1. Ministry of Education and Science of the Russian Federation
2. Peter the Great St. Petersburg Polytechnic University
3. —
4. Institute of Applied Mathematics and Mechanics.
5. **Department of Information Security of Computer Systems**
6. **Laboratory work №2**
7. **USING MVC PATTERN WHEN DEVELOPING**
8. **GRAPHIC APPLICATIONS**

on the discipline " Object-oriented programming"

1. Completed
2. a student group 33636/1 A.V. Malinko
3. Teacher A.Y. Chernov
4. Saint Petersburg
5. 2018

**Objective**

Acquaintance with the concept of patterns and their classification, study principles and purpose of patterns “Strategy”, “Linker”, “Observer” and MVC, gaining practical skills of pattern application when designing an application as well as using tools Qt libraries for GUI development.

**Theory**

As with other software patterns, MVC expresses the "core of the solution" to a problem while allowing it to be adapted for each system. Particular MVC architectures can vary significantly from the traditional description here.

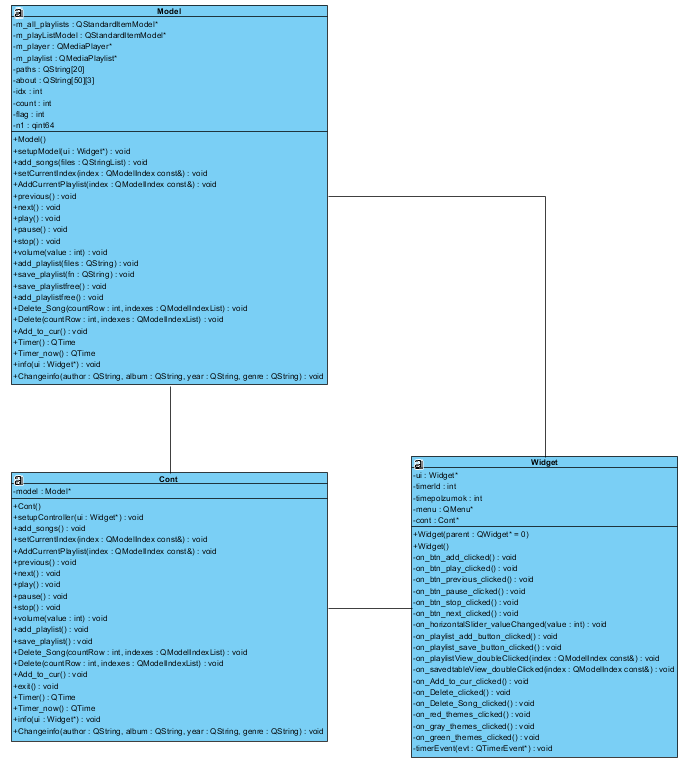
**Components**

* The *model* is the central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application.
* A *view* can be any output representation of information, such as a chart or a diagram. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.
* The third part or section, the *controller*, accepts input and converts it to commands for the model or view.

**Execution process**

In order to comply with the MVС pattern, I added two new classes - the controller and the model. The view already exists when creating a new project and is located in the .ui and widget.сpp files. When pressing the buttons in the form, the controller methods are invoked, in which the necessary processing and preparation of information is performed, and then the model methods are called. Inside the model is the main part of the player.

**UML class diagram**



**Conclusion**

While working on this task, I studied the basics of QT, learned about the MVC and was able to write a fully functional music player. The main difficulty nevertheless arose in the study and creation of a MVC model. I also spent a lot of time exploring ways to change tags in mp3 files, but unfortunately I did not succeed. In general, I find the work done useful and quite interesting.

**Program listing**

#include "widget.h"

#include "cont.h"

#include <QTime>

#include <QtWinExtras>

#include <QMenu>

Widget::Widget(QWidget \*parent) :

QWidget(parent),

ui(new Ui::Widget),

cont(new Cont()),

timepolzumok(0)

{

timerld = startTimer(1000);

ui->setupUi(this);

cont->setupController(ui);

menu = new QMenu(this);

}

Widget::~Widget()

{

cont->exit();

delete ui;

}

void Widget::on\_btn\_add\_clicked()

{

cont->add\_songs();

}

void Widget::on\_btn\_previous\_clicked()

{

cont->previous();

}

void Widget::on\_btn\_play\_clicked()

{

cont->play();

}

void Widget::on\_btn\_pause\_clicked()

{

cont->pause();

}

void Widget::on\_btn\_stop\_clicked()

{

cont->stop();

ui->horizontalSlider\_2->setValue(0);

