МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»
(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова)

Кафедра О7 Информационные системы и программная инженерия
шифр наименование кафедры, по которой выполняется работа

Дисциплина Визуальное программирование
наименование дисциплины

ПРАКТИЧЕСКАЯ РАБОТА №4

Использование шаблонов проектирования при разработке программ

Язык: C++ (вместе с Qt)

Вариант №10

ОБУЧАЮЩИЙСЯ

группы О726Б

подпись Махов Н.М.
фамилия и инициалы

дата сдачи

ПРОВЕРИЛ

преподаватель каф. О7
ученая степень, ученое звание, должность

подпись Устиновский Г.С.
фамилия и инициалы

Оценка / балльная оценка

дата проверки

г. Санкт-Петербург
2024 г.

СОДЕРЖАНИЕ

1 Цель работы и постановка задачи

1.1 Цель работы

Научиться применять архитектурные шаблоны при проектировании и разработке приложений.

1.2 Постановка задачи

Для решения поставленной задачи был использован C++, совместно с QT.

1.3 Вариативная часть задания

Вариант №10.

Требуется разработать программу для отображения иерархической структуры данных о космических объектах.

В данной работе требуется создать интерактивный справочник

космических объектов. Для примера можно привести следующую иерархию:

Земля → Солнечная система→ Местное межзвёздное облако → Местный пузырь → Пояс Гулда → Рукав Ориона → Млечный Путь → Подгруппа Млечного Пути → Местная группа → Местный лист → Местное сверхскопление галактик → Ланиакея → Комплекс сверхскоплений Рыб-Кита → Объём Хаббла → Метагалактика → Вселенная → Мультивселенная

В левой части приложения должна быть представлена данная иерархия, для отображения которой должен использоваться QTreeView.

По каждому объекту иерархии должна быть возможность получения дополнительной информации - изображения и описания, которые отобразятся в правой части приложения.

Каждый объект иерархии хранится в виде структуры, обязательными полями в которой являются имя, текст и изображение.

Для работы с данными должна быть использована собственная модель, унаследованная от QAbstractItemModel.

Следует реализовать возможность полноценной работы с моделью: добавление, удаление и редактирование объектов, а также изменения иерархии. Также необходимо предусмотреть возможность сохранения

иерархии в бинарный файл, не используя общепринятый язык разметки, и загрузки из него.

