestVideoPlayer(ostream& ost);
static bool TestMusicPlayer(ostream& ost);

int main(void){

        ofstream testoutput;
        bool TestOK = true;

        try {

                if (WRITE_OUTPUT == true) {
                        testoutput.open("TestOutput.txt");
                }

                VideoPlayer VPlayer;

                VPlayer.Add("Harry␣Potter1", 160, EVideoFormat::AVI);
                VPlayer.Add("Harry␣Potter2", 160, EVideoFormat::AVI);
                VPlayer.Add("Harry␣Potter3", 160, EVideoFormat::AVI);
                VPlayer.Add("Harry␣Potter4", 160, EVideoFormat::AVI);
                VPlayer.Add("Harry␣Potter5", 160, EVideoFormat::AVI);
                VPlayer.Add("Harry␣Potter6", 160, EVideoFormat::AVI);

                VideoPlayerAdapter VidAdapter{ VPlayer };

                Client client;

                cout << "Test␣VideoPlayer␣Adapter␣in␣Client" << endl;
                TestOK = TestOK && client.Test_IPlayerVolumeCTRL(cout,VidAdapter,VideoPlayer::MAX_VOLUME,VideoPlayer
                      ::DEFAULT_VOLUME);
                TestOK = TestOK && client.Test_IPlayerPlay(cout,VidAdapter);

                if (WRITE_OUTPUT) {
                        testoutput << "Test␣VideoPlayer␣Adapter␣in␣Client" << endl;
                        TestOK = TestOK && client.Test_IPlayerVolumeCTRL(testoutput, VidAdapter, VideoPlayer::
                             MAX_VOLUME, VideoPlayer::DEFAULT_VOLUME);
                        TestOK = TestOK && client.Test_IPlayerPlay(testoutput, VidAdapter);
                }

                VideoPlayer EmptyPlayer;
                VideoPlayerAdapter EmptyAdapter { EmptyPlayer };

                TestOK = TestOK && client.Test_IPlayerEmptyPlay(cout, EmptyAdapter);
                if (WRITE_OUTPUT) TestOK = TestOK && client.Test_IPlayerEmptyPlay(testoutput, EmptyAdapter);


                MusicPlayer MPlayer;

                MPlayer.Add("Harry␣Potter1", 160);
                MPlayer.Add("Harry␣Potter2", 160);
                MPlayer.Add("Harry␣Potter3", 160);
                MPlayer.Add("Harry␣Potter4", 160);
                MPlayer.Add("Harry␣Potter5", 160);
                MPlayer.Add("Harry␣Potter6", 160);

                MusicPlayerAdapter MusAdapter{ MPlayer };

                cout << "Test␣MusicPlayer␣Adapter␣in␣Client" << endl;
                TestOK = TestOK && client.Test_IPlayerVolumeCTRL(cout, MusAdapter,MusicPlayer::MAX_VOLUME,
                      MusicPlayer::DEFAULT_VOLUME);
                TestOK = TestOK && client.Test_IPlayerPlay(cout, MusAdapter);

                if (WRITE_OUTPUT) {
                        testoutput << "Test␣MusicPlayer␣Adapter␣in␣Client" << endl;
                        TestOK = TestOK && client.Test_IPlayerVolumeCTRL(testoutput, MusAdapter, MusicPlayer::
                             MAX_VOLUME, MusicPlayer::DEFAULT_VOLUME);
                        TestOK = TestOK && client.Test_IPlayerPlay(testoutput, MusAdapter);
                }

                MusicPlayer EmptyMPlayer;
                MusicPlayerAdapter EmptyMAdapter{ EmptyMPlayer };

                TestOK = TestOK && client.Test_IPlayerEmptyPlay(cout, EmptyMAdapter);
                if (WRITE_OUTPUT) TestOK = TestOK && client.Test_IPlayerEmptyPlay(testoutput, EmptyMAdapter);


                TestOK = TestOK && TestSong(cout);
                if (WRITE_OUTPUT) TestOK = TestOK && TestSong(testoutput);

                TestOK = TestOK && TestVideo(cout);
                if (WRITE_OUTPUT) TestOK = TestOK && TestVideo(testoutput);

                TestOK = TestOK && TestVideoPlayer(cout);
                if (WRITE_OUTPUT) TestOK = TestOK && TestVideoPlayer(testoutput);

                TestOK = TestOK && TestMusicPlayer(cout);
                if (WRITE_OUTPUT) TestOK = TestOK && TestMusicPlayer(testoutput);

                if (WRITE_OUTPUT) {
                        if (TestOK) TestCaseOK(testoutput);
                        else TestCaseFail(testoutput);

                        testoutput.close();
                }

                if (TestOK) TestCaseOK(cout);
                else TestCaseFail(cout);

