**3 СОЗДАНИЕ ПРОГРАММНОЙ СИСТЕМЫ С QT ИНТЕРФЕЙСОМ НА C++**

Задание:

Выполнить объектную декомпозицию, разработать формы интерфейса, диаграмму состояний интерфейса, диаграммы классов интерфейсной и предметной областей, диаграмму последовательности действий одной из реализуемых операций. Разработать, протестировать и отладить программу.

В базе данных (файле) «Мир животных» для каждого животного указан класс, отряд, семейство, вид. Например: класс Млекопитающие, отряд Хищные, семейство Кошачьи, снежный барс. Программа должна в интерактивном режиме формировать файл, добавлять и удалять данные, а также воспринимать каждый из перечисленных запросов и давать на него ответ.

1. Получить список животных данного класса с указанием отряда и семейства.

2. Выяснить, в каком из двух заданных семейств видов больше, и вывести список животных этого семейства.

3. Выяснить, к какому классу, отряду и семейству принадлежит данное животное.

4. Построить гистограмму, отображающую количество отрядов в каждом классе.

1.1 Объектная декомпозиция приложения

При проектировании программного продукта была выделена структура **animal**, включающая в себя имя класса, отряда, семейства и вида животного, а также класс **Animals** для хранения массива из объектов **animal**.

Диаграмма объектов представлена на рисунке 1.

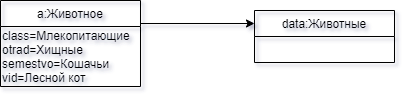


Рисунок 1 – Диаграмма объектов приложения

1.2 Разработка форм интерфейса

Разрабатываемые формы интерфейса должны обеспечивать пользователю возможность выполнения заданных функций.

Форма **mainwindow** предназначена для переключения между выполняемыми задачами, соответственно она должна включать кнопки, приводящие к переходу к соответствующим формам. (рисунок 2).

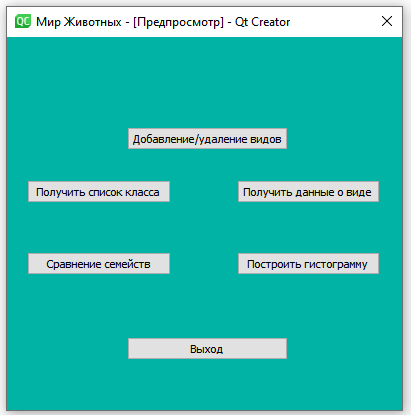


Рисунок 2 – Внешний вид формы «mainwindow»

Форма **comprassion** предназначена для сравнения семейств животных, соответственно она должна включать поля, для ввода сравниваемых семейств и поле для вывода результата (рисунок 3).

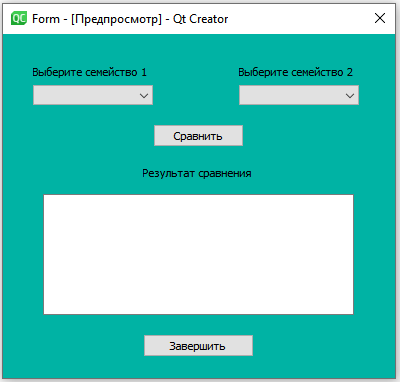


Рисунок 3 – Внешний вид формы «comprassion»

Форма **edit** предназначена для добавления и удаления видов животных, соответственно она должна включать поля для ввода информации о добавляемом и удаляемом виде, а также соответствующие кнопки, приводящие к выполнению заданной операции (рисунок 4).

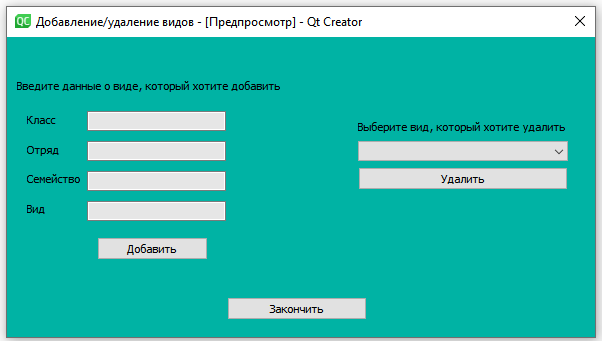


Рисунок 4 – Внешний вид формы «edit»

Форма **getdata** предназначена для того, чтобы получить данные о соответствующем виде, соответственно она должна включать в себя поле ввода вида, о котором пользователь хочет получить данные и соответственные поля для вывода данных (рисунок 5).