}

void Widget::on\_btn\_next\_clicked()

{

cont->next();

}

void Widget::on\_horizontalSlider\_valueChanged(int value)

{

cont->volume(value);

}

void Widget::on\_playlist\_add\_button\_clicked()

{

cont->add\_playlist();

}

void Widget::on\_playlist\_save\_button\_clicked()

{

cont->save\_playlist();

}

void Widget::on\_playlistView\_doubleClicked(const QModelIndex &index)

{

cont->setCurrentIndex(index);

}

void Widget::on\_savedtableView\_doubleClicked(const QModelIndex &index)

{

cont->AddCurrentPlaylist(index);

}

void Widget::on\_Add\_to\_cur\_clicked()

{

cont->Add\_to\_cur();

}

void Widget::on\_Delete\_clicked()

{

**Widget.cpp**

QModelIndexList indexes = ui->savedtableView->selectionModel()->selectedRows();

int countRow = indexes.count();

cont->Delete(countRow, indexes);

}

void Widget::on\_Delete\_Song\_clicked()

{

QModelIndexList indexes = ui->playlistView->selectionModel()->selectedRows();

int countRow = indexes.count();

cont->Delete\_Song(countRow, indexes);

}

void Widget::on\_red\_themes\_clicked()

{

setStyleSheet(QString("QPushButton\

{background-color: white;\

border-style: solid;\

border-width:1px;\

border-radius:12px;\

border-color: red;\

max-width:100px;\

max-height:100px;\

min-width:25px;\

min-height:25px;}\

QTableView\

{selection-background-color: qlineargradient(x1: 0, y1: 0, x2: 0.5, y2: 0.5,\

stop: 0 #FF92BB, stop: 1 #E0FFFF);\

background-color: #FFC0CB;}\

QSlider::groove:horizontal \

{background: red;\

position: absolute; \

left: 4px; right: 4px;}\

QSlider::handle:horizontal {\

height: 10px;\

background: gray;\

margin: 0 -4px;}\

QSlider::add-page:horizontal {\

background: white;}\

QSlider::sub-page:horizontal {\

background: pink;}"));

}

void Widget::on\_gray\_themes\_clicked()

{

//setStyleSheet(("border-image:url(./rocket.png);");

setStyleSheet(("QPushButton{background-image: url(:/buttons/kiss.png);}\

QTableView{background-image: url(:/buttons/kiss.png);}"));

}

void Widget::on\_green\_themes\_clicked()

{

setStyleSheet(QString("QPushButton {default style};\

QTableView {default style};"));

}

void Widget::timerEvent(QTimerEvent \*evt)

{

if (evt->timerId() == timerld)

{

QTime datatime = cont->Timer();

QString data = datatime.toString();

timepolzumok = 0;

timepolzumok += datatime.minute()\*60 + datatime.second();

ui->all\_time->setText(data);

QTime time = cont->Timer\_now();

ui->now\_time->setText(time.toString());

int value = time.second();

if (value!=0){

int bala = value/(timepolzumok/100);

ui->horizontalSlider\_2->setValue(bala);

}

}

cont->info(ui);

}

**Cont.cpp**

#include <QFileDialog>

#include <QDir>

#include <QString>

#include <Qtime>

#include "cont.h"

#include "model.h"

Cont::Cont():

model(new Model())

{

}

void Cont::setupController(Ui::Widget \*ui)

{

model->setupModel(ui);

}

void Cont::add\_songs()

{

QStringList files = QFileDialog::getOpenFileNames(nullptr,

("Open files"),QString(), ("Audio Files (\*.mp3)"));

model->add\_songs(files);

}

void Cont::setCurrentIndex(const QModelIndex &index)

{

model->setCurrentIndex(index);

}

void Cont::previous()

{

model->previous();

}

void Cont::next()

{

model->next();

}

void Cont::play()

{

model->play();

}

void Cont::pause()

{

model->pause();

}

void Cont::stop()

{

model->stop();

}

void Cont::volume(int value)

{

model->volume(value);

}

void Cont::add\_playlist()

{

QString files = QFileDialog::getOpenFileName(nullptr,"Open file",QDir::currentPath(),

"Playlist files (\*.m3u)");

model->add\_playlist(files);

}

void Cont::AddCurrentPlaylist(const QModelIndex &index)

{

model->AddCurrentPlaylist(index);

}

void Cont::save\_playlist()

{

QString fn = QFileDialog::getSaveFileName(nullptr,"Save file",QDir::currentPath(),

"Playlist files (\*.m3u)");

model->save\_playlist(fn);