        }
        catch (const string& err) {
                cerr << err;
        }
        catch (bad_alloc const& error) {
                cerr << error.what();
        }
        catch (const exception& err) {
                cerr << err.what();
        }
        catch (...) {
                cerr << "Unhandelt␣Exception";
        }

        if (testoutput.is_open()) testoutput.close();

        return 0;
}

bool TestSong(ostream& ost)
{
        assert(ost.good());

        TestStart(ost);

        bool TestOK = true;
        string error_msg = "";

        try {

                Song HelloWorld("Hello␣World", 123);

                TestOK = TestOK && check_dump(ost,"Test␣Song␣Getter␣Duration", static_cast<size_t>(123), HelloWorld.
                      GetDuration());

                TestOK = TestOK && check_dump(ost,"Test␣Song␣Getter␣Name",static_cast<string>( "Hello␣World"),
                      HelloWorld.GetTitle());

        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Check␣for␣Exception␣in␣Testcase", true, error_msg.empty());
        error_msg.clear();

        try {
                Song song{ "Hello␣World",0};
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Test␣Exception␣in␣Song␣CTOR␣with␣duration␣0", error_msg, Song::
             ERROR_DURATION_NULL);
        error_msg.clear();

        try {

                Song song{ "",12};

        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Test␣Exception␣in␣Song␣CTOR␣with␣empty␣string", error_msg, Song::
             ERROR_EMPTY_NAME);
        error_msg.clear();

        TestEnd(ost);
        return TestOK;
}


bool TestVideo(ostream& ost)
{
        assert(ost.good());

        TestStart(ost);

        bool TestOK = true;
        string error_msg = "";

        try {

                Video HelloWorld("Hello␣World", 123,EVideoFormat::AVI);

                TestOK = TestOK && check_dump(ost,"Test␣Song␣Getter␣Duration", static_cast<size_t>(123), HelloWorld.
                     GetDuration());

                TestOK = TestOK && check_dump(ost,"Test␣Song␣Getter␣Name",static_cast<string>( "Hello␣World"),
                     HelloWorld.GetTitle());

                TestOK = TestOK && check_dump(ost,"Test␣Song␣Getter␣Format", static_cast<string>("AVI-Format"),
                     HelloWorld.GetFormatID());
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Check␣for␣Exception␣in␣Testcase", true, error_msg.empty());
        error_msg.clear();

        try {
                Video vid{ "Hello␣World",0, EVideoFormat::AVI };
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Test␣Exception␣in␣Video␣CTOR␣with␣duration␣0", error_msg, Video::
              ERROR_DURATION_NULL);
        error_msg.clear();

        try {

                Video vid{ "",12,EVideoFormat::MKV };

        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Test␣Exception␣in␣Video␣CTOR␣with␣empty␣string", error_msg, Video::
              ERROR_EMPTY_NAME);
        error_msg.clear();


        TestEnd(ost);
        return TestOK;
}

bool TestVideoPlayer(ostream& ost)
{
        assert(ost.good());

        TestStart(ost);

        bool TestOK = true;
        string error_msg = "";

        try {

                Video HelloWorld("Hello␣World", 123, EVideoFormat::AVI);

                VideoPlayer VPlayer;

                VPlayer.Add("Hello␣World1",123,EVideoFormat::AVI);
                VPlayer.Add("Hello␣World2",124,EVideoFormat::MKV);
                VPlayer.Add("Hello␣World3",125,EVideoFormat::WMV);
                VPlayer.Add("Hello␣World4",126,EVideoFormat::AVI);
                VPlayer.Add("Hello␣World5",127,EVideoFormat::MKV);

                TestOK = TestOK && check_dump(ost, "Test␣Videplayer␣Initial␣Index", static_cast<size_t>(0), VPlayer.
                      CurIndex());

                VPlayer.First();

                TestOK = TestOK && check_dump(ost, "Test␣Videplayer␣Index␣after␣First", static_cast<size_t>(0),
                     VPlayer.CurIndex());

                VPlayer.Next();

                TestOK = TestOK && check_dump(ost, "Test␣Videplayer␣Index␣after␣Next", static_cast<size_t>(1),
                     VPlayer.CurIndex());

                for (int i = 0; i < 100;i++) VPlayer.Next();

                TestOK = TestOK && check_dump(ost, "Test␣Videplayer␣Index␣Upper␣Bound", static_cast<size_t>(4),
                     VPlayer.CurIndex());

                VPlayer.First();

                TestOK = TestOK && check_dump(ost, "Test␣Videplayer␣Index␣after␣First", static_cast<size_t>(0),
                     VPlayer.CurIndex());

                TestOK = TestOK && check_dump(ost, "Test␣Default␣Volume", static_cast<size_t>(8), VPlayer.GetVolume
                     ());

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

                stringstream result;

