SEMINAR 4

Contents

1. OBJECTIVES

- Use inheritance and polymorphism to solve a problem.
- Define and use derived classes.
- Use method overriding and dynamic binding.

2. REMEMBER THESE

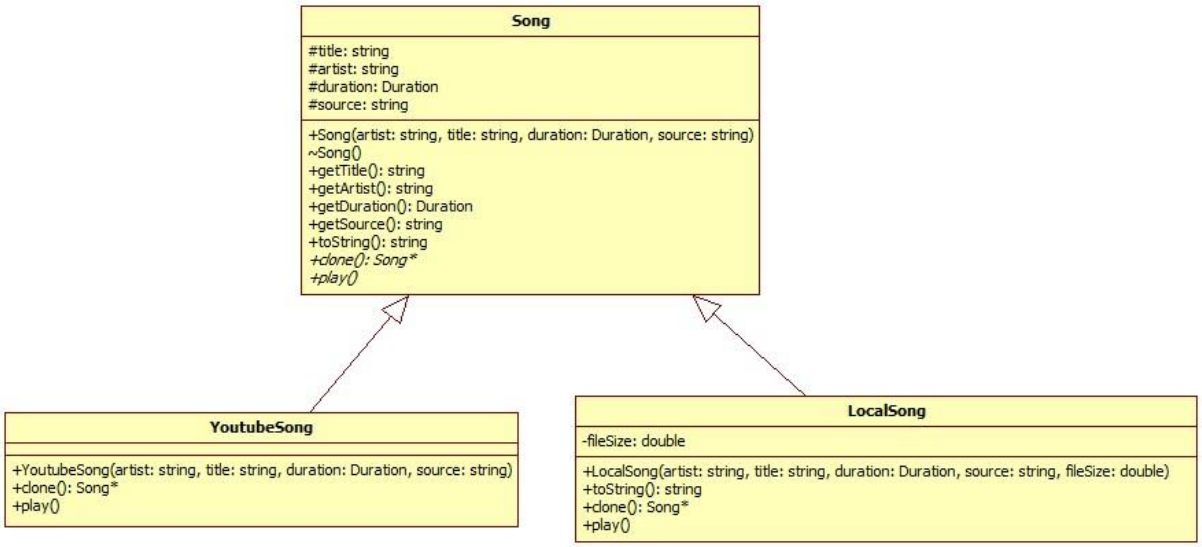
- The constructor of a derived class must explicitly call the constructor of the base class, otherwise C++ will call the default constructor.
- Virtual constructors are not allowed, because to access a virtual method, the object must be already constructed.
- If a base class has defined any type of constructor, the compiler will not generate a default constructor for the derived class.
- Pay attention to the virtual keyword
- Destructors of a base class must be virtual.

3. PROBLEM STATEMENT

Extend your playlist application such that it can handle songs stores locally. Every stored song has the following information: artist, title, duration (minutes and seconds), absolute path towards the file and size of the file on disk.

Both your song repository and your playlist can contain youtube songs, as well as songs stored locally. When the playlist is being played, the youtube songs should be played using chrome, while the locally stored songs should be played using a local application (which might not necessarily be the default music player in your operating system).

4. UML DIAGRAM



5. SOURCE CODE

----------Song.h----------

```
#pragma once
#include <iostream>
#include <string>

class Song
{
protected:
	std::string title;
	std::string artist;
	Duration duration;
	std::string source;		// might be a youtube link or a local file

public:
	// default constructor for a song
	Song();

	// constructor with parameters
	Song(const std::string& artist, const std::string& title, const Duration&
duration, const std::string& source);

	virtual ~Song() {}
```

```
        std::string getTitle() const { return title; }
        std::string getArtist() const { return artist; }
        Duration getDuration() const { return duration; }
        std::string getSource() const { return source; }

        // pure virtual function which will "know" how to clone a Song
        virtual Song* clone() const = 0;

        // converts a Song to its string representation
        virtual std::string toString() const;

        // pure virtual function which "knows" how to play a Song
        virtual void play() const = 0;
```

----------Song.cpp----------

```
#include "Song.h"
#include <sstream>

using namespace std;

Song::Song(): title(""), artist(""), duration(Duration()), source("") {}

Song::Song(const string& artist, const string& title, const Duration& duration, const
string& source)
{
        this->artist = artist;
        this->title = title;
        this->duration = duration;
        this->source = source;
}

string Song::toString() const
{
        stringstream s;
        s << this->artist << " - " << this->title << ", " << this->duration.getMinutes()
<< ":" << this->duration.getSeconds() << endl;
        s << "Source: " << this->source << endl;
        return s.str();
}
```

----------YoutubeSong.h----------

```
#pragma once
#include "Song.h"

class YoutubeSong: public Song
{
public:
        YoutubeSong(const std::string& artist, const std::string& title, const Duration&
duration, const std::string& source);

        /*
                Overrides the "play" function in the base class.
```

```
		Input: -
		Output: plays a youtube song, using chrome.
	*/
	void play() const override;

	/*
		Overrides the "clone" function in the base class. This function will know
how to clone a youtube song.
		Input: -
		Output: returns a new Youtube song, containing the same information as the
current song.
	*/
	Song* clone() const override;
};
```

----------YoutubeSong.cpp----------

```
#include "YoutubeSong.h"
#include <Windows.h>

YoutubeSong::YoutubeSong(const std::string& artist, const std::string& title, const
Duration& duration, const std::string& source) : Song(artist, title, duration, source)
{
}

void YoutubeSong::play() const
{
	ShellExecuteA(NULL, NULL, "chrome.exe", this->getSource().c_str(), NULL,
SW_SHOWMAXIMIZED);
}

Song* YoutubeSong::clone() const
{
	return new YoutubeSong(*this);
}
```