2 Реализация

2.1 Содержание файла objecttreemodel.cpp

```
#include "mainwindow.h"
//констурктор по умолчанию создает корень
ObjectTreeModel::ObjectTreeModel (QObject *parent)
    : QAbstractItemModel(parent)
{
    _rootItem = new QObject(this); //теперь можно проверить что элемент является корневым
}
//регистрируем количесво колонок в иерархии
void ObjectTreeModel :: setColumns (QStringList cols){
    _columns = cols;
}
void ObjectTreeModel::addItem(const QString &name, const QString &description, const QString &icon, const QModelIndex &parentIdx) {
    QObject* parentObj = objByIndex(parentIdx);
    int newRow = parentObj->children().count();  // Количество до вставки

    beginInsertRows(parentIdx, newRow, newRow);  // Указываем место для нового элемента

    QObject* newItem = new QObject(parentObj);    // Создаем объект
    _itemData[newItem] = {name, description, icon};    // Заполняем данные с иконкой
```

```
    endInsertRows();  // Завершаем вставку
}

//метод по преобразованию индекс в указатель QObject
QObject    *ObjectTreeModel::objByIndex(const    QModelIndex
&index) const
{
    //если индекс корень то вернуть корень
    if (!index.isValid()) return _rootItem;
    //иначе   вернуть   указатель   по   текущему   индексу
преобразованного в указатель QOblect
    return static_cast<QObject*>(index.internalPointer());
}
//метод для определения индекса ячейки по переданным строке
столбце и индексу родителя
QModelIndex ObjectTreeModel::index(int row, int column, const
QModelIndex &parent) const
{
    //проверка   существует   ли   такой   индекс   вообще   иначе
возвращает пустой индекс
    if(!hasIndex(row, column, parent))
        return QModelIndex();
    QObject * parentObj  = objByIndex(parent);

    return      createIndex(row,      column,      parentObj-
>children().at(row));
}
//метод по получению индекса родителя элемента по индексу
```

```
QModelIndex ObjectTreeModel::parent(const QModelIndex &child)
const
{
    QObject * childObj = objByIndex(child);
    QObject * parentObj = childObj->parent();
    if (parentObj == _rootItem)
        return QModelIndex();
    QObject* grandParentObj = parentObj->parent();
    int row = grandParentObj->children().indexOf(parentObj);
    return createIndex(row, 0, parentObj);
}
//количество дочерних строк по родительскому индексу
int  ObjectTreeModel::rowCount(const  QModelIndex  &parent)
const
{
    return objByIndex(parent) -> children().count();
}

int ObjectTreeModel::columnCount(const QModelIndex &parent)
const
{
    Q_UNUSED(parent);
    return 1; //
}
//передает пакет данных взятых из ячейки по переданному
индесу, где role это роль данных, это отдельная тема Qt,
//короче буквально определяет характер данных
```

```
QVariant ObjectTreeModel::data(const QModelIndex &index, int
role) const {
    if (!index.isValid())
        return QVariant();
    QObject *obj = objByIndex(index);
    const TreeItemData &itemData = _itemData[obj];

    if (role == Qt::DisplayRole && index.column() == 0) {
        return itemData.name;
    } else if (role == Qt::UserRole) {
        return itemData.description;
    }    else    if    (role    ==    Qt::DecorationRole    &&
index.column()==0) {
        QIcon icon(itemData.icon);
        if (icon.isNull()){
            qWarning()  <<  "Icon  is  null  for  path:  "  <<
itemData.icon;
        }
        return icon;
    }

    return QVariant();
}

void ObjectTreeModel::loadFromFile(const QString &filePath) {
    QFile file(filePath);
    if (!file.open(QIODevice::ReadOnly)) {
        qWarning() << "Не удалось открыть файл:" << filePath;
        return;
```

```
    }

    QByteArray data = file.readAll();
    file.close();
    QJsonDocument doc = QJsonDocument::fromJson(data);
    if (!doc.isObject()) {
        qWarning() << "Неверный формат JSON в файле:" <<
filePath;
        return;
    }
