Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки Программная инженерия

РАБОТА ПРОВЕРЕНА

_____(И.О. Ф.)
«____»_____20___г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н.,
доцент
_____/А.А.Замышляева
«____»_____2018 г.

Разработка программы построения интерфейса классов по UML-диаграмме для редактора Umlet

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–09.03.04.2018.74.ПЗ ВКР

Руководитель работы, доцент
_____/А.К. Демидов
«____»_____2018 г.
Автор работы
Студент группы ЕТ-414
_____/ А.В.Байрамгулова
«____»_____2018 г.
Нормоконтролер, доцент
_____/Т.Ю.Оленчикова
«____»_____2018 г.

Челябинск 2018

АННОТАЦИЯ

Байрамгулова А. В. Разработка программы построения интерфейса классов по UML-диаграмме для редактора Umlet. – Челябинск: ЮУрГУ, ЕТ-414, 72 с., 56 ил., 10 табл., библиогр. список – 21 наим., 2 прил.

Данная работа посвящена разработке программы построения интерфейса классов по UML-диаграмме для редактора Umlet.

В первом разделе выполнен анализ потребностей пользователя, проведен обзор существующих решений.

Во втором разделе разработаны требования к программному генератору кода. Рассмотрены все случаи для программной поддержки кода на языке C++.

В третьем разделе приводим описание алгоритмов для работы приложения.

В четвертом разделе приведены результаты тестов кодогенератора.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящее время информационные технологии развиваются в быстром темпе. Информационные системы стали настолько сложными, что способностей человека не достаточно, чтобы представить архитектуру проектируемого программного обеспечения. Обычные подходы к разработке ПО не дают точной гарантии, что результат будет соответствовать заданным требованиям, что правильно совершается переход от одного этапа разработки к другому. Таким образом, чтобы ускорить и упростить разработку программного обеспечения, применяется визуальное моделирование.

Одним из способов проектирования программного обеспечения является применение алгоритмов на основе UML диаграмм. С их помощью можно смоделировать систему с различных точек зрения. Стандарт UML включает 13 видов диаграмм. Для моделирования объектно-ориентированных систем часто используются диаграммы классов. Они наглядно показывают классы и отношения между ними.

На сегодняшний день имеется множество инструментов для работы с UML диаграммами, одним из них является редактор Umlet.

Umlet – это инструмент с открытым исходным кодом для разработки моделей различных процессов и проектирования. Данная программа имеет простой пользовательский интерфейс. С ее помощью можно быстро построить UML диаграммы, экспортировать их в различные графические форматы.

Объекты диаграммы можно модифицировать и использовать в качестве шаблонов, что позволяет пользователям подстраивать приложения в соответствии с их потребностями. Также возможно добавлять собственные элементы для изображения объектов, которые не входят в стандарт UML.

Разработчик программы вынужден разбираться в моделях, которые написал проектировщик, и только потом «вручную» переписывать текст на языке UML в текст на языке программирования, а затем компилировать программу. На этом этапе происходит двойная работа и теряется время [1]. Темой квалификационной работы является разработка генератора программного кода из моделей UML.

Для достижения поставленной цели необходимо решить следующие задачи:

- рассмотреть существующие решения для работы с UML-диаграммами;
- разработать требования к генератору;
- разработать алгоритм работы приложения;
- реализовать программу и выполнить проверку.

1 ПОСТАНОВКА ЗАДАЧИ

1.1 Анализ потребностей пользователя

Данная задача представляет большой интерес для широкого круга разработчиков. Для любого программного проекта приходится разрабатывать большое количество взаимосвязанных моделей. И на этом этапе очень часто возникают проблемы. Для программиста реализация проекта заключается в написании кода. Такой подход не является правильным и может привести к следующему:

- может возникнуть недопонимание между заказчиком и разработчиком, а также между самими разработчиками;
- затруднительно представить, как взаимодействует система с ее компонентами.

Использование UML позволяет решить эти проблемы. Генерация кода из диаграмм существенно упрощает написание программ, так как гарантируется, что код программы будет соответствовать диаграмме.

1.2 Анализ существующих программ

На сегодняшний день существуют множество программ по построению диаграмм с возможностью генерации кода, но большинство из них являются платными и имеют свои недостатки.

1.2.1 StarUML

StarUML – средство для создания UMLдиаграмм, с которым можно работать на следующих платформах: Windows, OS X, Linux (см. рисунок 1.1). Программа поддерживает одиннадцать типов диаграмм и нотацию стандарта UML версии 2.0. StarUML предоставляет максимальную степень адаптации среды разработки пользователя, предлагая настройку параметров, которые могут влиять на методологию разработки программного обеспечения, проектную платформу и язык [2].

Создание элементов в StarUML 2 стало гораздо проще, поддерживаются много сокращений в Quick Edit для одновременного создания элементов и отношений (см. рисунок 1.2).

Чтобы соответствовать растущим потребностям пользователей в увеличении функциональных возможностей моделирования, программа имеет простую и мощную архитектуру с поддержкой плагинов. Это дает платформе возможность участвовать в расширении функций программы, разрабатывать и подключать собственный модуль. С помощью официального реестра Extension Manager можно легко находить и устанавливать расширения (см. рисунок 1.3). Некоторые расширения имеют открытый исходный код.

StarUML хранит модели в формате Javascript Object Notation (JSON). Пользователь может сам разработать модель с использованием шаблонов.

Программа позволяет генерировать исходные коды из своих моделей или строить модель из исходного кода для языков программирования Java, C++. Есть возможность поделиться своими диаграммами при помощи HTML-документов. Сгенерировав HTML-документ, можно просматривать его в большинстве веб-браузеров.

Недостаток заключается в том, что при длительном использовании программы пользователю необходимо прибрести лицензию. Ниже приведен пример, где показывается еще один недостаток генератора.

