|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

|  |  |
| --- | --- |
| **ФАКУЛЬТЕТ** | **ИУК «Информатика и управление»** |
| **КАФЕДРА** | **ИУК4 «Программное обеспечение ЭВМ,** |
| **информационные технологии»** | |

**ЛАБОРАТОРНАЯ РАБОТА №6**

**«Модульное тестирование программного продукта»**

**ДИСЦИПЛИНА: «Основы программной инженерии»**



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Выполнил: студент гр. ИУК4-21Б | |  |  | ( | Суриков Н.С. | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |
| Проверил: | |  |  | ( | Амеличев Г. Э. | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |

|  |  |
| --- | --- |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | |
|  | - Балльная оценка:  - Оценка: |

**Цель:** получить практические навыки разработки модульных тестов кода, написанного на языке С++ с применением средств Unit Test(QtTest).

**Задачи**: написать набор юнит тестов, для разработанных ранее функций.

**Программный продукт:** Отельный бизнес

**Пройденные тесты:**



**Вывод:** в ходе данной лабораторной работы были получены практические навыки разработки пользовательских интерфейсов на этапе проектирования программного продукта средствами QT.

**Листинг программы:**

#include <QtTest/QtTest>

#include "DataBase.h"

class DataBaseTest : public QObject

{

Q\_OBJECT

private slots:

void initTestCase()

{

m\_database = std::make\_unique<DataBase>("test\_database.csv");

}

void cleanupTestCase()

{

m\_database.reset();

QFile::remove("test\_database.csv");

}

void testLoadRecords()

{

QCOMPARE(m\_database->getRecords().size(), size\_t(0));

m\_database->addRecord(HotelRoom(1, m\_database->getRoomTypes().at("Стандартный"), "2023-06-01", "2023-06-03", 2, 4000.0, ""));

m\_database->addRecord(HotelRoom(2, m\_database->getRoomTypes().at("Люкс"), "2023-07-01", "2023-07-05", 4, 28000.0, ""));

QCOMPARE(m\_database->getRecords().size(), size\_t(2));

}

void testGetRecord()

{

m\_database->addRecord(HotelRoom(1, m\_database->getRoomTypes().at("Стандартный"), "2023-06-01", "2023-06-03", 2, 4000.0, ""));

HotelRoom\* room = m\_database->getRecord(1);

QVERIFY(room != nullptr);

QCOMPARE(room->getId(), 1);

QCOMPARE(room->getRoomType().getName(), "Стандартный");

QCOMPARE(room->getCheckInDate(), "2023-06-01");

QCOMPARE(room->getCheckOutDate(), "2023-06-03");

QCOMPARE(room->getNumGuests(), 2);

QCOMPARE(room->getTotalCost(), 4000.0);

QCOMPARE(room->getNotes(), "");

}

void testAddRecord()

{

m\_database->addRecord(HotelRoom(1, m\_database->getRoomTypes().at("Стандартный"), "2023-06-01", "2023-06-03", 2, 4000.0, ""));

QCOMPARE(m\_database->getRecords().size(), size\_t(1));

}

void testDeleteRecord()

{

m\_database->addRecord(HotelRoom(1, m\_database->getRoomTypes().at("Стандартный"), "2023-06-01", "2023-06-03", 2, 4000.0, ""));

m\_database->deleteRecord(1);

QCOMPARE(m\_database->getRecords().size(), size\_t(0));

}

void testEditRecord()

{

m\_database->addRecord(HotelRoom(1, m\_database->getRoomTypes().at("Стандартный"), "2023-06-01", "2023-06-03", 2, 4000.0, ""));

HotelRoom newRoom(1, m\_database->getRoomTypes().at("Люкс"), "2023-07-01", "2023-07-05", 4, 28000.0, "");

m\_database->editRecord(1, newRoom);

HotelRoom\* room = m\_database->getRecord(1);

QVERIFY(room != nullptr);

QCOMPARE(room->getId(), 1);

QCOMPARE(room->getRoomType().getName(), "Люкс");

QCOMPARE(room->getCheckInDate(), "2023-07-01");

QCOMPARE(room->getCheckOutDate(), "2023-07-05");

QCOMPARE(room->getNumGuests(), 4);

QCOMPARE(room->getTotalCost(), 28000.0);

QCOMPARE(room->getNotes(), "");

}

private:

std::unique\_ptr<DataBase> m\_database;

};

QTEST\_APPLESS\_MAIN(DataBaseTest)