----------LocalSong.h----------

```
#pragma once
#include "Song.h"

class LocalSong: public Song
{
private:
	double fileSize;
public:
	LocalSong(const std::string& artist, const std::string& title, const Duration&
duration, const std::string& source, double fileSize);

	/*
		Overrides the "play" function in the base class.
		Input: -
		Output: plays a locally stored song, using Windows Media Player.
	*/
	void play() const override;
```

```
        std::string toString() const override;

        /*
                Overrides the "clone" function in the base class. This function will know
how to clone a youtube song.
                Input: -
                Output: returns a new local song, containing the same information as the
current song.
        */
        Song* clone() const override;
};
```

----------LocalSong.cpp----------

```
#include "LocalSong.h"
#include <Windows.h>
#include <sstream>

using namespace std;

LocalSong::LocalSong(const std::string& artist, const std::string& title, const Duration&
duration, const std::string& source, double fileSize) : Song(artist, title, duration,
source)
{
        this->fileSize = fileSize;
}

void LocalSong::play() const
{
        ShellExecuteA(NULL, NULL, "C:\\Program Files (x86)\\Windows Media
Player\\wmplayer.exe", this->getSource().c_str(), NULL, SW_SHOWMAXIMIZED);
}

string LocalSong::toString() const
{
        stringstream s;
        s << "Size: " << this->fileSize << " MB." << endl;
        return Song::toString() + s.str();
}

Song* LocalSong::clone() const
{
        return new LocalSong(*this);
}
```

----------Repository.h----------

```
#pragma once
#include "Song.h"
#include <vector>

class Repository
{
private:
        std::vector<Song*> songs;
        std::string filename;
```

```
public:
	Repository() {}

	// Copy constructor - is needed, as the Repository contains a vector of pointers.
	Repository(const Repository& r);

	// Destructor - will deallocate the memory allocated when adding songs to the
vector of pointers.
	~Repository();

	/*
		Adds a song to the repository.
		Input: s - Song.
		Output: the song is added to the repository.
	*/
	void addSong(Song* s);

	/*
		Finds a song, by artist and title.
		Input: artist, title - string
		Output: the song that was found, or an null pointer, if nothing was found.
	*/
	Song* findByArtistAndTitle(const std::string& artist, const std::string& title);

	std::vector<Song*> getSongs() const { return songs; }
};
```

----------Repository.cpp----------

```
#include "Repository.h"
#include <string>
#include <algorithm>
#include "Utils.h"
#include "YoutubeSong.h"
#include "LocalSong.h"

using namespace std;

Repository::Repository(const Repository& r)
{
	for (auto s : r.songs)
		this->songs.push_back(s->clone());
}

Repository::~Repository()
{
	for (auto s : this->songs)
		delete s;
}

void Repository::addSong(Song* s)
{
	// create a clone of the song (allocate memory and copy the song) and add this to
the repository
	Song* song = s->clone();
	this->songs.push_back(song);
```

```
}

Song* Repository::findByArtistAndTitle(const std::string& artist, const std::string&
title)
{
	auto it = find_if(this->songs.begin(), this->songs.end(), [artist, title](Song* s)
{return (s->getArtist() == artist && s->getTitle() == title); });
	if (it != songs.end())
		return *it;
	return nullptr;
}
```

----------Controller.h----------

```
#pragma once
#include "Repository.h"
#include "PlayList.h"

class Controller
{
private:
	Repository repo;
	PlayList playList;

public:
	Controller(Repository r) : repo(r) {}

	Repository& getRepo() { return repo; }
	PlayList getPlaylist() const { return playList; }

	// Adds a song with the given data to the song repository.
	void addSongToRepository(const std::string& type, const std::string& artist, const
std::string& title, double minutes, double seconds, const std::string& source, double
size = 0);

	/*
		Adds a given song to the current playlist.
		Input: song - Song, the song must belong to the repository.
		Output: the song is added to the playlist.
	*/
	void addSongToPlaylist(Song* song);

	// Adds all the songs from the repository, that have the given artist, to the
current playlist.
	void addAllSongsByArtistToPlaylist(const std::string& artist);

	void startPlaylist();
	void nextSongPlaylist();
};
```

----------Controller.cpp----------

```
#include "Controller.h"
#include "YoutubeSong.h"
#include "LocalSong.h"
#include <algorithm>
#include <iterator>
```

```
using namespace std;

void Controller::addSongToRepository(const string& type, const string& artist, const
string& title, double minutes, double seconds, const string& source, double size)
{
	// create a new song, with the given type and information
	Song* s = nullptr;
	if (type == "youtubeSong")
		s = new YoutubeSong(artist, title, Duration(minutes, seconds), source);
	else
		s = new LocalSong(artist, title, Duration(minutes, seconds), source, size);
	this->repo.addSong(s);

	// after the song was added to the repository, it can be deleted
	delete s;
}

void Controller::addSongToPlaylist(Song* song)
{
	this->playList.add(song);
}

void Controller::addAllSongsByArtistToPlaylist(const std::string& artist)
{
	// get all the songs from the repository
	vector<Song*> v = this->repo.getSongs();
	vector<Song*> songsByArtist;
	copy_if(v.begin(), v.end(), back_inserter(songsByArtist), [artist](Song* s) {
return s->getArtist() == artist; });
	for (auto s : songsByArtist)
		this->playList.add(s);
}

void Controller::startPlaylist()
{
	this->playList.play();
}

void Controller::nextSongPlaylist()
{
	this->playList.next();
}
```

The entire source code can be found on the OOP web page, Seminar 4 – Week 1.