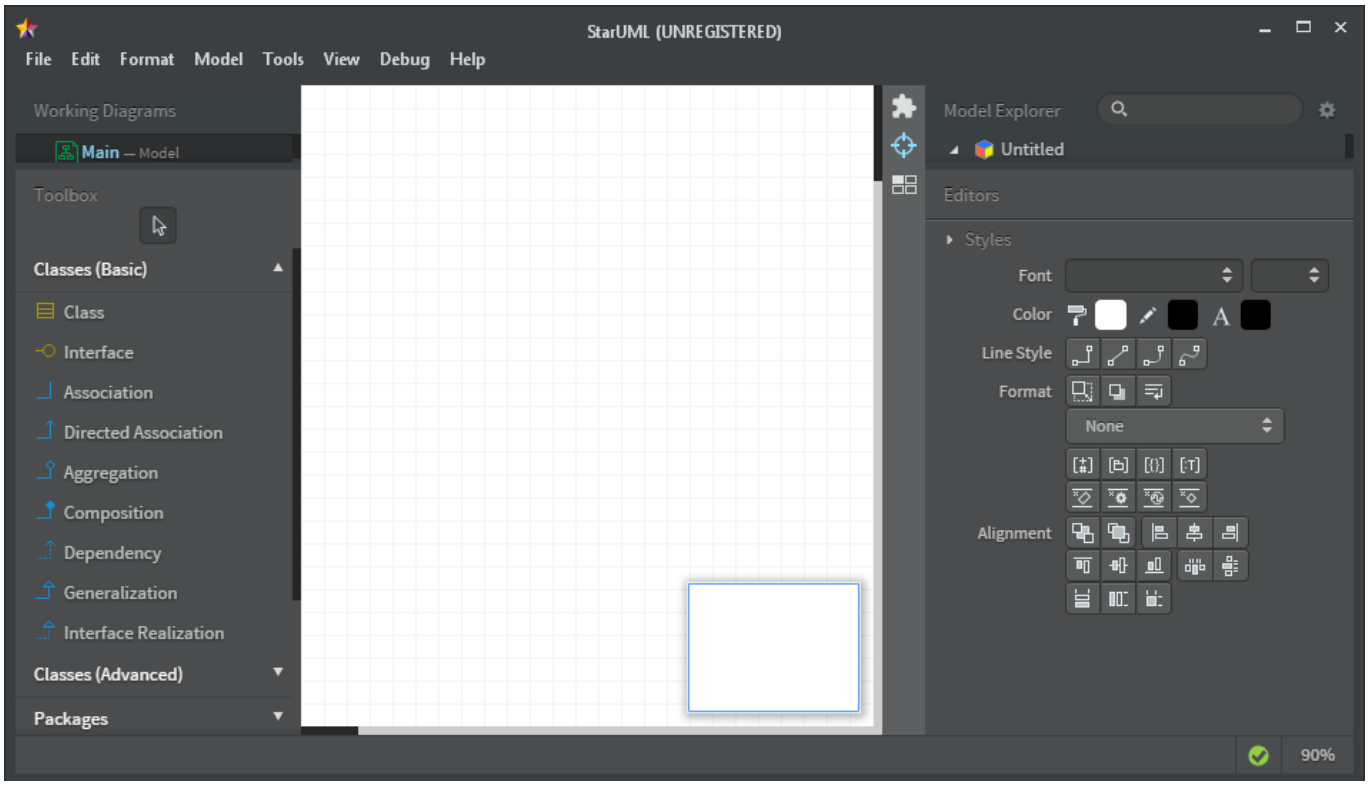


Рисунок 1.1 – Интерфейс программы

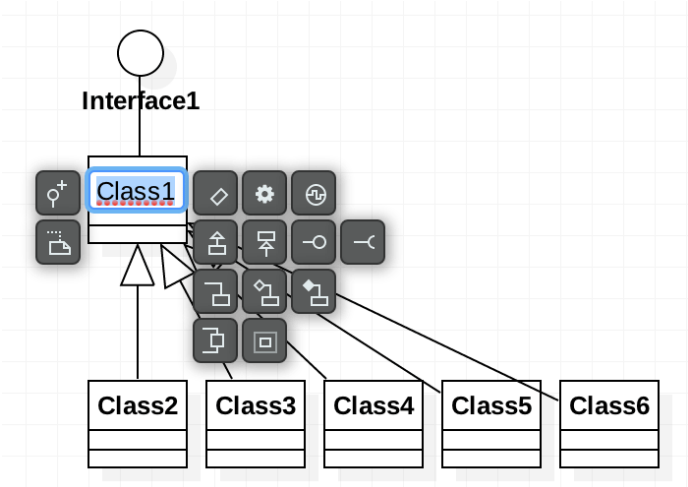


Рисунок 1.2 – Быстрое моделирование

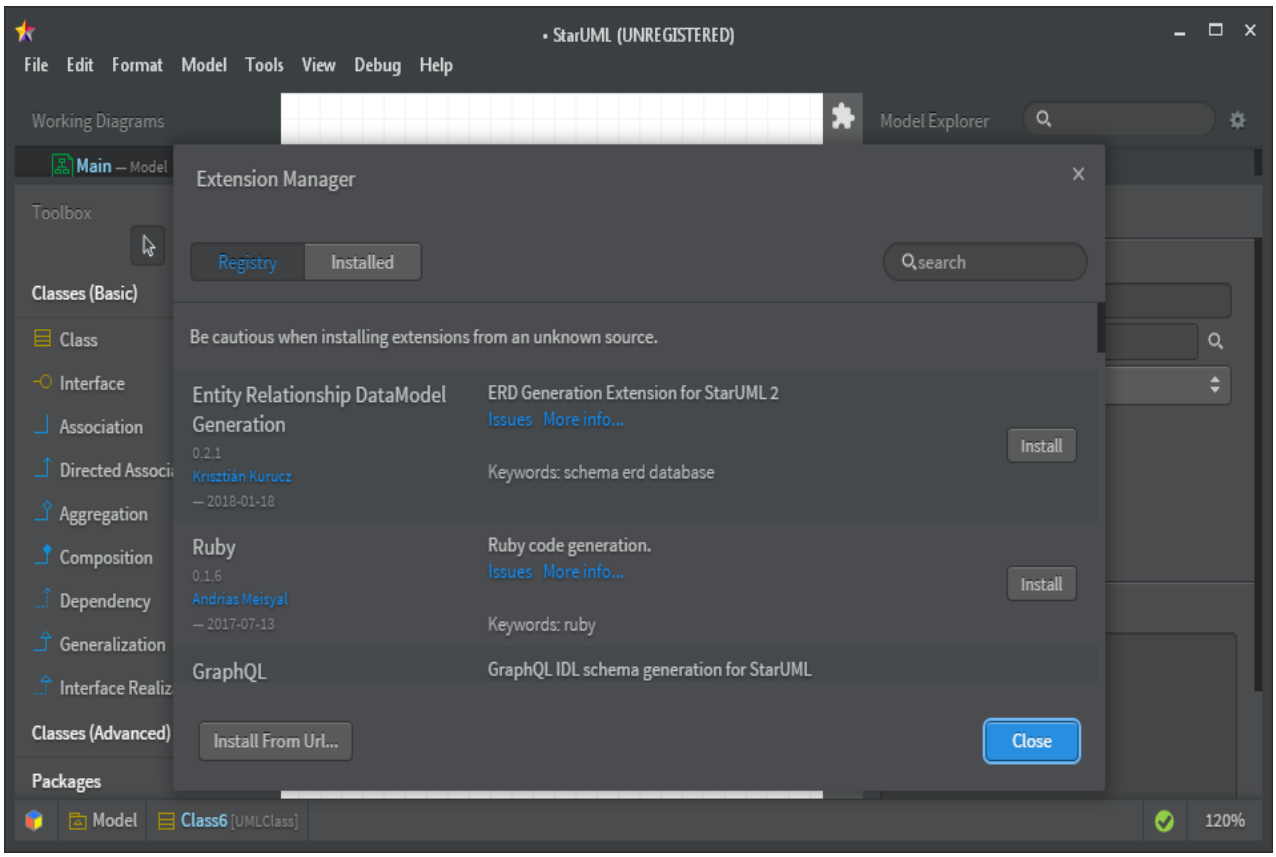


Рисунок 1.3 – Менеджер расширения

Пример работы программы.

В качестве прикладной области был взят класс животных, и построена его модель (см. рисунок 1.4). Для генерации кода по диаграмме, необходимо было в менеджере расширения установить «C++» (см. рисунок 1.5). После работы кодогенератора было получено:

UMLClass – преобразовался в обычный класс C++. Объявление класса записано в файле с расширением h (см. рисунок 1.6). Определение – в .cpp. По умолчанию видимость класса protected. Также поддерживаются шаблоны. Каждый класс реализуется в новом файле (см. рисунок 1.7).

UMLAttribute – поле класса. Видимость также по умолчанию protected. Имя атрибута преобразуется в имя поля. Тип атрибута преобразуется в тип поля. Также инициализируется значение атрибута defaultValue в поле данных.

UMLOperatio – метод. Видимость метода по умолчанию protected. Поддерживается модификатор isAbstract, в таком случае метод будет виртуальным. Поддерживается модификатор isStatic. В случае если нет возвращаемого значения, устанавливается void[2].

В результате программа создала 6 файлов для 3-х диаграмм. Если будет спроектировано больше моделей UML, то количество файлов увеличится. Это может привести к тому, что работать с файлами будет не совсем удобно.