#include "DataBaseTest.moc"

#include "QtUiTools"

#include "SetupUi.h"

#include <boost/algorithm/string/classification.hpp>

#include <boost/algorithm/string/split.hpp>

#include "QWidget"

#include "string"

#include "Functions.h"

#include "Deposit.h"

#include "unordered\_map"

#include "boost/container/string.hpp"

using namespace std;

static QWidget \*loadUiFile(QWidget \*parent, const std::string &path) {

QString qPath = QString::fromStdString(path);

QFile file(qPath);

file.open(QIODevice::ReadOnly);

QUiLoader loader;

return loader.load(&file, parent);

}

void Ui::ChangeTheme(QPushButton \*button) {

if (button->text() == "Светлая тема") {

QApplication::setStyle("windowsvista");

button->setText("Темная тема");

} else {

QApplication::setStyle("fusion"); // fusion other option (win11 currently bugged)

button->setText("Светлая тема");

}

}

void Ui::DrawTable() {

QTableWidget \*Table = windows["TableWindow"]->findChild<QTableWidget \*>("Table");

qDebug() << "tavle init" << Table->rowCount();

for (int i = 0; i < Table->rowCount(); ++i) {

Table->setRowHidden(i, false);

}

Table->sortByColumn(0, Qt::AscendingOrder);

Table->clear();

Table->setColumnCount(11);

Table->setHorizontalHeaderItem(0, new QTableWidgetItem("Отчет"));

Table->setHorizontalHeaderItem(1, new QTableWidgetItem("Логин"));

Table->setHorizontalHeaderItem(2, new QTableWidgetItem("И.Ф."));

Table->setHorizontalHeaderItem(3, new QTableWidgetItem("Телефон"));

Table->setHorizontalHeaderItem(4, new QTableWidgetItem("Email"));

Table->setHorizontalHeaderItem(5, new QTableWidgetItem("Тип вклада"));

Table->setHorizontalHeaderItem(6, new QTableWidgetItem("Срок в месяцах"));

Table->setHorizontalHeaderItem(7, new QTableWidgetItem("Сумма"));

Table->setHorizontalHeaderItem(8, new QTableWidgetItem("Процент"));

Table->setHorizontalHeaderItem(9, new QTableWidgetItem("Доход"));

Table->setHorizontalHeaderItem(10, new QTableWidgetItem(""));

Table->setRowCount(Deposits->size());

qDebug() << Deposits->size() << "Table init";

for (int i = 0; i < Deposits->size(); i++) {

QTableWidgetItem \*Login = new QTableWidgetItem(QString::fromStdString(Deposits->at(i)->getLogin()));

QTableWidgetItem \*Name\_Surname = new QTableWidgetItem(

QString::fromStdString(Deposits->at(i)->getName\_Surname()));

QTableWidgetItem \*Phone = new QTableWidgetItem(QString::fromStdString(Deposits->at(i)->getPhone()));

QTableWidgetItem \*Email = new QTableWidgetItem(QString::fromStdString(Deposits->at(i)->getEmail()));

QTableWidgetItem \*Type = new QTableWidgetItem(QString::fromStdString(Deposits->at(i)->getType()));

QTableWidgetItem \*Time = new QTableWidgetItem;

Time->setData(Qt::EditRole, Deposits->at(i)->getTimeInMonths());

QTableWidgetItem \*Amount = new QTableWidgetItem;

Amount->setData(Qt::EditRole, Deposits->at(i)->getAmount());

QTableWidgetItem \*Percent = new QTableWidgetItem;

Percent->setData(Qt::EditRole, Deposits->at(i)->getPercent());

QTableWidgetItem \*Income = new QTableWidgetItem;

Income->setData(Qt::EditRole, Deposits->at(i)->getIncome());

QCheckBox \*Report = new QCheckBox("");

Table->setCellWidget(i, 0, Report);

Table->setItem(i, 1, Login);

Table->setItem(i, 2, Name\_Surname);

Table->setItem(i, 3, Phone);

Table->setItem(i, 4, Email);

Table->setItem(i, 5, Type);

Table->setItem(i, 6, Time);

Table->setItem(i, 7, Amount);

Table->setItem(i, 8, Percent);

Table->setItem(i, 9, Income);

QPushButton \*DeleteButton = new QPushButton("Удалить");

Table->setCellWidget(i, 10, DeleteButton);

QObject::connect(DeleteButton, &QPushButton::clicked, [=, this]() {

QPushButton \*DeleteButton2 = windows["DeletingConfirmWindow"]->findChild<QPushButton \*>("DeleteButton");

