Qontainer Project Report

Federico Omodei, mat. 1126500

06/21/2019

Object Oriented Programming course, final project.

Indice

1 Introduction

1.1 Compiling instructions

Project compilation requires a different project file (.pro) from the standard one. The right Qontainer.pro file is already given in the folder[1], so the only commands to invoke are
qmake
make

1.2 Development environment

- Operating System: Manjaro Linux (4.14.94)
- C++ compiler: gcc 8.2
- Qt library: 5.12
- Editor: Geany 1.34
- IDE: Qt Creator
- Version control: git
- Repository hosting: `https://gitlab.com`
- Testing: lab OS on VirtualBox

1.3 Abstract

The software represents an archive for video files with a graphical interface to show its content. It guarantees basic functionalities such as video files insertion, removal and search within the personal library, and provides also the persistence of data through save and load functionalities.

[1]In case of need, the command to generate the .pro file is: *qmake -project "QT+=widgets" "CONFIG+=c++11"*

2 Hierarchy description

2.1 Model

The software's logic is based upon VideoFile base class, from which other classes inherit general infos and features. Specializations of the VideoFile class are Movie, Anime, TvSerie and Match classes. There is only one level of inheritance as shown on the schema below:

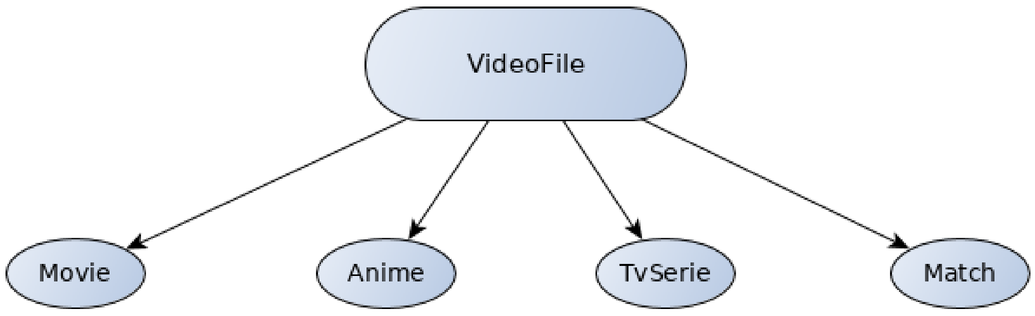


Figura 1: Types hierarchy schema.