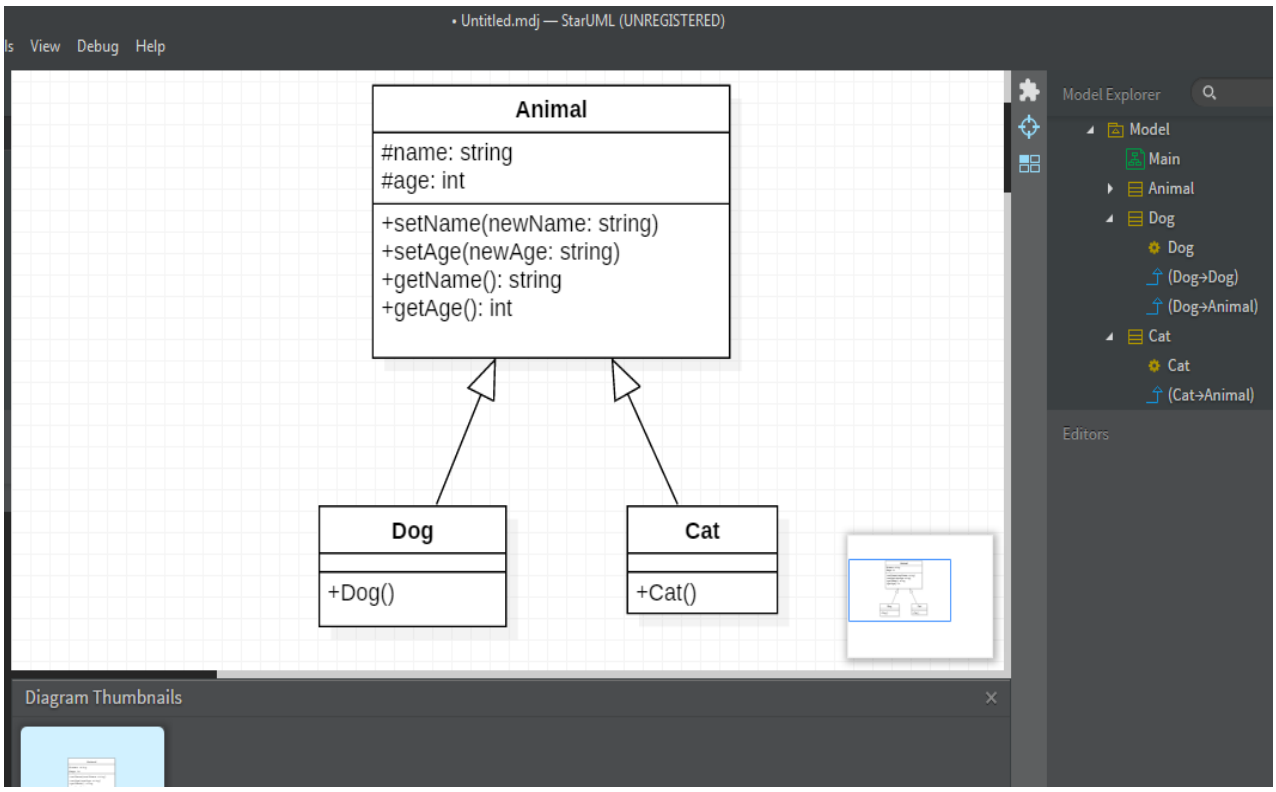


Рисунок 1.4 – Диаграмма классов в StarUML

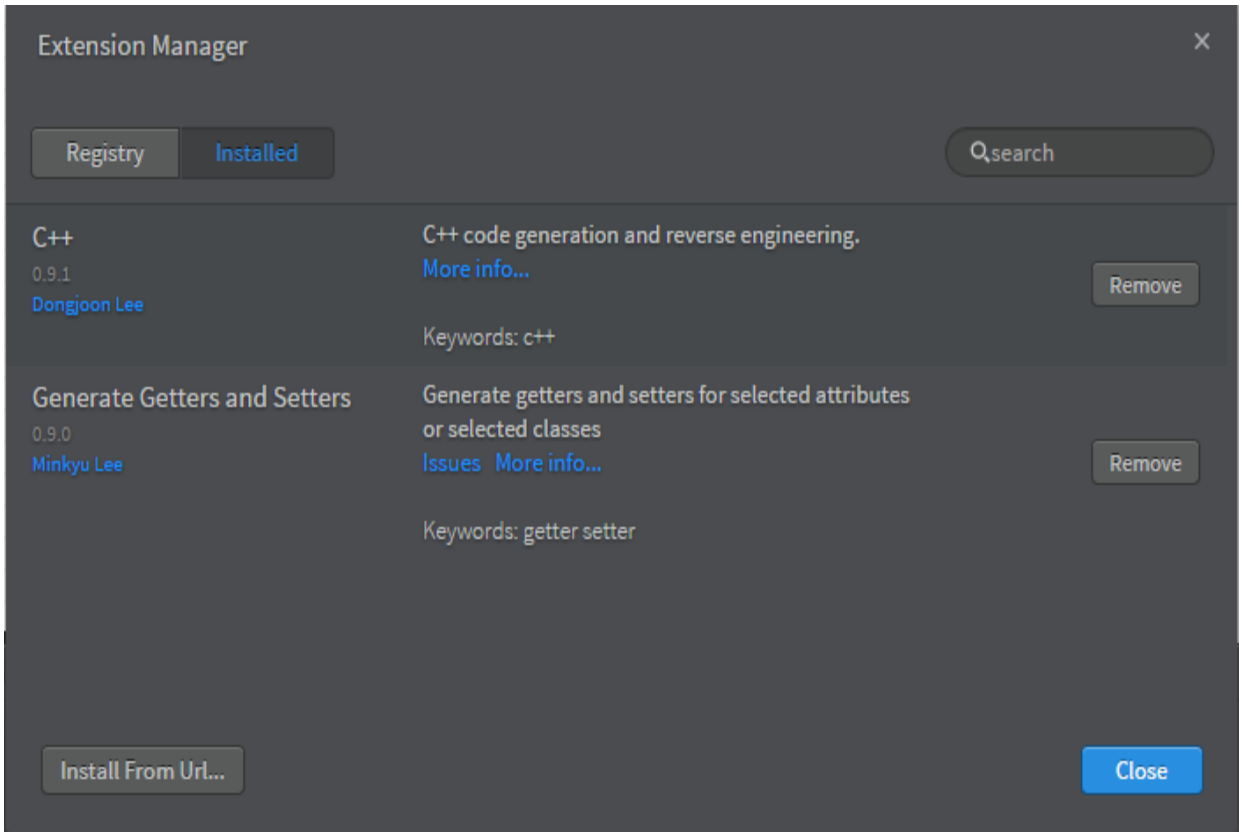


Рисунок 1.5 – Extension Manager с установленным расширение «C++»

Animal.h

```
/**
 * Project Untitled
 */


#ifndef _ANIMAL_H
#define _ANIMAL_H

class Animal: public Animal {
public:

    /**
     * @param newName
     */
    void setName(string newName);

    /**
     * @param newAge
     */
    void setAge(string newAge);

    string getName();

    int getAge();
protected:
    string name;
    int age;
};

#endif //_ANIMAL_H
```

Рисунок 1.6 – Файл Animal.h

Dog.h Cat.h Animal.h

```
/**
 * Project Untitled
 */


#ifndef _DOG_H
#define _DOG_H

#include "Animal.h"


class Dog: public Dog, public Animal {
public:

    void Dog();
};

#endif //_DOG_H
```

Cat.h Animal.h

```
/**
 * Project Untitled
 */


#ifndef _CAT_H
#define _CAT_H

#include "Animal.h"


class Cat: public Animal {
public:

    void Cat();
};

#endif //_CAT_H
```

Рисунок 1.7 – Файлы Dog.h и Cat.h

1.2.2 Visual Studio Ultimate 2013

Еще одно средство по созданию диаграмм – это Visual Studio Ultimate 2013 (см. рисунок 1.8). Это программа позволяет конструировать классы визуальным образом. Для этого поставляется расширение Feature Packs, которое улучшает и дополняет существующие инструменты.

Среда имеет область проектирования Class Designer, в которой можно сформировать диаграмму классов. В них можно добавлять поля и методы, а также устанавливать взаимоотношения между ними. Система Visual Studio позволяет внедрить моделирование в среду разработки в любое время. Диаграмма классов динамически строится по исходному коду. Любые изменения немедленно отражаются на диаграмме, а любое изменение отражается на коде.

С помощью средств Visual Studio можно на основе существующего кода построить UML-модель, но при этом не прилагать гигантские усилия по созданию диаграмм вручную и поддержанию их в актуальном состоянии [3].

Visual Studio Ultimate 2013 поддерживает пять типов диаграмм UML, включая диаграммы классов. Достоинство данной программы заключается в том, что существует возможность генерации кода на основании диаграммы.

Недостатки данной программы:

- занимает достаточно большое пространство памяти;
- нет генерации кода на языке C++;
- является коммерческой версией.

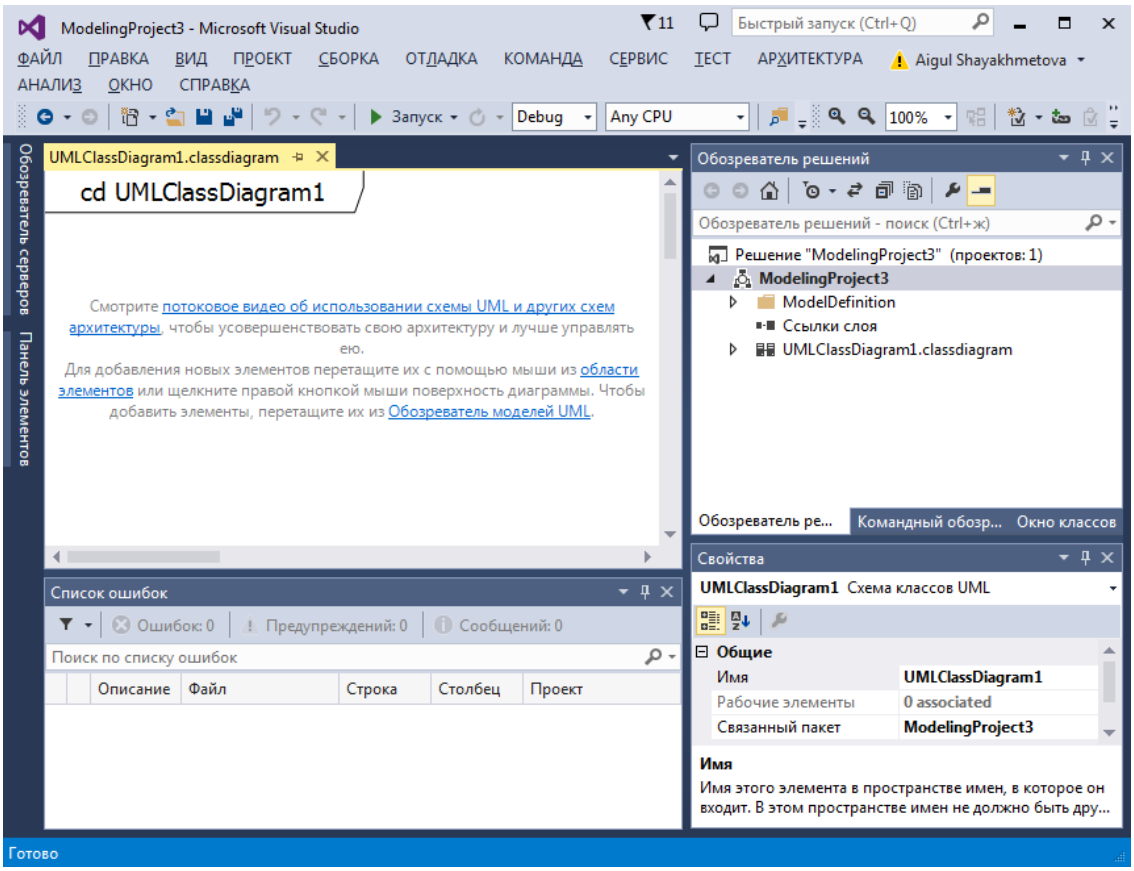


Рисунок 1.8 – Интерфейс Visual Studio 2013

Пример работы программы.

Чтобы создать код на основе схем классов UML в Visual Studio, необходимо воспользоваться командой «создать код».

Предметная область представлена тремя классами: Animal, Dog и Cat (см. рисунок 1.9). После создания диаграммы, создаем код (см. рисунок 1.10). Результат получаем на языке C# (см. рисунок 1.11).