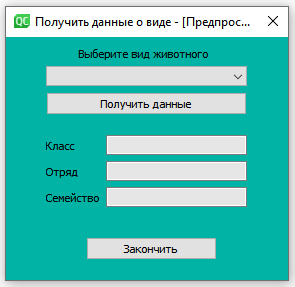


Рисунок 5 – Внешний вид формы «getdata»

Остальные формы были спроектированы аналогично.

1.3 Разработка диаграммы состояний интерфейса

Диаграмма состояний интерфейса показывает возможные варанты переключения форм интерфейса (рисунок 6).

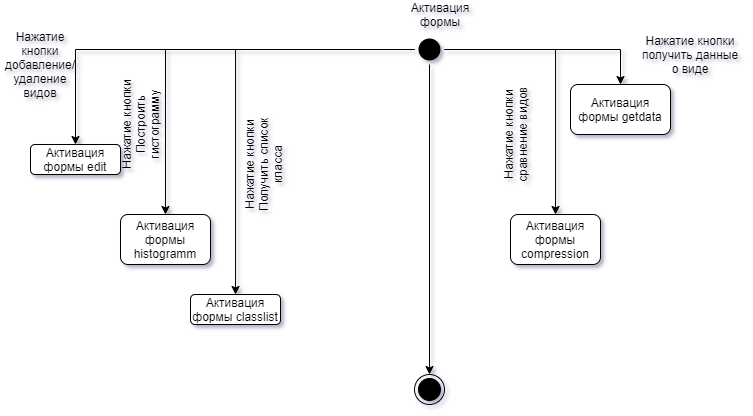
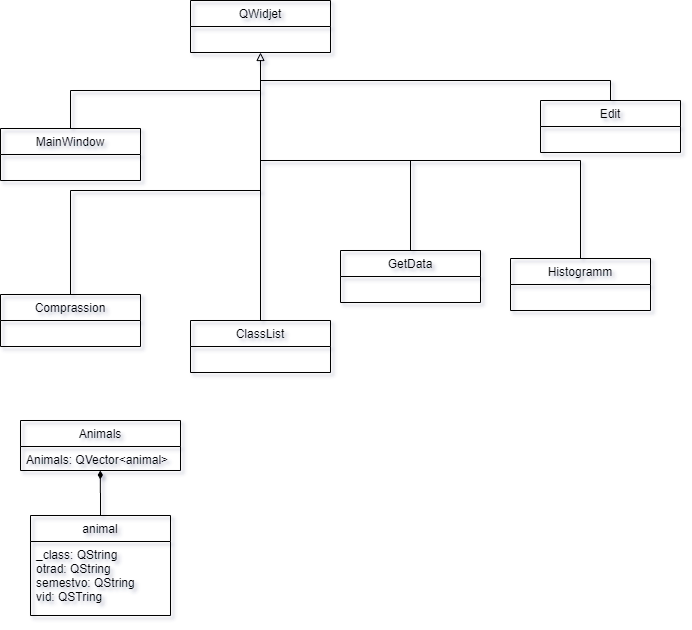


Рисунок 6 – Диаграмма состояний интерфейса

Аналогично разрабатываются диаграммы состояний интерфейса для отдельных форм.

1.4 Разработка диаграммы классов интерфейсной и предметной областей приложения

Диаграмма классов интерфейсной и предметной областей показывает связи между классами (рисунок 7).

Рисунок 7 – Диаграмма классов приложения

1.5 Разработка диаграммы последовательности действий выполнения операции **Удаление вида**

Диаграмма последовательностей действий позволяет уточнить порядок выполнения фрагментов операции различными объектами (рисунок 8).