QPushButton \*CancelButton = windows["DeletingConfirmWindow"]->findChild<QPushButton \*>("CancelButton");

QLabel \*LoginLabel = windows["DeletingConfirmWindow"]->findChild<QLabel \*>("LoginLabel");

LoginLabel->setText(Login->text());

QLabel \*NSLabel = windows["DeletingConfirmWindow"]->findChild<QLabel \*>("NSLabel");

NSLabel->setText(Name\_Surname->text());

QLabel \*PhoneLabel = windows["DeletingConfirmWindow"]->findChild<QLabel \*>("PhoneLabel");

PhoneLabel->setText(Phone->text());

QLabel \*EmailLabel = windows["DeletingConfirmWindow"]->findChild<QLabel \*>("EmailLabel");

EmailLabel->setText(Email->text());

QLabel \*TypeLabel = windows["DeletingConfirmWindow"]->findChild<QLabel \*>("TypeLabel");

TypeLabel->setText(Type->text());

QLabel \*TimeLabel = windows["DeletingConfirmWindow"]->findChild<QLabel \*>("TimeLabel");

TimeLabel->setText(Time->text());

QLabel \*AmountLabel = windows["DeletingConfirmWindow"]->findChild<QLabel \*>("AmountLabel");

AmountLabel->setText(Amount->text());

QLabel \*PercentLabel = windows["DeletingConfirmWindow"]->findChild<QLabel \*>("PercentLabel");

PercentLabel->setText(Percent->text());

QLabel \*IncomeLabel = windows["DeletingConfirmWindow"]->findChild<QLabel \*>("IncomeLabel");

IncomeLabel->setText(Income->text());

windows["DeletingConfirmWindow"]->show();

QObject::connect(DeleteButton2, &QPushButton::clicked, [=, this]() {

qDebug() << "Deleting" << LoginLabel->text();

DepositFunctions::Delete(Deposits, Deposits->at(i)->getLogin());

DepositFunctions::SaveData(Deposits);

DrawTable();

windows["DeletingConfirmWindow"]->close();

});

QObject::connect(CancelButton, &QPushButton::clicked, [=, this]() {

windows["DeletingConfirmWindow"]->close();

});

});

}

Table->setColumnWidth(0, 20);

Table->setColumnWidth(1, 100);

Table->setColumnWidth(2, 150);

Table->setColumnWidth(3, 100);

Table->setColumnWidth(4, 100);

Table->setColumnWidth(5, 100);

Table->setColumnWidth(6, 100);

}

void Ui::TableSearch() {

QString type = windows["TableWindow"]->findChild<QComboBox \*>("SearchType")->currentText();

QString data = windows["TableWindow"]->findChild<QLineEdit \*>("SearchLine")->text();

QTableWidget \*Table = windows["TableWindow"]->findChild<QTableWidget \*>("Table");

int temp2;

if (type == "Сумма") {

temp2 = 6;

} else if (type == "Сроку") {

temp2 = 5;

} else if (type == "Типу") {

temp2 = 4;

}

for (int i = 0; i < Table->rowCount(); ++i) {

if (Table->item(i, temp2)->text() != data) {

// 4 is the column index for "Type"

Table->setRowHidden(i, true);

} else {

Table->setRowHidden(i, false);

}

}

}

void Ui::ResetDepositWindow() {

QLineEdit \*Name = windows["AddDepositWindow"]->findChild<QLineEdit \*>("NameLine");

QLineEdit \*Surname = windows["AddDepositWindow"]->findChild<QLineEdit \*>("SurnameLine");

QLineEdit \*Phone = windows["AddDepositWindow"]->findChild<QLineEdit \*>("PhoneLine");

QLineEdit \*Email = windows["AddDepositWindow"]->findChild<QLineEdit \*>("EmailLine");

QSlider \*Time = windows["AddDepositWindow"]->findChild<QSlider \*>("TimeSlider");

QLineEdit \*Amount = windows["AddDepositWindow"]->findChild<QLineEdit \*>("AmountLine");

QLabel \*Percent = windows["AddDepositWindow"]->findChild<QLabel \*>("PercentLabel");

QLabel \*Error = windows["AddDepositWindow"]->findChild<QLabel \*>("ErrorLabel");

QLabel \*TimeLabel = windows["AddDepositWindow"]->findChild<QLabel \*>("TimeNumLabel");

