|  |  |
| --- | --- |
|  | |
| МИНОБРНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«Московский технологический университет»**  **МИРЭА** | |
|  | Институт информационных технологий |
|  | Кафедра инструментального и прикладного программного обеспечения |

|  |
| --- |
| **ОТЧЁТ**  **ЛАБОРАТОРНАЯ РАБОТА №4** |
| **дисциплина** |
| **«Разработка программного приложения»**  *(наименование дисциплины)* |
|  |
| Выполнил студент группы \_ИКБО-06-16\_\_\_  *(учебная группа)* | |
| Принял доцент кафедры ИТС, доцент, к.т.н.  *должность, звание, ученая степень* | |

|  |  |  |
| --- | --- | --- |
| Работы выполнены | «\_\_»\_\_\_\_\_\_\_201\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_201\_\_ г. | *(подпись руководителя)* |

Москва 2017

1. **Цель**

Изучение встроенных абстрактных классов и их сигнального взаимодействия с интерфейсом в среде разработки Qt.

1. **Задание**

Взять программу из лабораторной работы №3. Реализовать передачу данных в таблицу с помощью реализации собственной модели. (В Qt это будет класс, порожденный от QAbstractTableModel).

1. **Описание программы**

Относительно третьей лабораторной работы добавлено:  
Класс MyModel, наследуемый от QAbstractItemModel (от которого наследуется QAbstractTableModel). Данный класс реализует все необходимые функции для взаимодействия с таблицей, включая сигнальное соединение модели и таблицы в функциях редактирования таблицы.

Благодаря этому, а класс mainWindow с третьей лабораторной работы изменения практически не вносились, за исключением использования объекта класса MyModel вместо QStandartItemModel, а также ф-ии insertRow, переопределенной в классе MyModel, вместо ф-ии setRowCount из 3-й лабораторной работы.

1. **Код программы**

Заголовочный файл класса MyModel:

#ifndef MYMODEL\_H

#define MYMODEL\_H

#include <QObject>

#include <QAbstractItemModel>

class MyModel : public QAbstractItemModel

{

Q\_OBJECT

public:

int rows;

int columns;

QList<QList<QVariant>> list;

QList<QVariant> up\_header;

QList<QVariant> left\_header;

MyModel();

MyModel(int rows,int columns);

void setRowCount(int row);

virtual QVariant *data*(const QModelIndex &index, int role = Qt::DisplayRole) const;

//virtual QVariant data(const QModelIndex&) const = 0;

virtual int *columnCount*(const QModelIndex &parent) const;

virtual int *rowCount*(const QModelIndex &parent) const;

virtual int *columnCount*() const;

virtual int *rowCount*() const;

void addRow();

virtual bool *insertRows*(int row, int count, const QModelIndex &parent = QModelIndex());

virtual QModelIndex *index*(int row, int column, const QModelIndex &parent = QModelIndex()) const;

//virtual QModelIndex index(int row, int column, const QModelIndex &parent) const;

//virtual QModelIndex index(int row, int column) const;

virtual QModelIndex *parent*(const QModelIndex &child) const;

virtual bool *setData*(const QModelIndex &index, const QVariant &value, int role = Qt::EditRole);

virtual bool *removeRow*(int row, const QModelIndex &parent);

virtual bool *removeRow*(int row);

virtual QVariant *headerData*(int section, Qt::Orientation orientation,

int role = Qt::DisplayRole) const;

virtual bool *setHeaderData*(int section, Qt::Orientation orientation,

const QVariant &value, int role = Qt::EditRole);

};

#endif // MYMODEL\_H

Исполняемый файл класса MyModel:

#include "mymodel.h"

#include <QDebug>

MyModel::MyModel()

{

}

MyModel::MyModel(int rows, int columns)

{

this->rows = rows;

this->columns = columns;

QVariant smth= "";

for (int i=0;i<rows;i++)

{

QList<QVariant> ls;

for (int j=0;j<columns;j++)

{

ls.append(smth);

}

list.append(ls);

}

}

void MyModel::setRowCount(int row)

{

}

QVariant MyModel::data(const QModelIndex &index, int role) const

{

if (role == Qt::DisplayRole)

return list.at(index.row()).at(index.column());

else return QVariant();

}

int MyModel::columnCount(const QModelIndex &parent) const

{

return columns;

}

int MyModel::rowCount(const QModelIndex &parent) const

{

return rows;

}

int MyModel::columnCount() const

{

return columns;

}

int MyModel::rowCount() const

{

//if (list.length()!=rows) qDebug()<<"POLUNDRA\_ROWS!!";

return rows;

}

void MyModel::addRow()

{

beginInsertRows(createIndex(0,0,(void\*)NULL),0,rows);

rows++;

QVariant smth= "";

QList<QVariant> ls;

for (int j=0;j<columns;j++)

{

ls.append(smth);

}

list.append(ls);

endInsertRows();

}

bool MyModel::insertRows(int row, int count, const QModelIndex &parent)

{

beginInsertRows(parent,row,row+count-1);

for(int i=0;i<count;i++)

{

rows++;

QVariant smth= "";

QList<QVariant> ls;

for (int j=0;j<columns;j++)

{

ls.append(smth);

}

list.append(ls);

}

endInsertRows();

}

QModelIndex MyModel::index(int row, int column, const QModelIndex &parent) const

{

//qDebug()<<"INDEX should be: "<<row<<" "<<column;

return createIndex(row,column,(quintptr)NULL);//\*index;

}

QModelIndex MyModel::parent(const QModelIndex &child) const

{

return createIndex((int)-1,(int)-1,(void\*)NULL);

}

bool MyModel::setData(const QModelIndex &index, const QVariant &value, int role)

{

qDebug()<<"INDEX: "<<index.row()<<" "<<index.column();

emit layoutAboutToBeChanged();

try

{

list[index.row()][index.column()] = value;

emit layoutChanged();

return true;

}

catch(...)