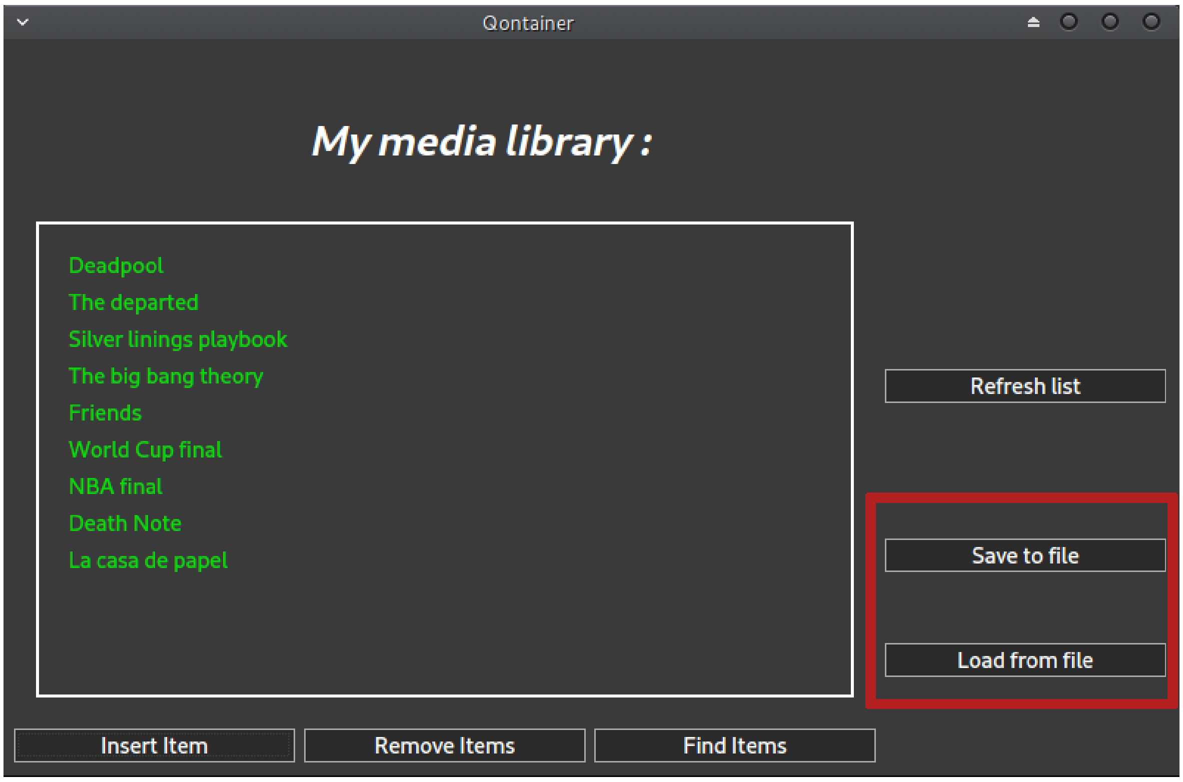
- **VideoFile class:** is the base class of the hierarchy. It represents generic video files such as *amateurs holiday's film* and is characterized by a *title, a genre, a nation* and *a publishing year*.
- **Movie class:** child of VideoFile class, represents cinematographic movies and is characterized by every detail of a generic video file plus a *director* and a *length* expressed in minutes.
- **Anime class:** child of VideoFile class, represents Japanese animation series and is characterized by the *number of episodes* and the fact that it is still ongoing or not.
- **TvSerie class:** child of VideoFile class, represents tv series characterized by *number of seasons* and the fact that is still ongoing or not.
- **SportMatch class:** child of VideoFile class, represents a recording of a sport match, characterized by a belonging *championship*, a *home team* and a *guest team*.

2.2 View

The GUI has been made with maximum component-reuse in mind. In fact every custom widget, a part from MainWindow, is a refinement and a specialization of another one. Examples are the widget needed to insert an object on Qontainer, that is most part like the one to modify an object; also the insertion and find ones share most of the widgets, preventing exceeding code duplication.

3 Saving and loading from file

Loading items from file into Qontainer or saving items from Qontainer to file is possible thanks to a custom utility file, named *LoadSave.cpp*, that contains the functions needed by the application. The chosen format for keeping data safe and persistent even after program quits is *JSON*. The file, named *savedLibrary.json*, already contains a few default objects and is placed in *C++/* folder.
Using these functionalities is incredibly simple and intuitive, it takes only to click on the dedicated buttons on bottom-right corner, as shown in picture:



4 Polymorphism

Pointers belonging to application's main custom container (*Qontainer<VideoFile*>*) can assume polymorphic behavior as they can be used to interact with child classes' objects.
RTTI[2] and polymorphic behavior can be found in these points of the software:

- **View/modifyWidget.cpp:** into *activateFields()* function, depending on the object pointed by *VideoFile *vid* GUI widgets dynamically change their state.
- **View/modifyWidget.cpp:** into *confirmModify()* function, depending on the object pointed by *VideoFile *vid* container objects are dynamically modified.
- **Model/Qontainer.h:** into searching functions such as *searchByDirector()*, searching fields are dynamically chosen at run-time, depending on the pointed object.
- **LoadSave.cpp:** into *saveToFile()* function, object type to be saved in JSON file is identified by RTTI.
- **View/MainWindow.cpp:** into *windowSelector()* function, QPushButto *button pointer recognize at run-time which type of widget to create, thanks to RTTI.

[2]Run-Time Type Identification

5 Project log

Completing the project required **49 hours** in total, divided as follows:

- Problem analysis: 2h;
- Model and template design: 7h;
- Qt learning and GUI design: 13h;
- Coding: 23,5h;
- Testing: 3,5h .