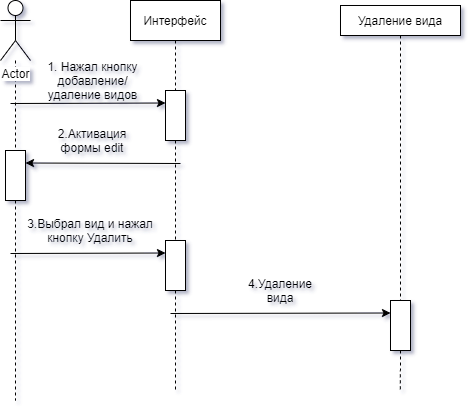


Рисунок 8 – Диаграмма последовательностей действий выполнения операции Удаление вида

1.6 Разработка кода приложения

Уточнение поведения объектов во время выполнения программы позволило разработать код программы.

Ниже в качестве примера приведен код **getdata:**

#include "getdata.h"

#include "ui\_getdata.h"

static Animals adata;

GetData::**GetData**(QWidget \*parent) :

QWidget(parent),

ui(new Ui::GetData)

{

ui->setupUi(this);

setWindowIcon(QIcon(":/img/cat.png"));

}

GetData::~***GetData***()

{

delete ui;

}

void GetData::**on\_pushButton\_2\_clicked**()

{

ui->comboBox->clear();

ui->lineEdit->clear();

ui->lineEdit\_2->clear();

ui->lineEdit\_3->clear();

this->close();

emit mainwindow();

}

void GetData::**recieveData**(Animals a){

ui->comboBox->clear();

if(a.size() != 0){

a.sortVid();

adata = a;

for(int i = 0; i < adata.size(); i++){

ui->comboBox->addItem(adata[i].vid);

}

}

}

void GetData::**on\_pushButton\_clicked**()

{

ui->lineEdit->clear();

ui->lineEdit\_2->clear();

ui->lineEdit\_3->clear();

animal a = adata[adata.search(ui->comboBox->currentText())];

ui->lineEdit->setText(a.\_class);

ui->lineEdit\_2->setText(a.otrad);

ui->lineEdit\_3->setText(a.semestvo);

}

Аналогично был разработан остальной код приложения.

1.7 Тестирование приложения

Разработанное приложение позволяет выполнять функции, предусмотренные заданием (рисунки 9-11).

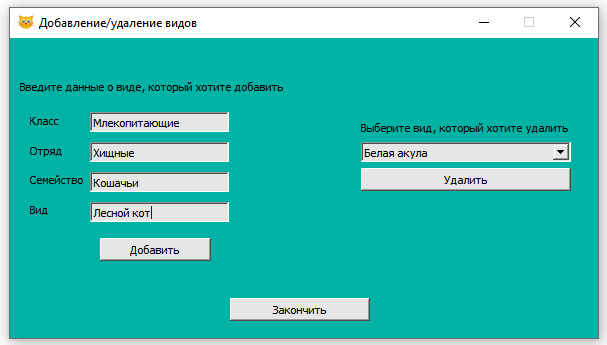


Рисунок 9. Добавление/удаление видов.

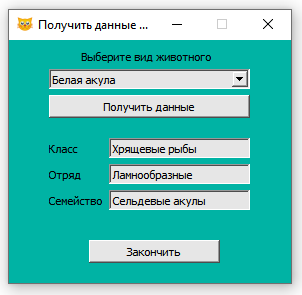


Рисунок 10. Получить данные о виде.

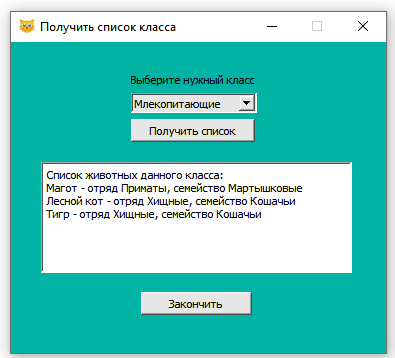


Рисунок 11. Получить список класса.

Вывод

При разработке приложения изучены средства разработки приложений с графическим интерфейсом на языке С++, используемые при объектном подходе, а также основные диаграммы, сопровождающие процесс разработки.