    beginResetModel();  // сбрасываем модель перед загрузкой
новых данных
    _itemData.clear();  // очищаем данные

    QObject* parentObj = _rootItem;  // начинаем с корня
    loadItems(doc.object(), parentObj);
    endResetModel();  // восстанавливаем модель
}
void ObjectTreeModel::loadItems(const QJsonObject &jsonObj,
QObject *parentObj) {
  // Создаем новый элемент и добавляем его в родителя
  QString name = jsonObj["name"].toString();
  QString description = jsonObj["description"].toString();
  QString icon = jsonObj["icon"].toString(); // Получаем путь
к иконке
  beginInsertRows(createIndex(parentObj->children().count(),
0,  parentObj),  parentObj->children().count(),  parentObj-
>children().count());
  QObject *newItem = new QObject(parentObj);
```

```
  _itemData[newItem] = {name, description, icon};
  endInsertRows();
  if             (jsonObj.contains("children")             &&
jsonObj["children"].isArray()) {
      QJsonArray children = jsonObj["children"].toArray();
      for (const QJsonValue &childVal : children) {
          loadItems(childVal.toObject(), newItem);
      }
  }
}

void ObjectTreeModel::saveToFile(const QString &filePath) {
    QFile file(filePath);
    if (!file.open(QIODevice::WriteOnly)) {
        qWarning() << "Не удалось открыть файл:" << filePath;
        return;
    }

    QJsonObject rootObj;

    // Получаем первого ребенка корня, если он существует
    QObject *firstChild = _rootItem->children().isEmpty() ?
nullptr : _rootItem->children().at(0);
    if (firstChild) {
        saveItems(rootObj, firstChild); // Передаем первого
ребенка
    }

    QJsonDocument doc(rootObj);
```

```
    file.write(doc.toJson());
    file.close();
}


void ObjectTreeModel::saveItems(QJsonObject &jsonObj, QObject
*parentObj) { //добавить условие что если родитель не корень
то поля 152 153 154 не выполняется
    const TreeItemData &itemData = _itemData[parentObj];
    //if (_itemData.contains(parentObj)) {
    if (parentObj != _rootItem) { //не добавляется description
    jsonObj["name"] = itemData.name; // Имя корневого объекта
(пустое, если это корень)
    jsonObj["description"] = itemData.description;
    jsonObj["icon"] = itemData.icon;
    }

    QJsonArray childrenArray;
    for (QObject * child : parentObj->children()){
        QJsonObject childObj;
        //if (child!=_rootItem)
        saveItems(childObj, child);
        childrenArray.append(childObj);
    }

    if    (!childrenArray.isEmpty()){    //&&    parentObj-
>parent()==_rootItem) {
        jsonObj["children"] = childrenArray; // это хуйня
перезаписывается
```

```
    }
}
void    ObjectTreeModel::updateItemIcon(const    QModelIndex
&index, const QString &newIconPath) {
    if (!index.isValid())
        return;

    QObject *obj = objByIndex(index);
    if (_itemData.contains(obj)) {
        _itemData[obj].icon = newIconPath; // Обновляем путь
к иконке
        emit dataChanged(index, index, {Qt::DecorationRole});
// Уведомляем о изменении данных
    }
}
void ObjectTreeModel::updateItemDescription(const QModelIndex
&index, const QString &newDescription) {
    if (!index.isValid())
        return;

    QObject *obj = objByIndex(index);
    if (_itemData.contains(obj)) {
        _itemData[obj].description   =   newDescription;   //
Обновляем описание
        emit  dataChanged(index,  index,  {Qt::UserRole});  //
Уведомляем о изменении данных
    }
}
void ObjectTreeModel::removeItem(const QModelIndex &index) {
```

```
    if (!index.isValid())
        return;

    QObject *itemToRemove = objByIndex(index);
    QObject *parentObj = itemToRemove->parent();
    int row = parentObj->children().indexOf(itemToRemove);

    beginRemoveRows(index.parent(), row, row);
    _itemData.remove(itemToRemove);   // Удаляем элемент из
данных
    delete itemToRemove;  // Удаляем объект
    endRemoveRows();
}
```