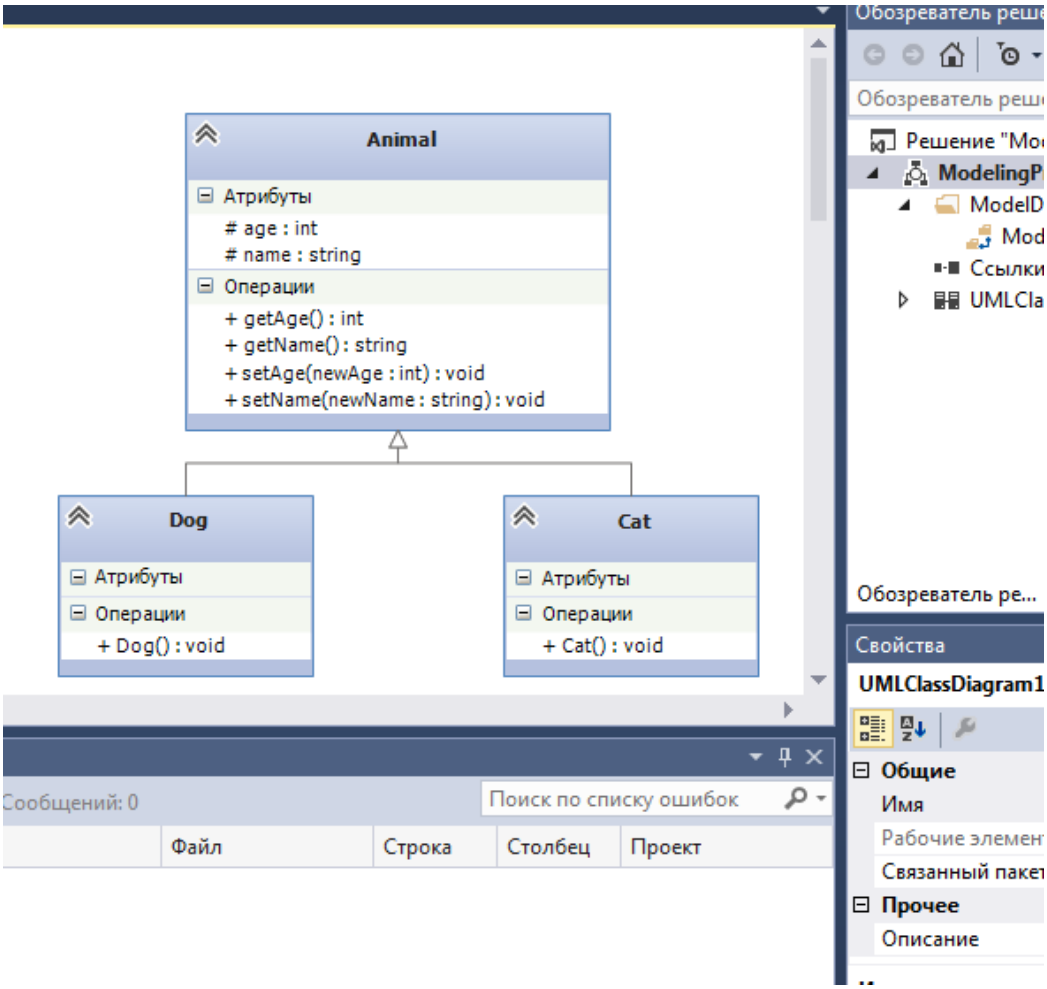


Рисунок 1.9 – Диаграмма класса в Visual Studio

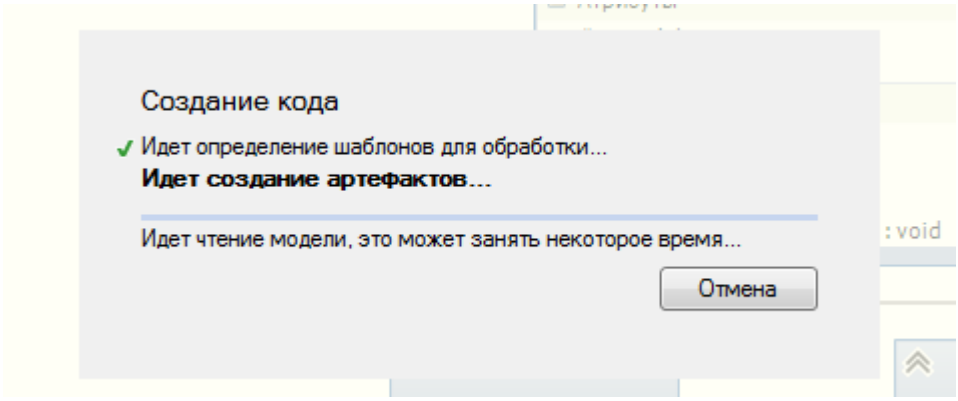


Рисунок 1.10 – Создание кода

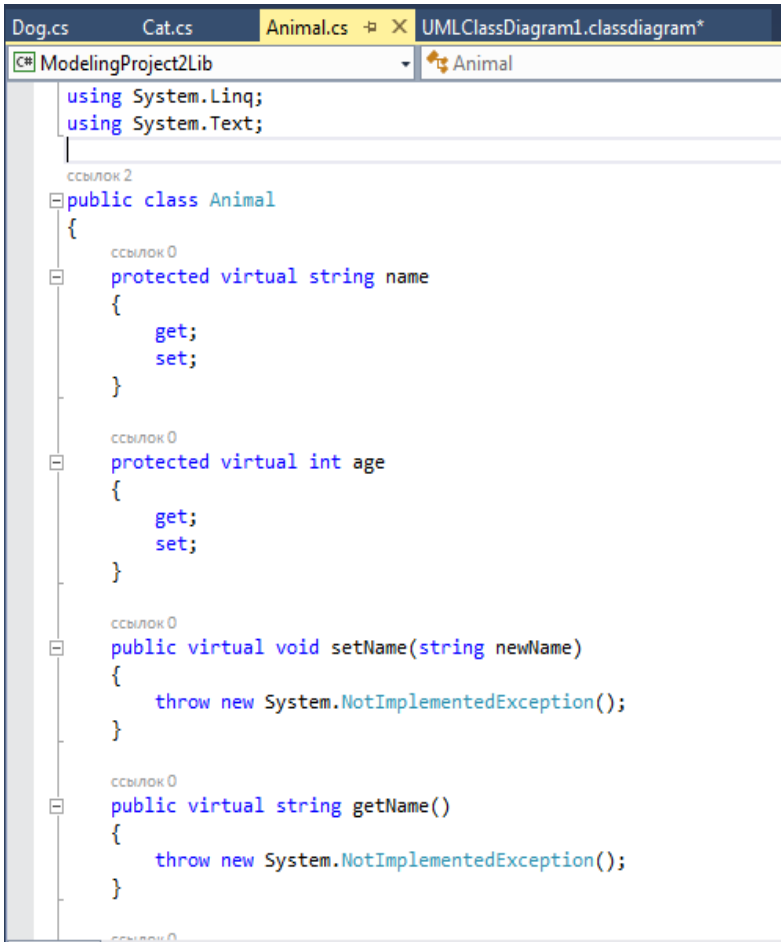


Рисунок 1.11 – Созданный код

1.2.3 BOUML

BOUML – это CASE-средство, предназначенное для автоматизации этапов анализа и проектирования программного обеспечения, а также для генерации кодов на различных языках и выпуска проектной документации. Это UML инструмент, который позволяет строить различные диаграммы и затем по ним определять и генерировать код на C++, Java, PHP, Python и IDL[19].

Основные достоинства программы:

- работает под Linux, MacOS, Windows;
- бесплатное распространение в сети;
- является расширяемым;
- может автоматически генерировать код из диаграмм классов UML и в наоборот.
- обладает высоким быстродействием и не требует много памяти.

В результате разработки в среде BOUML получаем следующие документы:

- диаграммы UML;
- спецификации классов, объектов, атрибутов и операций;
- файлы типа .h и .cpp.

Общий вид BOUML отличается от двух предыдущих программ. Окно состоит из трех частей (см. рисунок 1.12). Левое подокно отображает элементы проекта.

Нижнее правое подокно используется для отображения и изменения комментариев, связанных с выбранным элементом. Верхняя правая часть рабочего окна используется для отображения и изменения диаграмм [19].

Недостатки программы:

– возникли небольшие трудности с построением и отображением UML диаграмм;
– с 5.0 до 6.12 версии необходимо было приобрести лицензию. С 7 версии программа стала бесплатной. При дальнейшем использовании может вновь стать платной.

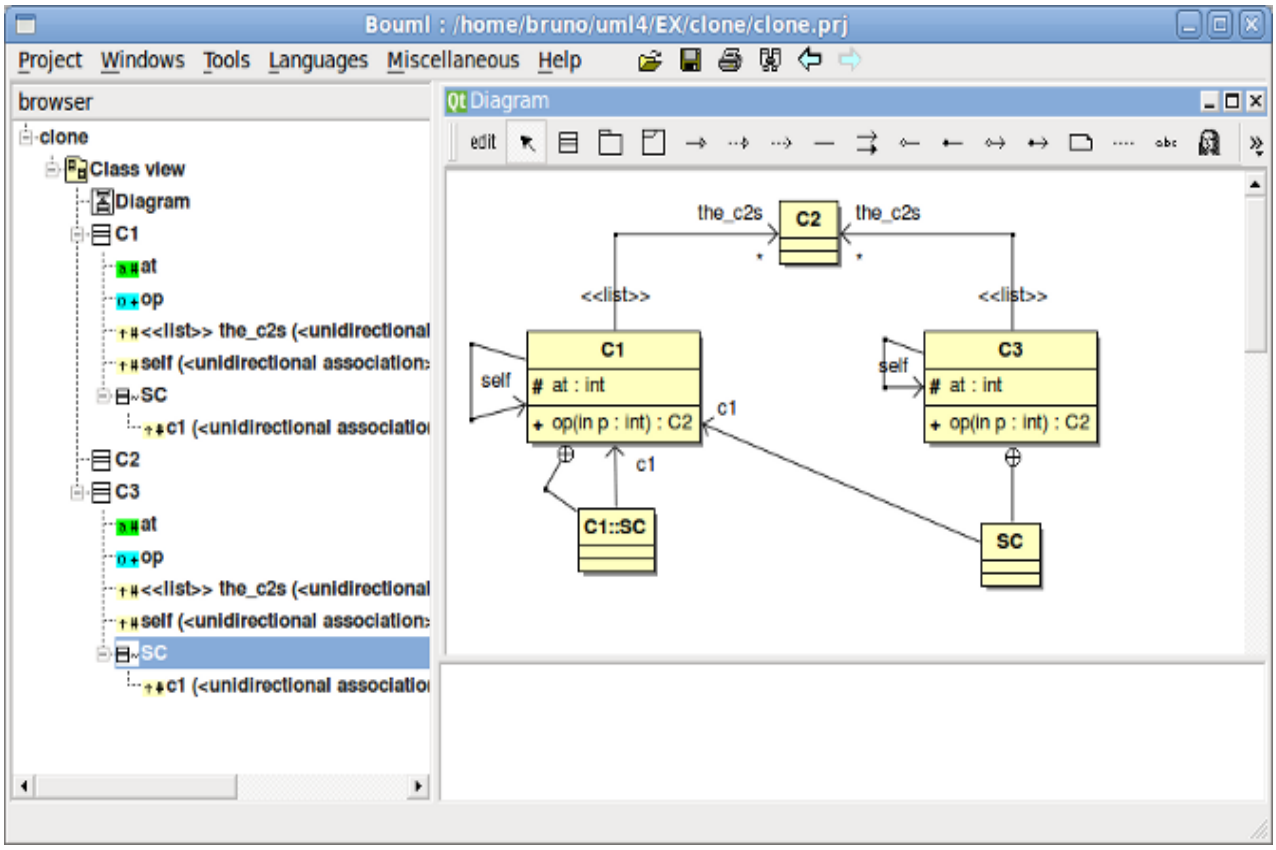


Рисунок 1.12 – Интрефейс BOUML

Пример работы программы.