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

                VPlayer.SetVolume(25);

                std::cout.rdbuf(coutbuf);

                TestOK = TestOK && check_dump(ost, "Test␣Set␣Volume", static_cast<size_t>(25), VPlayer.GetVolume());

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

                VPlayer.SetVolume(300);

                std::cout.rdbuf(coutbuf);

                TestOK = TestOK && check_dump(ost, "Test␣Set␣Volume␣Max␣Volume", static_cast<size_t>(VideoPlayer::
                     MAX_VOLUME), VPlayer.GetVolume());

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

                VPlayer.SetVolume(0);

                std::cout.rdbuf(coutbuf);

                TestOK = TestOK && check_dump(ost, "Test␣Set␣Volume␣Min␣Volume", static_cast<size_t>(VideoPlayer::
                     MIN_VOLUME), VPlayer.GetVolume());


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

                VPlayer.Play();

                std::cout.rdbuf(coutbuf);

                TestOK = TestOK && check_dump(ost, "Test␣Video␣Player␣Play", true, result.str().find(VPlayer.
                     CurVideo()) != string::npos);

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

                VPlayer.Stop();

                std::cout.rdbuf(coutbuf);

                TestOK = TestOK && check_dump(ost, "Test␣Video␣Player␣Stop", true, result.str().find("stop") !=
                     string::npos);
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Check␣for␣Exception␣in␣Testcase", true, error_msg.empty());
        error_msg.clear();

        try{
                VideoPlayer VidPlayer;
                VidPlayer.Add("", 123, EVideoFormat::AVI);

        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Test␣Exception␣in␣Add␣with␣empty␣string", error_msg, VideoPlayer::
             ERROR_EMPTY_NAME);
        error_msg.clear();

        try{
                VideoPlayer VidPlayer;
                VidPlayer.Add("234", 0, EVideoFormat::AVI);

        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Test␣Exception␣in␣Add␣with␣empty␣string", error_msg, VideoPlayer::
             ERROR_DURATION_NULL);
        error_msg.clear();


        TestEnd(ost);
        return TestOK;
}

bool TestMusicPlayer(ostream& ost)
{
        assert(ost.good());

        TestStart(ost);

        bool TestOK = true;
        string error_msg = "";

        // test basic functionality
        try {
                MusicPlayer music;
                std::string const song1 = "How␣much␣is␣the␣Fish␣-␣Scooter";
                std::string const song2 = "Die␣Blume␣aus␣dem␣Gemeindebau␣-␣Wolfgang␣Ambros";
                std::string const song3 = "Red␣Sun␣in␣the␣Sky␣-␣MaoZe";
                std::string const song4 = "Ski-Twist␣-␣Hansi␣Hinterseer";
                size_t const dur1 = 300;
                size_t const dur2 = 240;
                size_t const dur3 = 180;
                size_t const dur4 = 110;
                size_t const songCount = 4;
                music.Add(song1, dur1);
                music.Add(song2, dur2);
                music.Add(song3, dur3);
                music.Add(song4, dur4);

                TestOK = TestOK && check_dump(ost, "MusicPlayer␣-␣Basic␣Functionality␣-␣.GetCount()", music.GetCount(), songCount);
                TestOK = TestOK && check_dump(ost, "MusicPlayer␣-␣Basic␣Functionality␣-␣.GetIndex()␣initial", music.GetCurIndex(), static_cast<size_t>(0));
                TestOK = TestOK && check_dump(ost, "MusicPlayer␣-␣Basic␣Functionality␣-␣.Find()␣unknown␣song", music.Find("not␣a␣real␣song"), false);
                TestOK = TestOK && check_dump(ost, "MusicPlayer␣-␣Basic␣Functionality␣-␣.Find()␣song␣that␣exists", music.Find(song1), true);

                // for checking cout
                std::streambuf* coutbuf = std::cout.rdbuf();
                stringstream result;

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

                TestOK = TestOK && check_dump(ost, "MusicPlayer␣-␣Basic␣Functionality␣-␣Song␣name␣after␣initial␣.Start()", true, result.str().find(song1) != string::npos);
                result.str("");
                result.clear();

                music.SwitchNext();
                std::cout.rdbuf(result.rdbuf());
                music.Start();
                std::cout.rdbuf(coutbuf);

                TestOK = TestOK && check_dump(ost, "MusicPlayer␣-␣Basic␣Functionality␣-␣.GetIndex()␣after␣switching", static_cast<size_t>(1), music.GetCurIndex());
                TestOK = TestOK && check_dump(ost, "MusicPlayer␣-␣Basic␣Functionality␣-␣Song␣name␣switching", true, result.str().find(song2) != string::npos);
                result.str("");
                result.clear();