}

void Cont::Delete\_Song(int countRow, QModelIndexList indexes)

{

model->Delete\_Song(countRow, indexes);

}

void Cont::Delete(int countRow, QModelIndexList indexes)

{

model->Delete(countRow, indexes);

}

void Cont::Add\_to\_cur()

{

model->Add\_to\_cur();

}

void Cont::exit()

{

model->save\_playlistfree();

}

QTime Cont::Timer(){

QTime ret = model->Timer();

return ret;

}

QTime Cont::Timer\_now(){

QTime ret = model->Timer\_now();

return ret;

}

void Cont::info(Ui::Widget \*ui){

model->info(ui);

}

**Model.cpp**

#include <QDir>

#include <cstdlib>

#include <QTextStream>

#include <QFile>

#include <QTime>

#include <iostream>

#include <fstream>

#include "model.h"

#include "widget.h"

#include <QMediaMetaData>

#include <QtMultimedia/qmetadatawritercontrol.h>

#define GET\_VARIABLE\_NAME(Variable) (#Variable)

Model::Model():

m\_all\_playlists(new QStandardItemModel()),

m\_playListModel(new QStandardItemModel()),

m\_player(new QMediaPlayer()),

m\_playlist(new QMediaPlaylist(m\_player)),

idx(0),

count(),

flag(0),

n1(0)

{

}

void Model::setupModel(Ui::Widget \*ui){

ui->playlistView->setModel(m\_playListModel);

m\_playListModel->setHorizontalHeaderLabels(QStringList() << ("Audio Track")

<< ("File Path"));

ui->playlistView->hideColumn(1); //

ui->playlistView->verticalHeader()->setVisible(false); //

ui->playlistView->setSelectionBehavior(QAbstractItemView::SelectRows);

ui->playlistView->setSelectionMode(QAbstractItemView::SingleSelection);

ui->playlistView->setEditTriggers(QAbstractItemView::NoEditTriggers);

TableView

ui->playlistView->horizontalHeader()->setStretchLastSection(true);

m\_player->setPlaylist(m\_playlist);

m\_player->setVolume(70); //

m\_player->positionChanged(100);

m\_playlist->setPlaybackMode(QMediaPlaylist::Loop);

/\*------------------------------------------\*/

ui->savedtableView->setModel(m\_all\_playlists);

m\_all\_playlists->setHorizontalHeaderLabels(QStringList() << ("Playlists")

<< ("File Path"));

ui->savedtableView->hideColumn(1); //

ui->savedtableView->verticalHeader()->setVisible(false);

ui->savedtableView->setSelectionBehavior(QAbstractItemView::SelectRows);

ui->savedtableView->setSelectionMode(QAbstractItemView::SingleSelection);

ui->savedtableView->setEditTriggers(QAbstractItemView::NoEditTriggers); TableView

ui->savedtableView->horizontalHeader()->setStretchLastSection(true);

add\_playlistfree();

QFile("D:/Player/autosaved/aut.m3u").remove();

QFile("D:/Player/autosaved/aut.txt").remove();

}

void Model::add\_songs(QStringList files){

foreach (QString filePath, files) {

QList<QStandardItem \*> items;

items.append(new QStandardItem(QDir(filePath).dirName()));

items.append(new QStandardItem(filePath));

m\_playListModel->appendRow(items);

m\_playlist->addMedia(QUrl(filePath));

}

count++;

}

void Model::setCurrentIndex(const QModelIndex &index)

{

m\_playlist->setCurrentIndex(index.row());

}

void Model::previous()

{

m\_playlist->previous();

}

void Model::next()

{

m\_playlist->next();

}

void Model::play()

{

m\_player->play();

}

void Model::pause()

{

m\_player->pause();

}

void Model::stop()

{

m\_player->stop();

}

void Model::volume(int value)

{

m\_player->setVolume(value);

}

void Model::add\_playlist(QString files)

{

QList<QStandardItem \*> items;

items.append(new QStandardItem(files));

m\_all\_playlists->appendRow(items);

paths[idx] = files;

idx++;

}

void Model::AddCurrentPlaylist(const QModelIndex &index)

{

if (flag==0){

m\_playlist->clear();

}

while (m\_playListModel->rowCount() > 0)

m\_playListModel->removeRow(0);

m\_playlist->load(QUrl::fromLocalFile(paths[index.row()]),"m3u");

for (int i=0;i<m\_playlist->mediaCount();i++)