2.2 Содержание файла mainwindow.cpp

```
#include "mainwindow.h"
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    // Создаем центральный виджет и основной лейаут
    QWidget *centralWidget = new QWidget(this);
    QHBoxLayout *mainLayout = new QHBoxLayout(centralWidget);
// Основной лейаут
    // Лейауты для левой и правой частей
    QVBoxLayout    *LayoutLeft    =    new    QVBoxLayout();
// Левая половина экрана
```

```
    QVBoxLayout    *LayoutRight    =    new    QVBoxLayout();
// Правая половина экрана
    QVBoxLayout *LayoutExit = new QVBoxLayout();
    QSplitter   *splitter   =   new   QSplitter(Qt::Vertical,
centralWidget);

    // ЛЕВАЯ ЧАСТЬ: Виджет TreeView и ввод данных
    QLabel *labelTreeView = new QLabel("Космос");
    LayoutLeft->addWidget(labelTreeView);

    treeView = new QTreeView;
    LayoutLeft->addWidget(treeView);

    QHBoxLayout *inputLayout = new QHBoxLayout;
    lineEdit = new QLineEdit;
    addButton = new QPushButton("Добавить объект");
    deleteButton = new QPushButton("Удалить объект");
    inputLayout->addWidget(lineEdit);
    inputLayout->addWidget(addButton);
    inputLayout->addWidget(deleteButton);
    LayoutLeft->addLayout(inputLayout);

    // ЛЕВАЯ ЧАСТЬ: Виджеты для выбора иконки
    fileButton = new QPushButton("Выбрать изображение");
    fileLabel = new QLabel("Нет выбранного файла");
    updateIconButton = new QPushButton("Обновить иконку");
```

```
    QHBoxLayout *fileLayout = new QHBoxLayout;
    fileLayout->addWidget(fileButton);
    fileLayout->addWidget(fileLabel);
    fileLayout->addWidget(updateIconButton);

    LayoutLeft->addLayout(inputLayout);

    // Подключаем сигнал для удаления
    connect(deleteButton, &QPushButton::clicked, this,
&MainWindow::onDeleteButtonClicked);
    LayoutLeft->addLayout(fileLayout); // Добавляем выбор
иконки в левую часть интерфейса

    // Добавляем левую часть в основной лейаут
    mainLayout->addLayout(LayoutLeft);

    // ПРАВАЯ ЧАСТЬ: Описание и кнопки управления
    labelForDescription = new QLabel("Описание");
    LayoutRight->addWidget(labelForDescription);

    descriptionView = new QTextEdit();
    LayoutRight->addWidget(descriptionView);

    saveChangesButton = new QPushButton("Сохранить
изменения");
    LayoutRight->addWidget(saveChangesButton); // Добавляем
кнопку в правую часть

    // Добавляем правую часть в основной лейаут
```

```
    mainLayout->addLayout(LayoutRight);

    // Кнопка "Выйти" в основной лейаут (располагается
отдельно)
    exitButton = new QPushButton("Выйти");
    LayoutExit->addWidget(splitter);
    LayoutExit->addWidget(exitButton);
    mainLayout->addLayout(LayoutExit);

    // Устанавливаем центральный виджет
    setCentralWidget(centralWidget);

    // Подключаем сигналы и слоты
    connect(addButton,      &QPushButton::clicked,      this,
&MainWindow::onAddButtonClicked);
    connect(updateIconButton,  &QPushButton::clicked,  this,
&MainWindow::onUpdateIconButtonClicked);
    connect(fileButton, &QPushButton::clicked, this, [=]() {
        QString fileName = QFileDialog::getOpenFileName(this,
"Выберите файл", "/Users", "Images (*.png *.xpm *.jpg)");
        if (!fileName.isEmpty()) {
            fileLabel->setText("Выбранный    файл:    "    +
fileName);
            _currentIconPath = fileName;
        }
    });
    connect(saveChangesButton,  &QPushButton::clicked,  this,
&MainWindow::onSaveChangesButtonClicked);
```

```
    connect(exitButton,     &QPushButton::clicked,     this,
&MainWindow::onExitButtonClicked);

    // Модель и дерево объектов
    _model = new ObjectTreeModel(this);
    QStringList cols;
    cols << "Name" << "Description";
    _model->setColumns(cols);

    _model-
>loadFromFile("/Users/nikitamakhov/Documents/qtPr4/pr4Ready/
base.json");
    treeView->setModel(_model);

    // Обработка выбора элемента в TreeView
    connect(treeView->selectionModel(),
&QItemSelectionModel::currentChanged,                    this,
&MainWindow::onTreeItemSelected);
}


MainWindow::~MainWindow()
{}

void MainWindow::onAddButtonClicked()
{
    if   (lineEdit->text().isEmpty()   ||   descriptionView-
>toPlainText().isEmpty()) return;
```

```
    QString iconPath = _currentIconPath.isEmpty()
                           ?
"/Users/nikitamakhov/Documents/qtPr4/pr4Ready/icons/unknown
planet.png"
                           : _currentIconPath;

    // Добавление элемента с выбранной иконкой
    _model->addItem(lineEdit->text(),       descriptionView-
>toPlainText(), iconPath, treeView->currentIndex());

    lineEdit->clear();
    descriptionView->clear();
    _currentIconPath.clear();
}

void MainWindow::onExitButtonClicked() {
    _model-
>saveToFile("/Users/nikitamakhov/Documents/qtPr4/pr4Ready/ba
se.json");
    close();
}
void    MainWindow::onTreeItemSelected(const    QModelIndex
&current, const QModelIndex &previous)
{
    Q_UNUSED(previous);

    if (!current.isValid())
        return;
```

```
    // Получаем описание выбранного элемента и устанавливаем
его в descriptionView
    QString description = _model->data(current,
Qt::UserRole).toString();
    descriptionView->setText(description);
}
void MainWindow::onUpdateIconButtonClicked() {
    QModelIndex currentIndex = treeView->currentIndex();
    if (!currentIndex.isValid()) {
        QMessageBox::warning(this, "Ошибка", "Выберите
элемент для обновления иконки.");
        return;
    }

    if (_currentIconPath.isEmpty()) {
        QMessageBox::warning(this, "Ошибка", "Сначала
выберите изображение.");
        return;
    }
    _model->updateItemIcon(currentIndex, _currentIconPath);
}
void MainWindow::onSaveChangesButtonClicked() {
    QModelIndex currentIndex = treeView->currentIndex();
    if (!currentIndex.isValid()) {
        QMessageBox::warning(this, "Ошибка", "Выберите
элемент для сохранения изменений.");
        return;
    }
```

```
    QString newDescription = descriptionView->toPlainText();
    _model->updateItemDescription(currentIndex,
newDescription);
}


void MainWindow::onDeleteButtonClicked() {
    QModelIndex currentIndex = treeView->currentIndex();
    if (!currentIndex.isValid()) {
        QMessageBox::warning(this,    "Ошибка",    "Выберите
элемент для удаления.");
        return;
    }

    // Удаление элемента из модели
    _model->removeItem(currentIndex);

    // Сохранение изменений
    _model-
>saveToFile("/Users/nikitamakhov/Documents/qt/ShablonPodPr4/
base.json");
}
```