{

return false;

}

}

bool MyModel::removeRow(int row, const QModelIndex &parent)

{

try

{

beginRemoveRows(QModelIndex(),row,row);

list.removeAt(row);

rows--;

endRemoveRows();

return true;

}

catch(...)

{

return false;

}

}

bool MyModel::removeRow(int row)

{

try

{

beginRemoveRows(QModelIndex(),row,row);

list.removeAt(row);

rows--;

endRemoveRows();

return true;

}

catch(...)

{

return false;

}

}

QVariant MyModel::headerData(int section, Qt::Orientation orientation, int role) const

{

if (role==Qt::DisplayRole)

{

if (orientation == Qt::Orientation::Horizontal)

{

return up\_header[section];

}

else if (orientation == Qt::Orientation::Vertical)

{

return left\_header[section];

}

}

}

bool MyModel::setHeaderData(int section, Qt::Orientation orientation, const QVariant &value, int role)

{

if (role==Qt::DisplayRole)

{

if (orientation == Qt::Orientation::Horizontal)

{

return !(up\_header[section]=value).isNull();

}

else if (orientation == Qt::Orientation::Vertical)

{

return !(left\_header[section]=value).isNull();

}

}

return false;

}

1. **Скриншоты и примеры работы**

Интерфейс программы представлен на Рис. 1-4 (не отличается от интерфейса лаб. раб №3) :

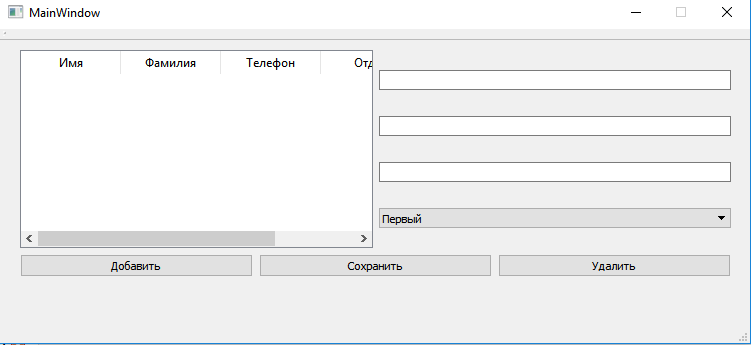


Рис. 1 – Интерфейс при запуске.

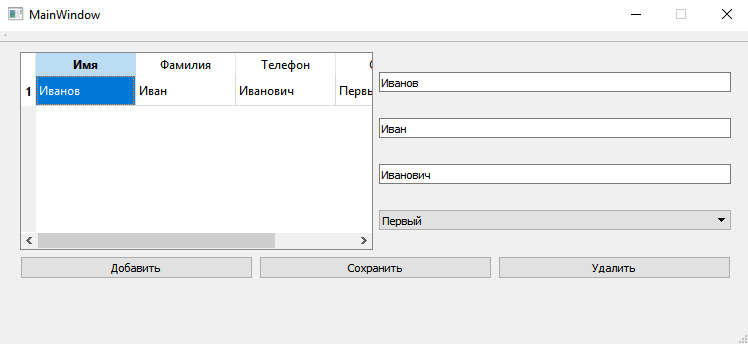


Рис. 2 – Добавлена строка

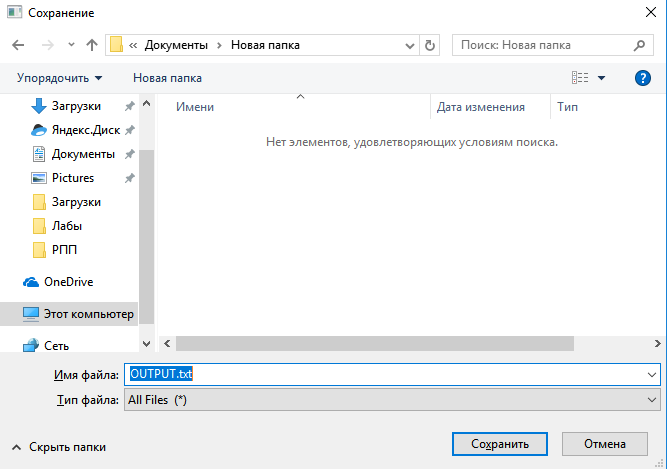


Рис. 3 – Окно сохранения

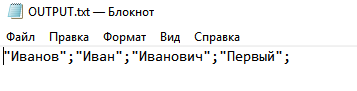


Рис. 4 – Полученное значение в файле вывода

1. **Заключение**

В результате данной лабораторной работы произошло знакомство с абстрактными классами и их функциями, а также способами сигнального взаимодействия интерфейса и объекта класса в среде разработки Qt.

1. **Используемая литература**

*Герберт Шилдт.* Теория и практика С++ = Shildt's Expert C++. — СПб.: BHV — Санкт- Петербург, 1996.

Qt 5.3 Профессиональное программирование на C++. – СПб.:БХВ-Петербург, 2015. - 928 с.

Дейт К. Дж*.* Введение в системы баз данных — 8-е изд. — М.: Вильямс, 2005. — 1328 с.

КузнецовС. Д. Основы баз данных. — 2-е изд. — М.: Интернет-университет информационных технологий; БИНОМ. Лаборатория знаний, 2007. — 484 с