Перед тем как начать работать в среде BOUML, необходимо было изучить руководство пользователя. При первом запуске программы отображается диалоговое окно среды, где необходимо ввести собственный идентификатор. После этого можно работать в программе.

На рисунке 1.13 представлено изображение созданной диаграммы. В разделе редактора есть область, где можно прописать код для методов класса (см. рисунок 1.14). Пропишем там «return age» и посмотрим, как отразится на сгенерированном коде. На рисунке 1.15 приведены результаты работы программы. В файле Animal.cpp отразилась строчка, которую прописали вручную (см. рисунок 1.16).

В итоге, получилось 6 файлов, что при создании большего количества элементов диаграммы, будет увеличено количество файлов вдвое. BOUML справился со своей задачей.

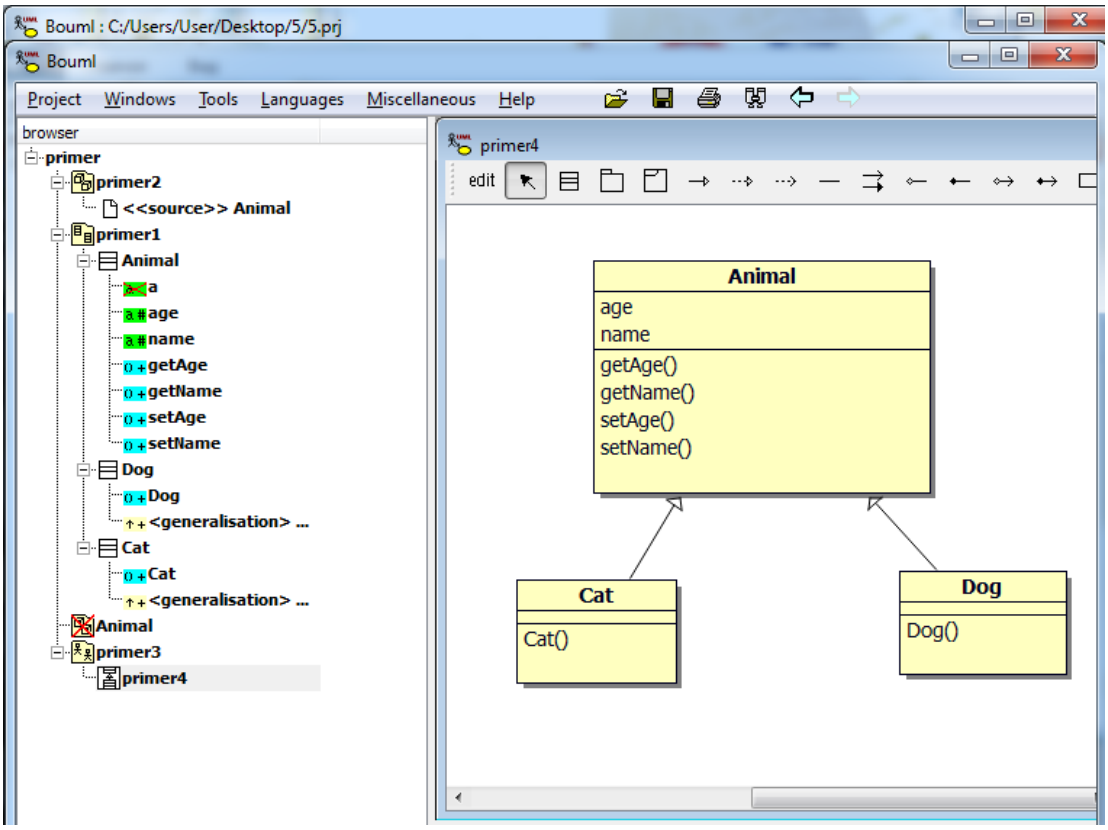


Рисунок 1.13 – Диаграмма классов в BOUML

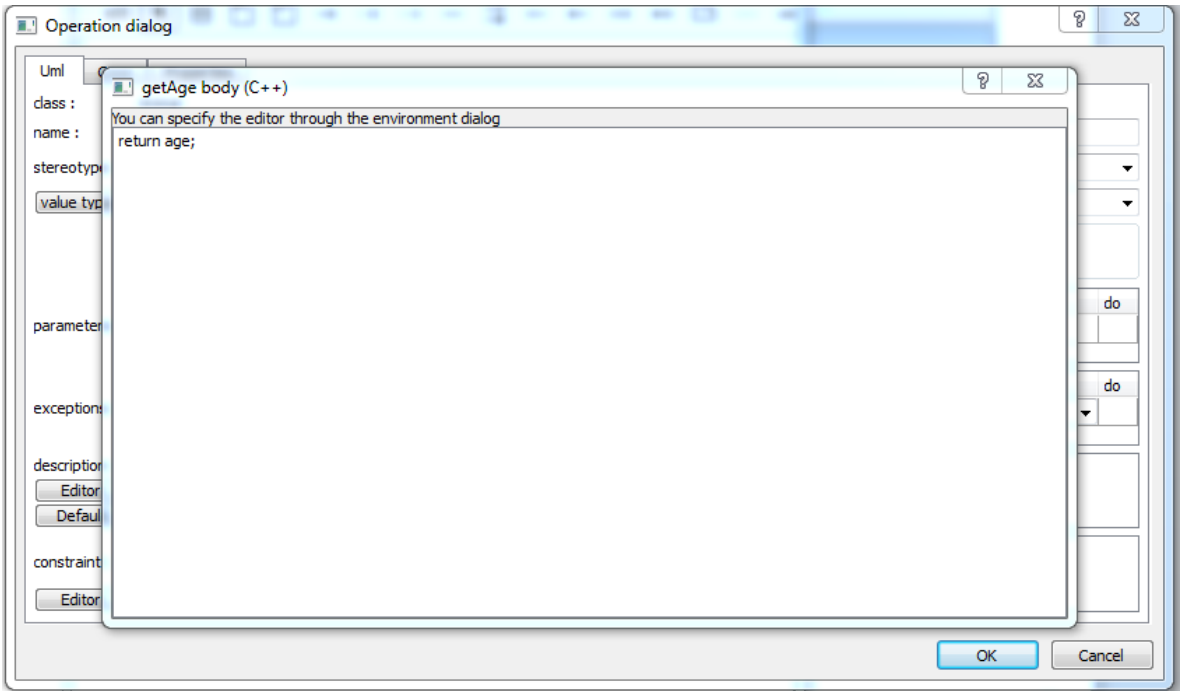


Рисунок 1.14 – Edit Body C++

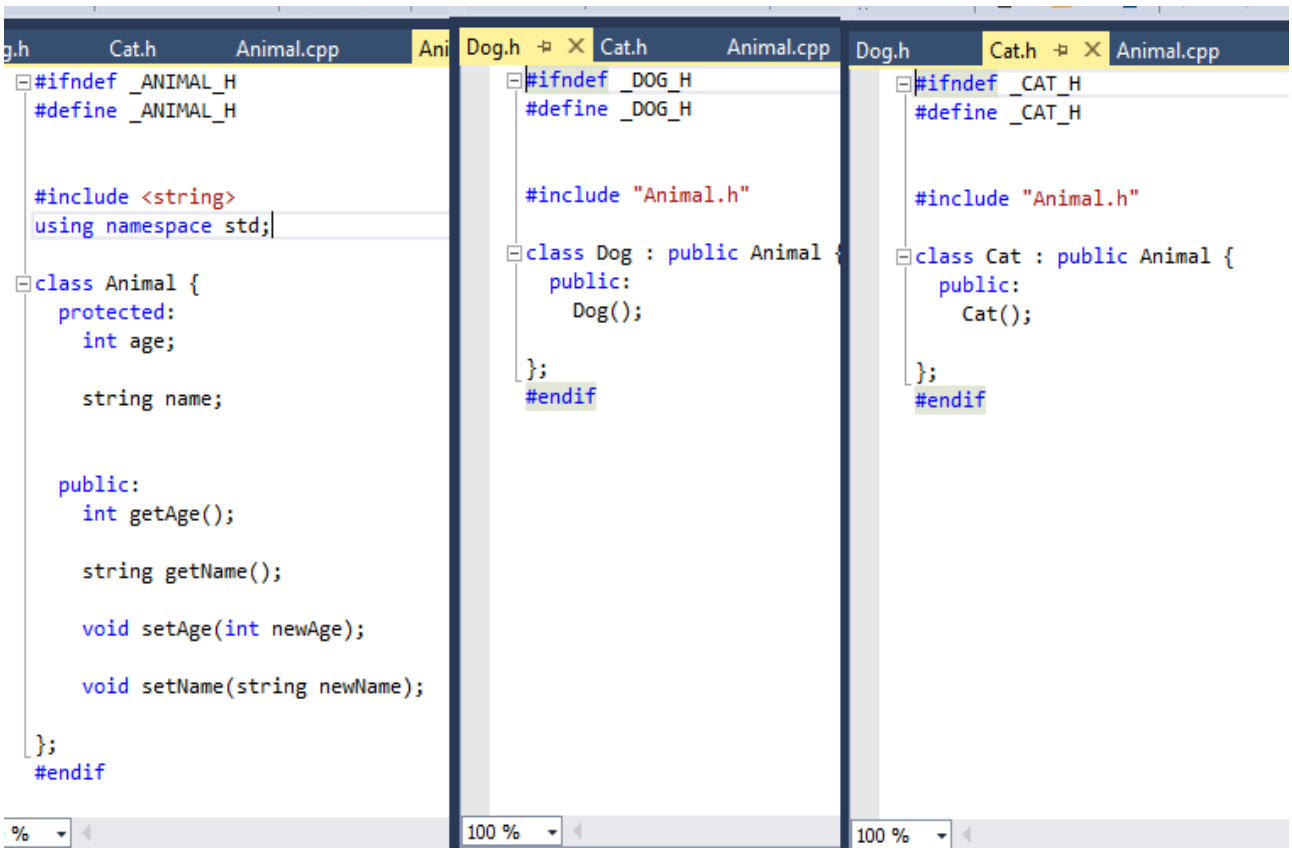


Рисунок 1.15 - Результат генерации кода

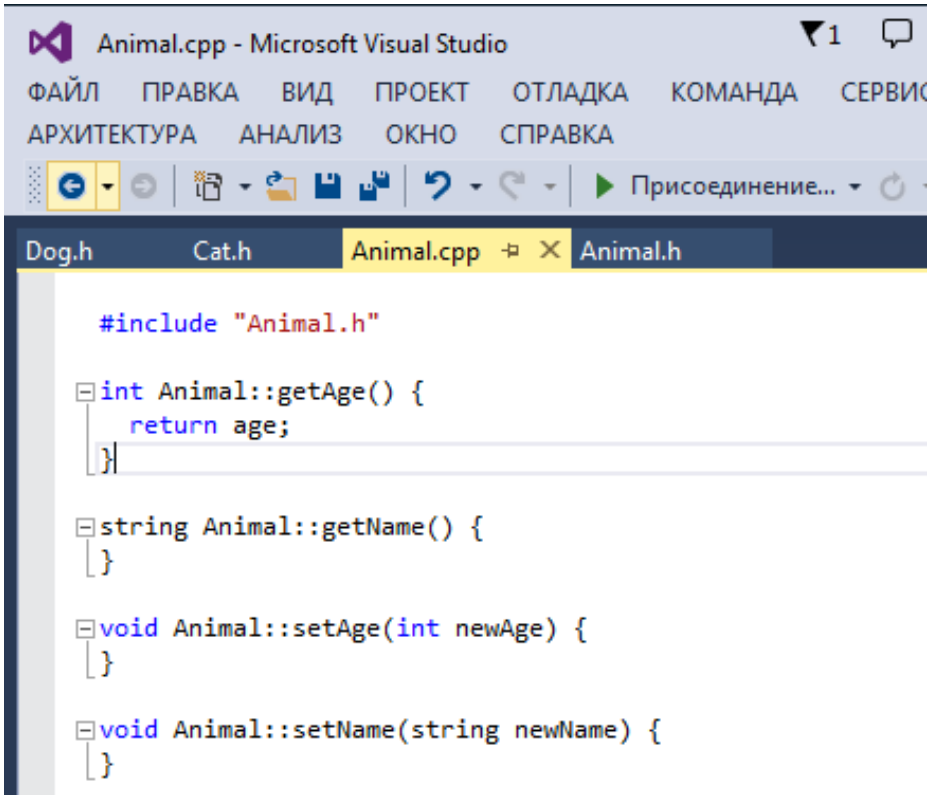


Рисунок 1.16 – Файл Animal.cpp

1.3 Обзор платформы Umlet

Umlet – это инструмент для быстрого создания диаграмм UML. Распространяется по общедоступной лицензии GNU. Поддерживает все типы диаграмм. Имеет собственный формат – UXF. Программа может работать автономно или как плагин Eclipse.

1.3.1 Интерфейс

Область редактора: здесь располагаются компоненты диаграммы, с которыми предполагается взаимодействие. Для них можно применить стандартные операции (сохранить, скопировать, удалить и т.п.). Если открыто более одного файла, они отображаются в виде вкладок.

Область компонентов: с помощью выпадающего списка выбирается соответствующий тип диаграмм, а под ним отображаются его компоненты. При нажатии мыши двойным щелчком на них, элемент появляется в области редактора.