Выравнивание – по ширине.

Абзацный отступ – отсутствует.

2.3 Содержание файл mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
```

```
#include "objecttreemodel.h"
#include <QtWidgets>
#include <QAbstractItemModel>
#include <QStandardItemModel>
#include <QMainWindow>
#include <QIcon>
#include <QJsonDocument>
#include <QJsonObject>
#include <QJsonArray>
#include <QFile>
#include <QSplitter>
#include <QDebug>

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    ObjectTreeModel* _model;
    QTreeView *treeView;
    QLineEdit *lineEdit;
    QTextEdit *descriptionView;
    QLabel *labelForDescription;
    QPushButton *addButton;
    QPushButton *exitButton;
```

```
    QPushButton *fileButton;
    QPushButton *deleteButton;
    QPushButton *updateIconButton;
    QPushButton *saveChangesButton;
    QLabel *fileLabel;
    QString _currentIconPath;
private slots:
    void onAddButtonClicked();
    void onTreeItemSelected(const QModelIndex &current, const
QModelIndex &previous);
    void onExitButtonClicked();
    void onUpdateIconButtonClicked();
    void onSaveChangesButtonClicked();
    void onDeleteButtonClicked();
};
#endif // MAINWINDOW_H
```

2.4 Содержание файла objecttreemodel.h

```
#ifndef OBJECTTREEMODEL_H
#define OBJECTTREEMODEL_H
#include <QAbstractItemModel>
#include <QObject>
#include <QIcon>

struct TreeItemData {
    QString name;
    QString description;
    QString icon;
};
```

```
class ObjectTreeModel : public QAbstractItemModel
{
    Q_OBJECT
public:
    ObjectTreeModel(QObject* parent = nullptr);
    void setColumns(QStringList cols);
    void addItem(const QString &name, const QString &description, const QString &icon, const QModelIndex &parentIdx);
    QObject *_rootItem;  // Явно создаем корень, поскольку объявлен внутри класса он не валиден для присваивания из вне


protected:
    QStringList _columns;
    QObject * objByIndex(const QModelIndex & index) const; // Это для хранения указателей на элементы дерева
    void saveItems(QJsonObject &jsonObj, QObject *parentObj); // Вспомогательный метод для рекурсивного сохранения
// QAbstractItemModel interface
public:
virtual QModelIndex index(int row, int column, const QModelIndex &parent) const override;
virtual QModelIndex parent(const QModelIndex &child) const override;
virtual int rowCount(const QModelIndex &parent) const override;
```