                // wrap around
                for (int i = 0; i < music.GetCount(); i++)
                {
                        music.SwitchNext();
                }

                std::cout.rdbuf(result.rdbuf());
                music.Stop();
                std::cout.rdbuf(coutbuf);

                TestOK = TestOK && check_dump(ost, "MusicPlayer␣-␣Basic␣Functionality␣-␣.GetIndex()␣wrap␣around", static_cast<size_t>(1), music.GetCurIndex());
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "MusicPlayer␣-␣Basic␣Functionality␣-␣Error␣Buffer", true, error_msg.empty());
        error_msg.clear();

        // Add empty song
        try {
                MusicPlayer music;
                std::string const song = "";
                size_t const dur = 1;
                music.Add(song, dur);
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "MusicPlayer␣-␣Add␣Song␣without␣title", MusicPlayer::ERROR_EMPTY_NAME,
              error_msg);
        error_msg.clear();

        // Add song with 0 duration
        try {
                MusicPlayer music;
                std::string const song = "This␣is␣a␣legit␣song";
                size_t const dur = 0;
                music.Add(song, dur);
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "MusicPlayer␣-␣Add␣Song␣without␣title", MusicPlayer::ERROR_DURATION_NULL,
               error_msg);
        error_msg.clear();

        // find empty name
        try {
                MusicPlayer music;
                music.Find("");
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "MusicPlayer␣-␣Add␣Song␣without␣title", MusicPlayer::ERROR_EMPTY_NAME,
              error_msg);
        error_msg.clear();

        return TestOK;
}
```

6.19 Test.hpp

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

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

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

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

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

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

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

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

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

        return ost;
}

```

```
inline std::ostream& TestCaseFail(std::ostream& ost) {

#if COLOR_OUTPUT
        if (ost.good()) {
                ost << colorRed << "TEST␣FAILED␣!!" << colorWhite << std::endl;

        }
#else
        if (ost.good()) {
                ost << "TEST␣FAILED␣!!" << std::endl;

        }
#endif // COLOR_OUTPUT

        return ost;
}

/**
        * \brief function that reports if the testcase was successful.
        *
        * \param testcase       String that indicates the testcase
        * \param succsessful true -> reports to cout test OK
        * \param succsessful false -> reports test failed
        */
template <typename T>
bool check_dump(std::ostream& ostr, const std::string& testcase, const T& expected, const T& result) {
        if (ostr.good()) {
#if COLOR_OUTPUT
                if (expected == result) {
                        ostr << testcase << std::endl <<  colorGreen << "[Test␣OK]␣" << colorWhite <<"Result:␣(
                             Expected:␣" << std::boolalpha << expected << "␣==" << "␣Result:␣" << result << ")"  <<
                              std::noboolalpha << std::endl << std::endl;
                }
                else {
                        ostr << testcase << std::endl << colorRed << "[Test␣FAILED]␣" << colorWhite << "Result:␣(
                             Expected:␣" << std::boolalpha << expected << "␣!=" << "␣Result:␣" << result << ")"  <<
                              std::noboolalpha << std::endl << std::endl;
                }
#else
                if (expected == result) {
                        ostr << testcase << std::endl << "[Test␣OK]␣"  << "Result:␣(Expected:␣" << std::boolalpha <<
                              expected << "␣==" << "␣Result:␣" << result << ")" << std::noboolalpha << std::endl <<
                              std::endl;
                }
                else {
                        ostr << testcase << std::endl  << "[Test␣FAILED]␣"  << "Result:␣(Expected:␣" << std::
                             boolalpha << expected << "␣!=" << "␣Result:␣" << result << ")" << std::noboolalpha <<
                             std::endl << std::endl;
                }
#endif
                if (ostr.fail()) {
                        std::cerr << "Error:␣Write␣Ostream" << std::endl;
                }
        }
        else {
                std::cerr << "Error:␣Bad␣Ostream" << std::endl;
        }
        return expected == result;
}

template <typename T1, typename T2>
std::ostream& operator<< (std::ostream& ost,const std::pair<T1,T2> & p) {
        if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
        ost << "(" << p.first << "," << p.second << ")";
        return ost;
}

template <typename T>
std::ostream& operator<< (std::ostream& ost,const std::vector<T> & cont) {
        if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
        std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>{ost, "␣"});
        return ost;
}

template <typename T>
std::ostream& operator<< (std::ostream& ost,const std::list<T> & cont) {
        if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
        std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>{ost, "␣"});
        return ost;
}

template <typename T>
std::ostream& operator<< (std::ostream& ost,const std::deque<T> & cont) {
        if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
        std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>{ost, "␣"});
        return ost;
}

template <typename T>
std::ostream& operator<< (std::ostream& ost,const std::forward_list<T> & cont) {
        if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
        std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>{ost, "␣"});
        return ost;
}


#endif // !TEST_HPP
```