Область редактирования текста. При включении программы, в этой области находится небольшая справка для работы с компонентами. Чтобы производить изменения, необходимо просто нажать на соответствующий элемент диаграммы.

1.3.2 Формат UXF для обмена проектами в виде файлов.

UML eXchange Format (UXF) представляет собой формат обмена данными на основе XML для унифицированного языка моделирования (UML), который является стандартным языком моделирования программного обеспечения. UXF представляет собой структурированный формат, описанный в 1998 году [4].

На рисунке 1.17 представлено содержимое файла в виде XML-документа. Сам документ состоит из текста и разметки.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<diagram program="umlet" version="14.3.0-SNAPSHOT">
  <zoom_level>10</zoom_level>
  <element>
    <id>UMLClass</id>
    <coordinates>
      <x>20</x>
      <y>20</y>
      <w>100</w>
      <h>30</h>
    </coordinates>
    <panel_attributes>SimpleClass</panel_attributes>
    <additional_attributes/>
  </element>
</diagram>
```

Рисунок 1.17 – Листинг XML-документа

Расширение XML — это конкретная грамматика, созданная на базе XML и представленная словарем тегов и их атрибутов, а также набором правил, определяющих какие атрибуты и элементы могут входить в состав других элементов [5].

Корневым элементом документа является <diagram> (см. рисунок 1.18). Он содержит два атрибута: информацию о названии программы и её версии.

```
<diagram program="umlet" version="14.3.0-SNAPSHOT">
```

Рисунок 1.18 – Листинг корневого элемента

Элемент <diagram> имеет двух потомков: <zoom_level> и <element>. В работе будем использовать данные только одного потомка <element> (см. рисунок 1.19).

```
<id>
<coordinates>
<panel_attributes>
<additional_attributes>
```

Рисунок 1.19 – Листинг потомков элемента

На рисунке 1.20 показано вложение элемента <coordinates>.