{

QList<QStandardItem \*> items;

items.append(new QStandardItem(QDir(m\_playlist->media(i).canonicalUrl().toString()).dirName()));

items.append(new QStandardItem(m\_playlist->media(i).canonicalUrl().toString()));

m\_playListModel->appendRow(items);

}

flag = 0;

}

void Model::save\_playlist(QString fn)

{

m\_playlist->save(QUrl::fromLocalFile(fn),"m3u");

}

void Model::save\_playlistfree()

{

QString aut = "D:/Player/autosaved/aut.m3u";

m\_playlist->save(QUrl::fromLocalFile(aut), "m3u");

QFile file("D:/Player/autosaved/aut.txt");

int i=0;

//for (i=0; i<idx; i++) paths[i].chop(1);

i=0;

QTextStream out(&file);

if (file.open((QIODevice::WriteOnly| QIODevice::Text))){

while (i<idx) {

int u=0;

while(1){

if((paths[i][u]=='3')&&(paths[i][u+1]=='u')){

out << paths[i][u];

u++;

if (i==(idx-1)) out << paths[i][u];

else out << paths[i][u]<< "\n";

break;

}

else {

out << paths[i][u];

u++;

}

}

i++;

}

//i--;

}

file.close();

QFile file2("D:/Player/autosaved/autinf.txt");

i=0;

QTextStream out2(&file2);

if (file2.open((QIODevice::WriteOnly| QIODevice::Text))){

while (i < count) {

out2 << about[i][0]<< "\n" << about[i][1]<< "\n" << about[i][2]<< "\n" ;

i++;

}

}

file2.close();

}

void Model::add\_playlistfree()

{

QString aut = "D:/Player/autosaved/aut.m3u";

m\_playlist->load(QUrl::fromLocalFile(aut),"m3u");

for (int i=0;i<m\_playlist->mediaCount();i++)

{

QList<QStandardItem \*> items;

items.append(new QStandardItem(QDir(m\_playlist->media(i).canonicalUrl().toString()).dirName()));

items.append(new QStandardItem(m\_playlist->media(i).canonicalUrl().toString()));

m\_playListModel->appendRow(items);

}

QFile file("D:/Player/autosaved/aut.txt");

int i=0;

if ((file.exists())&&(file.open(QIODevice::ReadOnly))){

while (!file.atEnd()) {

QString temp;

paths[i] = file.readLine();

int u=0;

while(1){

if((paths[i][u]=='3')&&(paths[i][u+1]=='u')){

if (paths[i][u+2]==0x0d)

paths[i].chop(2);

break;

}

else

u++;

}

i++;

}

idx=i;

i=0;

file.close();

for (i=0;i<idx;i++){

QList<QStandardItem \*> items2;

items2.append(new QStandardItem(paths[i]));

m\_all\_playlists->appendRow(items2);

}

}

QFile file2("D:/Player/autosaved/autinf.txt");

i=0;

if ((file2.exists())&&(file2.open(QIODevice::ReadOnly))){

while (!file2.atEnd()) {

for (int j=0; j<3; j++){

about[i][j] = file2.readLine();

about[i][j].chop(2);

}

i++;

}

}

count=i;

i=0;

file2.close();

}

void Model::Delete\_Song(int countRow, QModelIndexList indexes)

{

m\_playListModel->removeRow( indexes.at(countRow-1).row(), QModelIndex());

m\_playlist->removeMedia(indexes.at(countRow-1).row());

count--;

}

void Model::Delete(int countRow, QModelIndexList indexes)

{

m\_all\_playlists->removeRow( indexes.at(countRow-1).row(), QModelIndex());

for (int i=indexes.at(countRow-1).row(); i<idx-1; i++){

paths[i] = paths[i+1];

}

idx--;

}

void Model::Add\_to\_cur()

{

flag=1;

}

QTime Model::Timer(){

n1 = m\_player->duration();

int h = ( n1 / (60\*60\*1000));

int m = (( n1 % (60\*60\*1000)) / (60\*1000));

int s = ((n1 % (60\*1000)) /1000);

return QTime(h,m,s);

}

QTime Model::Timer\_now(){

QTime time(0,0,0,0);

time = time.addMSecs(m\_player->position());

return time;

}

void Model::info(Ui::Widget \*ui){

if (m\_player->isMetaDataAvailable()) {

QString author = m\_player->metaData(QStringLiteral("Author")).toString();

QString year = m\_player->metaData(QStringLiteral("Year")).toString();

QString genre = m\_player->metaData(QStringLiteral("Genre")).toString();

ui->label\_author->setText(author);

ui->label\_year->setText(year);

ui->label\_genre->setText(genre);

}

}