QComboBox \*Type = windows["AddDepositWindow"]->findChild<QComboBox \*>("TypeOption");

Name->clear();

Surname->clear();

Phone->clear();

Email->clear();

Type->setCurrentIndex(0);

Time->setValue(1);

Amount->clear();

Percent->setText("5");

Error->clear();

TimeLabel->setText("1");

}

void Ui::AddDeposit() {

QLineEdit \*Name = windows["AddDepositWindow"]->findChild<QLineEdit \*>("NameLine");

QLineEdit \*Surname = windows["AddDepositWindow"]->findChild<QLineEdit \*>("SurnameLine");

QLineEdit \*Phone = windows["AddDepositWindow"]->findChild<QLineEdit \*>("PhoneLine");

QLineEdit \*Email = windows["AddDepositWindow"]->findChild<QLineEdit \*>("EmailLine");

QSlider \*Time = windows["AddDepositWindow"]->findChild<QSlider \*>("TimeSlider");

QLineEdit \*Amount = windows["AddDepositWindow"]->findChild<QLineEdit \*>("AmountLine");

QLabel \*Percent = windows["AddDepositWindow"]->findChild<QLabel \*>("PercentLabel");

QLabel \*Error = windows["AddDepositWindow"]->findChild<QLabel \*>("ErrorLabel");

QComboBox \*Type = windows["AddDepositWindow"]->findChild<QComboBox \*>("TypeOption");

Error->setStyleSheet("QLabel { color : red; }");

string s = DepositFunctions::AddDeposit(Deposits, "", Name->text().toStdString(), Surname->text().toStdString(),

Phone->text().toStdString(), Email->text().toStdString(),

Type->currentText().toStdString(), Amount->text().toInt(),

Time->value(), Percent->text().toInt());

if (s != "1") {

Error->setText(QString::fromStdString(s));

} else {

DepositFunctions::SaveData(Deposits);

ResetDepositWindow();

}

}

void Ui::DrawDiagram(QGraphicsScene \*scene) {

QComboBox \*DiagramType = windows["DiagramWindow"]->findChild<QComboBox \*>("DiagramType");

if (DiagramType->currentText() == "Сумме") {

scene->clear();

int sum = 0;

// Calculate sum of all deposits and Top10 deposits

vector<shared\_ptr<Deposit> > Top10;

for (auto &i: \*Deposits) {

if (Top10.size() < 10) {

Top10.push\_back(i);

} else {

for (int j = 0; j < Top10.size(); j++) {

if (Top10[j]->getAmount() < i->getAmount()) {

Top10[j] = i;

break;

}

}

}

}

std::sort(Top10.begin(), Top10.end(), [](const shared\_ptr<Deposit> &a, const shared\_ptr<Deposit> &b) {

return a->getAmount() > b->getAmount();

});

int startAngle = 0;

for (auto &i: \*Deposits) {

sum += i->getAmount();

}

// Draw top 10 deposits in a circle diagram

QGraphicsEllipseItem \*elips;

QGraphicsTextItem \*text;

double percent;

const QColor colors[10] = {

Qt::red, Qt::green, Qt::blue, Qt::yellow, Qt::cyan, Qt::magenta, Qt::gray, Qt::darkRed, Qt::darkGreen,

Qt::darkBlue

};

for (int i = Top10.size() - 1; i != -1; i--) {

percent = (Top10[i]->getAmount() \* 1.0 \* 340 \* 16) / sum + 32;

elips = scene->addEllipse(0, -155, 300, 300, QPen(Qt::black), QBrush(colors[i]));

elips->setStartAngle(startAngle);

elips->setSpanAngle(percent);

startAngle += percent;

text = scene->addText(QString::fromStdString(Top10[i]->getLogin()));

text->setPos(500, -155 + i \* 20);

text->setDefaultTextColor(colors[i]);

}

} else if (DiagramType->currentText() == "Типу") {

scene->clear();

int t1 = 0, t2 = 0, t3 = 0;

for (auto &i: \*Deposits) {

if (i->getType() == "Обычный") {

t1++;

} else if (i->getType() == "Пенсионный") {

t2++;

} else {

t3++;

}

}

// Draw diagram of deposit types

QGraphicsEllipseItem \*elips;

double percent, startAngle;

elips = scene->addEllipse(0, -155, 300, 300, QPen(Qt::black), QBrush(Qt::red));

percent = (t1 \* 1.0 \* 360 \* 16) / (t1 + t2);

elips->setSpanAngle(percent);

startAngle += percent;