```
<x>120</x>
<y>30</y>
<w>190</w>
<h>30</h>
```

Рисунок 1.20 – Листинг вложения

Следующая древовидная структура представляет собой вышеуказанный XML-документ (см. рисунок 1.21).

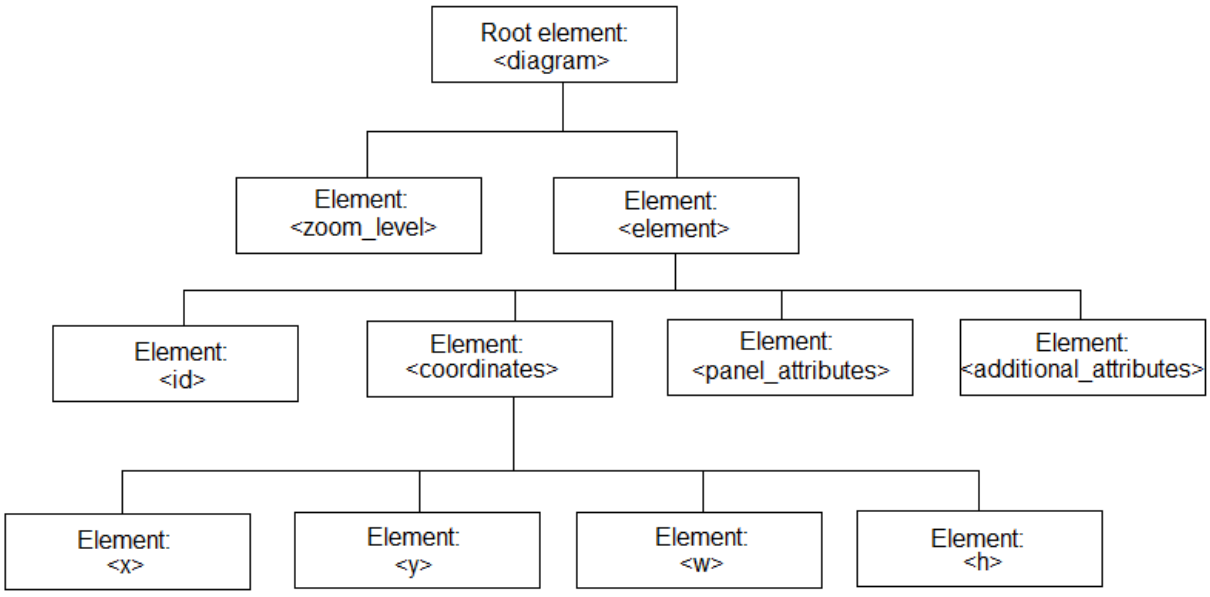


Рисунок 1.21 – Дерево XML-документа

1.4 Постановка задачи

Основной целью квалификационной работы является разработка программы построения интерфейса классов по UML-диаграмме для редактора Umlet.

Пользователь должен создать диаграмму классов, а затем сохранить ее в файле с расширением UFX. Он будет подан на вход программе. На выходе должен получиться исходный код на языке C++, записанный в файл типа .h (заголовочной файл, содержащий описание класса).

В данной работе будем использовать только самые основные компоненты для работы с классами.

1.5 Вывод по разделу

В данном разделе на примере одной задачи были рассмотрены существующие решения по построению моделей UML и ее генерации. Проводилось тестирование трех программ:

- StarUML
- Visual Studio Ultimate 2013
- BOUML.

В результате были отмечены недостатки каждого приложения. Также был проведен обзор платформы Umlet, в ходе которого было рассмотрено построение файла UXF. В конечном итоге была поставлена задача.

2 ТРЕБОВАНИЯ К ГЕНЕРАТОРУ

2.1 Общая характеристика языка UML

UML является языком широкого профиля, это – открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью. UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования[6].

В стандарт входят следующие виды диаграмм:

- диаграмма вариантов использования (use case diagram);
- диаграмма классов (class diagram);
- диаграмма состояний (state machine diagram);
- диаграмма деятельности (activity diagram);
- диаграмма последовательности (sequence diagram);
- диаграмма пакетов (package diagram);
- диаграмма коммуникации (communication diagram);
- диаграмма компонентов (component diagram);
- диаграмма объектов (object diagram);
- диаграмма развертывания (deployment diagram);
- диаграмма внутренней структуры (composite structure diagram);
- диаграмма синхронизации (timing diagram);
- обзорная диаграмма взаимодействия (interaction overview diagram).

2.1.1 Понятие диаграмм классов в UML

Диаграмма классов – это основной способ представления структуры программной системы. Она не отображает динамическое поведение объектов и состоит из множества элементов, которые в совокупности можно использовать для разработки программного кода. Диаграмма классов является статической структурной моделью проектируемой системы, которые не зависят от времени.

2.2 Классы UML

Классом (class) называется описание совокупности объектов с общими атрибутами (полями), операциями (методами), отношениями и семантикой. Графически класс изображается в виде прямоугольника (см. рисунок 2.1). У него должно быть имя, отличающее его от других классов. Имя класса – это текстовая строка [7].

Обязательным элементов обозначения класса является его имя. На начальных этапах разработки диаграммы отдельные классы могут обозначаться простым прямоугольником с указанием только имени соответствующего класса. По мере проработки отдельных компонентов диаграммы описания классов дополняются атрибутами и операциями.

Четвертая секция не обязательна и служит для размещения дополнительной информации справочного характера, например, об исключениях или ограничениях класса, сведения о разработчике или языке реализации. Предполагается, что окончательный вариант диаграммы содержит наиболее полное описание классов, которые состоят из трех или четырех секций [6].

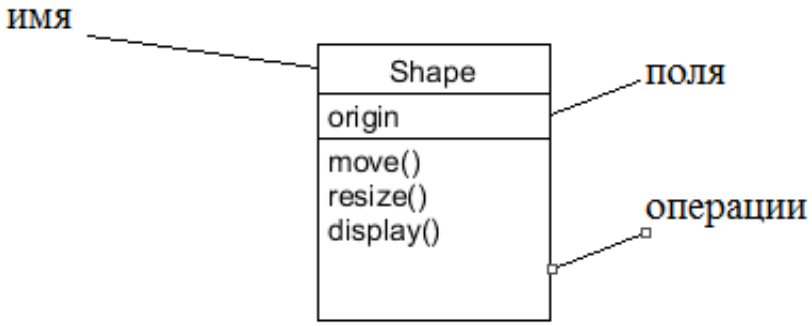


Рисунок 2.1 – Диаграмма класса

2.3 Поля класса и их структура

Поле – это именованное свойство класса, включающее описание множества значений, которые могут принимать экземпляры этого свойства. Класс может иметь любое число полей или не иметь их вовсе. Поле представляет некоторое свойство моделируемой сущности, общее для всех объектов данного класса. Они представлены в разделе, который расположен под именем класса; при этом указываются только их имена [8].

Общий формат записи отдельного поля, который будет применен для данной работы:

<поле> ::= [<квантор видимости>]<имя ><: тип>[= <исходное значение>]

Все элементы, стоящие в квадратных скобках «[]», являются необязательными спецификациями полей.

- <квантор видимости> ::= ‘+’|’-‘|’#’;
- <имя > - это имя поля;
- <: тип> - классификатор, определяющий тип свойства;
- <исходное значение> - значение по умолчанию.

В таблице 2.1 приведены форматы записей полей и соответствующий им код на языке C++.

Таблица 2.1

Примеры программной поддержки записей полей

Спецификация поля в UML	Генерируемый код
count: int	private: int count;
+name: string	public: string name;
#flag: bool = true	protected: flag = true;

2.3.1 Статические поля класса

Если поле объявлено с модификатором static, то существует только одна копия статистического элемента в классе. Такие поля удобно применять для присвоения уникального идентификатора. В UML статические поля подчеркиваются чертой снизу.

Таблица 2.2

Примеры программной поддержки статистических полей

Спецификация поля в UML	Генерируемый код
ID: int	private: static int ID
-ClassAttribute: double	private: static double ClassAttribute;

2.4 Кванторы видимости

Для каждого поля можно задать видимость, которая показывает доступно ли данное поле для других классов.

«+» – public (поле видно всем классам).
«#» – protected (любой потомок данного класса может пользоваться его защищенными свойствами).
«-» – private. (поле с этой областью видимости недоступен или не виден для всех классов без исключения) [9].

2.5 Методы класса и их структура

Методом называется реализация услуги, которую можно запросить у любого объекта класса для воздействия на поведение. Иными словами, операция - это абстракция того, что позволено делать с объектом. У всех объектов класса имеется общий набор операций. Класс может содержать любое число операций или не содержать их вовсе. Операции класса изображаются в разделе, расположенном ниже раздела с полями [10].

Общий формат записи отдельного метода, который будет применен для данной работы:

<операция> ::= [<квантор видимости>]<имя >([<список параметров>]) <: выражение типа возвращаемого значения> [= <исходное значение>]

Все элементы, стоящие в квадратных скобках «[]», являются необязательными спецификациями операций.

- <квантор видимости> ::= ‘+’|’-‘|’#’;
- <имя > - это имя операции;
- <список параметров> ::= <параметр> [, <параметр>] *
- <: выражение типа возвращаемого значения> – тип возвращаемого результата;
- <исходное значение> – значение по умолчанию.

В таблице 2.3 приведены форматы записей методов и соответствующий им код.

Таблица 2.3

Примеры программной поддержки методов

Спецификация метода в UML	Генерируемый код
show()	private: void show();
+print()	public: void print();
+getName(): string	public: string getName();
setName(newName: string)	private: void setName(string newName);
#setID(id: int, name: string)	protected: void setID(int id, string name)
ClassName() - конструктор	private: ClassName()
~ClassName() - деструктор	privete: ~ClassName()

2.5.1 Статические методы

Статические методы – функции, которые не требуют экземпляра класса и объявляются с модификатором static . Они могут использовать только статические поля и изменять их. В UML такие методы записываются с подчеркнутой чертой снизу. Рассмотрим пример в таблице 2.4.