```
virtual int columnCount(const QModelIndex &parent) const override;
virtual QVariant data(const QModelIndex &index, int role) const override;
void loadItems(const QJsonObject &jsonObj, QObject *parentObj);
void loadFromFile(const QString &filePath);
void saveToFile(const QString &filePath);
void updateItemIcon(const QModelIndex &index, const QString &newIconPath);
void updateItemDescription(const QModelIndex &index, const QString &newDescription);
void removeItem(const QModelIndex &index);

private:
    QHash<QObject*, TreeItemData> _itemData;

};
#endif
```

3 Демонстрация работы программы

Запуск приложения, загружается иерархия, модель солнечной системы. Реализованы методы: добавления, удаления объектов, загрузки и обновление изображения, добавление, удаление и редактирование текста описания. Результат представлен на рисунке 1.

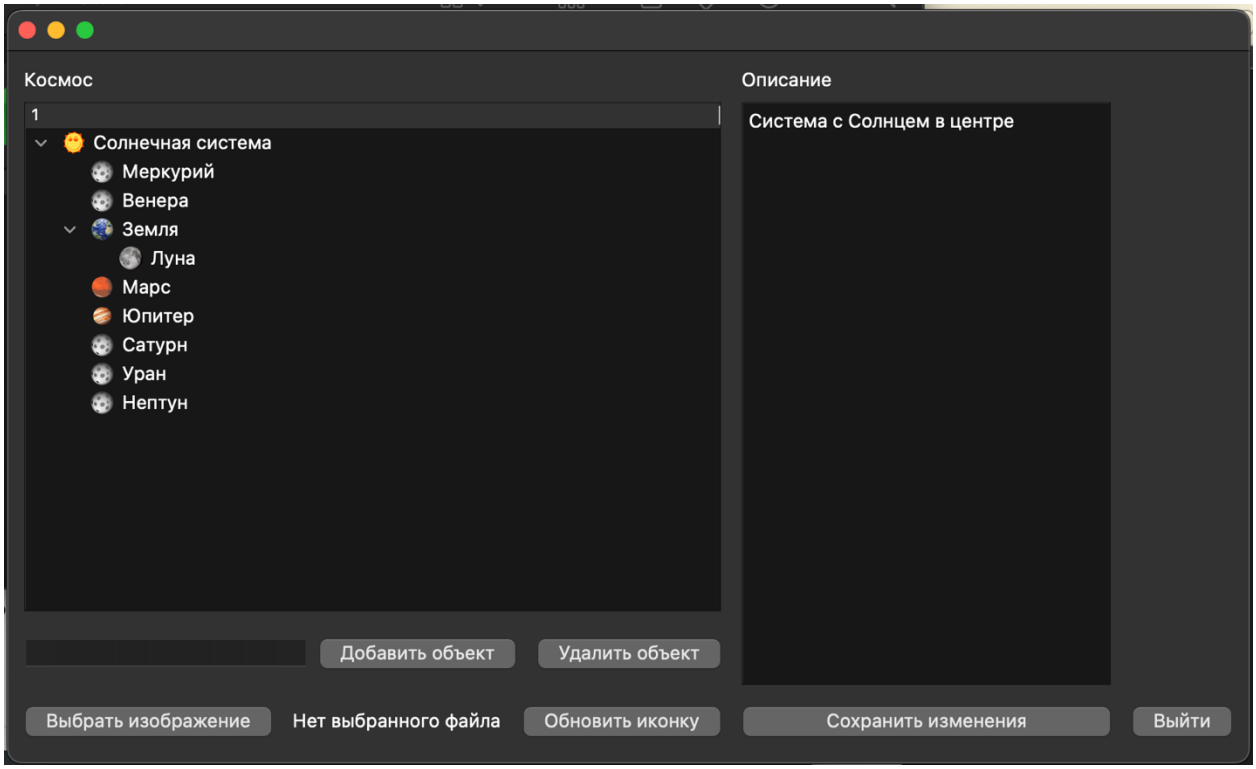

Рисунок 1 — Главное окно программы