QGraphicsTextItem \*text1 = scene->addText(QString::fromStdString("Тип 1"));

text1->setPos(500, -155);

text1->setDefaultTextColor(Qt::red);

elips = scene->addEllipse(0, -155, 300, 300, QPen(Qt::black), QBrush(Qt::green));

percent = (t2 \* 1.0 \* 360 \* 16) / (t1 + t2);

elips->setStartAngle(startAngle);

elips->setSpanAngle(percent);

startAngle += percent;

QGraphicsTextItem \*text2 = scene->addText(QString::fromStdString("Тип 2"));

text2->setPos(500, -135);

text2->setDefaultTextColor(Qt::green);

}

}

void Ui::MakeReport(QGraphicsScene \*scene, vector<string> Logins) {

}

void Ui::SetupWindows() {

// Load all windows and vars

const std::string windowsNames[] = {"AddDepositWindow", "TableWindow", "DiagramWindow"};

windows["AddDepositWindow"] = loadUiFile(nullptr, "../Ui/Добавление вклада.ui");

windows["TableWindow"] = loadUiFile(nullptr, "../Ui/Таблица.ui");

windows["DiagramWindow"] = loadUiFile(nullptr, "../Ui/Диаграмма.ui");

windows["DeletingConfirmWindow"] = loadUiFile(nullptr, "../Ui/Потверждение удаления.ui");

windows["Report"] = loadUiFile(nullptr, "../Ui/Отчет.ui");

// Create var for graphics

QGraphicsScene \*scene = new QGraphicsScene;

// setup mainwindow

main->takeCentralWidget();

main->setCentralWidget(tabWidget);

DrawTable();

tabWidget->addTab(windows["TableWindow"], "Таблица");

tabWidget->addTab(windows["AddDepositWindow"], "Добавление вклада");

tabWidget->addTab(windows["DiagramWindow"], "Диаграмма");

main->show();

main->setFixedWidth(1250);

main->setFixedHeight(860);

// Functionality of ui elements

// TableWindow buttons and functionality

qDebug() << "TableWindowINIT";

//Change theme

QPushButton \*ChangeThemeButton = windows["TableWindow"]->findChild<QPushButton \*>("ChangeTheme");

QObject::connect(ChangeThemeButton, &QPushButton::clicked,

[this, ChangeThemeButton] { ChangeTheme(ChangeThemeButton); });

//Search

QPushButton \*seatchButton = windows["TableWindow"]->findChild<QPushButton \*>("SearchButton");

QObject::connect(seatchButton, &QPushButton::clicked, [this] { TableSearch(); });

//Sorting

QTableWidget \*Table = windows["TableWindow"]->findChild<QTableWidget \*>("Table");

Table->setSortingEnabled(false);

QObject::connect(Table->horizontalHeader(), &QHeaderView::sectionClicked, [=, this](const int logicalIndex) {

if (logicalIndex >= 4 and logicalIndex <= 6) {

Table->sortByColumn(logicalIndex, Qt::AscendingOrder);

} else if (logicalIndex == 0) {

DrawTable();

}

});

//Edit toggle

Table->setEditTriggers(QAbstractItemView::NoEditTriggers);

QRadioButton \*radioButton = windows["TableWindow"]->findChild<QRadioButton \*>("EditToggle");

QObject::connect(radioButton, &QRadioButton::toggled, [=, this](bool checked) {

if (checked) {

Table->setEditTriggers(QAbstractItemView::DoubleClicked);

} else {

Table->setEditTriggers(QAbstractItemView::NoEditTriggers);

}

});

// AddDepositWindow buttons and functionality

qDebug() << "DepositWindowINIT";

QPushButton \*ResetFieldsButton = windows["AddDepositWindow"]->findChild<QPushButton \*>("ResetFields");

QLabel \*PercentLabel = windows["AddDepositWindow"]->findChild<QLabel \*>("PercentLabel");

QComboBox \*Type = windows["AddDepositWindow"]->findChild<QComboBox \*>("TypeOption");

//Dynamic changing of percent label

QObject::connect(Type, &QComboBox::currentTextChanged, [=, this]() {

if (Type->currentText() == "Обычный") {

PercentLabel->setText("5");

} else if (Type->currentText() == "Пенсионный") {

PercentLabel->setText("10");

}

});

**Вывод:**  в результате работы были получены практические навыки разработки модульных тестов кода, написанного на языке C++ с применением средств Unit Test.