Таблица 2.4

Примеры программной поддержки статистических методов

Спецификация метода в UML	Генерируемый код
operation()	private: static void operation();
+f1(): void	public: static void f1();

2.5.2 Абстрактные методы

В объектно-ориентированном программировании – это чистые виртуальные функции. Они объявляются в базовом классе и не имеют определения. Чистый виртуальный метод записывается наклонным шрифтом. В таблице 2.5 приведен пример.

Таблица 2.5

Примеры программной поддержки абстрактных методов

Спецификация метода в UML	Генерируемый код
abstractMethod(): void	private: virtual void abstractMethod()=0;
+f1()	public: virtual void f1()=0;

2.6 Отношения между классами

Связь представляет собой семантическую взаимосвязь между классами. Она дает классу возможность узнавать о полях, операциях и связях другого класса [11]. Графически она представлена линией, тип которой зависит от вида отношения [12].

2.6.1 Ассоциация

Ассоциация (association) – это семантическая связь между классами. В C++ это отношение можно не интерпретировать.

Они могут быть двунаправленными или однонаправленными (см. рисунок 2.2). На языке UML двунаправленные ассоциации рисуют в виде простой линии без стрелок. На однонаправленной ассоциации изображают только одну стрелку, показывающую ее направление[13].

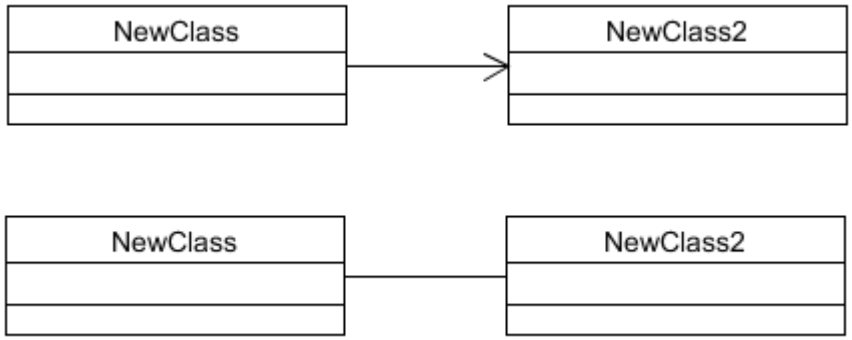


Рисунок 2.2 – Ассоциация

Ассоциация может иметь имя, которое описывает отношение между классами. Чтобы избежать двусмысленности в понимании имени, необходимо обозначить направление чтения связи (см. рисунок 2.3).

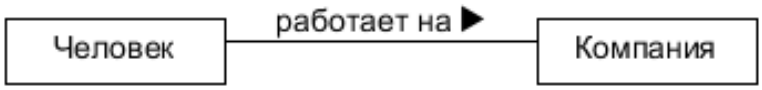


Рисунок 2.3 – Имя ассоциации

Также на диаграмме можно указать роль, которую играет класс в ассоциации. Имя роли выглядит в виде текста рядом с концом связи для соответствующего класса (см. рисунок 2.4).

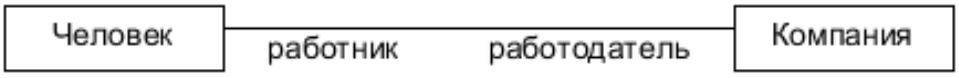


Рисунок 2.4 – Роли

Кратность конца ассоциации показывает сколько объектов класса должно соответствовать каждому объекту на противоположном конце (см. рисунок 2.5). Кратность можно не указывать.

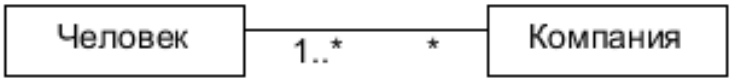


Рисунок 2.5 – Кратность

2.6.2 Зависимость

Зависимость (dependency) – отношение между классами объектов, которое говорит, что один объект использует другой (зависит от него). В C++ это отношение явно никак не объявляется. Графически зависимость изображается пунктирной линией со стрелкой (см. рисунок 2.6), направленной от данного элемента на тот, от которого он зависит [14].

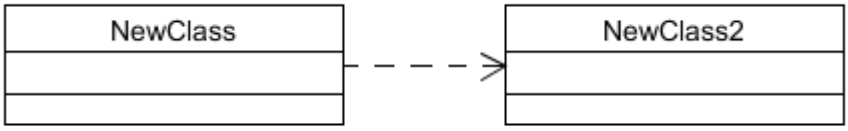


Рисунок 2.6 – Зависимость

2.6.3 Наследование

Наследование (generalization) описывает взаимодействие объектов-потомков (child) и объектов-родителей (parent), возможность их взаимозаменяемости. При этом, в объектно-ориентированном программировании, потомок наследует структуру и поведение своего предка. Графически отношение обобщения представляется в виде линии с не закрашенной стрелкой, указывающей на предка (см. рисунок 2.7) [15].

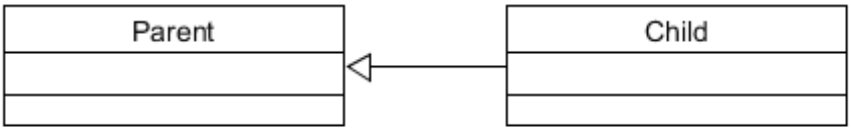
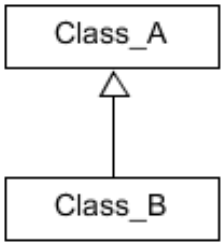


Рисунок 2.7 – Наследование

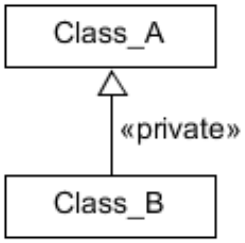
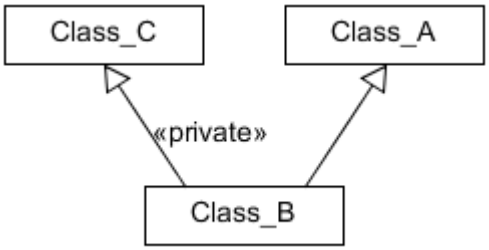
Рядом со связью наследования может размещаться текстовая строка, указывающая на некоторые дополнительные свойства этого отношения. В таблице 2.6 рассмотрены основные требования для генерации кода.

Таблица 2.6

Программная поддержка наследования

Наследование в UML	Генерируемый код	
	Класс	C++
	Class_A	class Class_A{}
	Class_B	class Class_B: public Class_A{}

Продолжение таблицы 2.6

	Class_A	class Class_A{}
	Class_B	class Class_B: private Class_A{}
	Class_A	class Class_A{}
	Class_B	class Class_B: public Class_A, private Class_C{}
	Class_C	class Class_C{}

2.5.4 Агрегация

Агрегация (aggregations) – особая разновидность ассоциации, представляющая структурную связь целого с его частями. Как тип ассоциации, агрегация может быть именованной. Одно отношение агрегации не может включать более двух классов (контейнер и содержимое).

Агрегация встречается, когда один класс является коллекцией или контейнером других. Причем, по умолчанию агрегацией называют агрегацию по ссылке, то есть когда время существования содержащихся классов не зависит от времени существования содержащего их класса. Если контейнер будет уничтожен, то его содержимое – нет.

Графически агрегация представляется пустым ромбом на блоке класса «целое», и линией, идущей от этого ромба к классу «часть» (см. рисунок 2.8) [16].

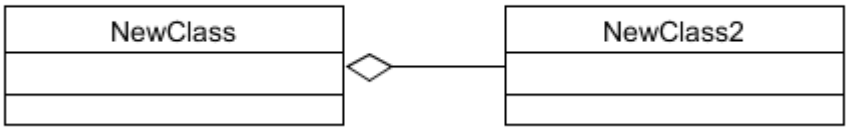


Рисунок 2.8 – Агрегация

Множественность (multiplicity) – это ограничение количества элементов (размера) множества. На диаграмме она может и не быть указана [17]. В таблице 2.7 приведена нотация множественности для связи агрегации, которая будет использоваться при создании программы.

Таблица 2.7

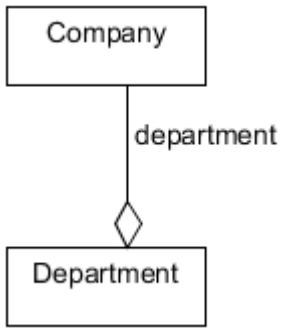
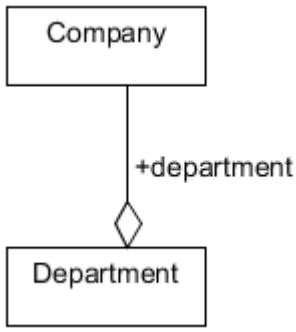
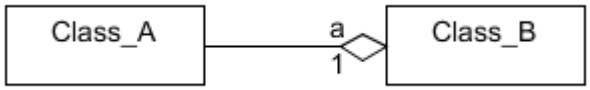
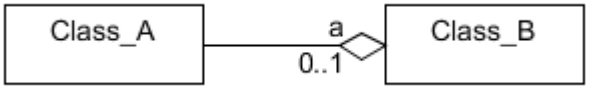
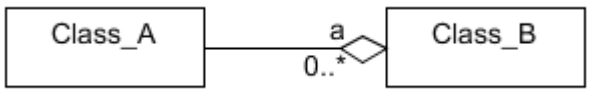
Множественность для агрегации

Множественность в UML	Значение
0..1	ноль или один
1	один
0..*	ноль и больше
1..*	один
*	много

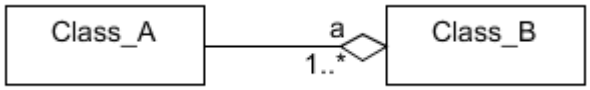
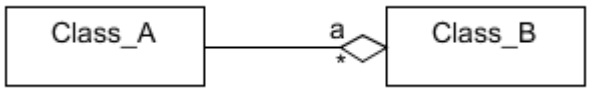
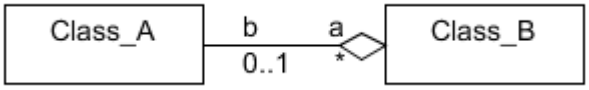
В таблице 2.8 рассмотрены основные требования программной поддержки агрегации в зависимости от их направленности и множественности.

Таблица 2.8

Программная поддержка агрегации

Агрегация в UML	Генерируемый код	
	Класс	C++
	Department	class Department{ private: Company department;};
	Company	class Company {}
	Department	class Department{ public: Company department;};
	Company	class Company {}
	Class_A	class Class_A{ private: Class_B a;};
	Class_B	class Class_B {}
	Class_A	class Class_A{ private: Class_B a;};
	Class_B	class Class_B {}
	Class_A	class Class_A{ private: vector<Class_B> a;};
	Class_B	class Class_B {}

Продолжение таблицы 2.8

	Class_A	class Class_A{ private: vector<Class_B> a;};
	Class_B	class Class_B {}
	Class_A	class Class_A{ private: vector<Class_B> a;};
	Class_B	class Class_B {}
	Class_A	class Class_A{ private: vector<Class_B> a;};
	Class_B	class Class_B { private: Class_A b; }

2.5.5 Композиция

Композиция – более строгий вариант агрегации. Известна также как агрегация по значению. Композиция – это форма агрегации с четко выраженными отношениями владения и совпадением времени жизни частей и целого. Она имеет жёсткую зависимость времени существования экземпляров класса-контейнера и экземпляров содержащихся классов. Если контейнер будет уничтожен, то все его содержимое будет также уничтожено [18].

Графически представляется как агрегация, но с закрашенным ромбиком (см. рисунок 2.9).

Рисунок 2.9 – Композиция

Понятие множественности было приведено выше. В таблице 2.9 приведена нотация уже для связи композиции.

Таблица 2.9

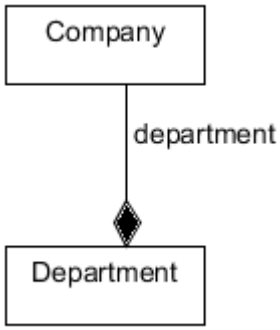
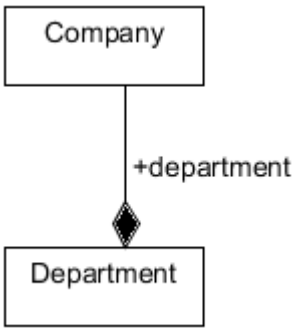
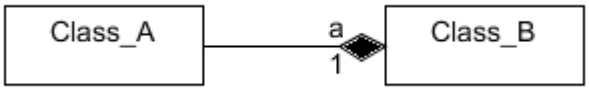
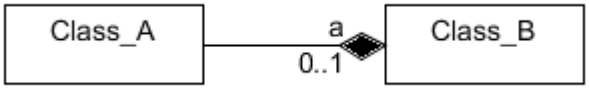
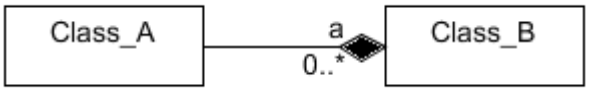
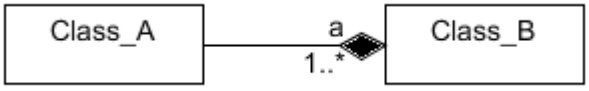
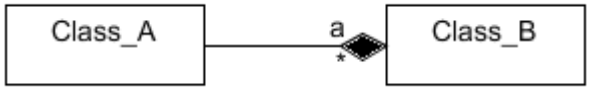
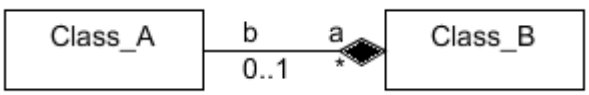
Множественность для композиции

Множественность в UML	Значение
0..1	ноль или один
1	один
0..*	ноль и больше
1..*	один
*	много

В таблице 2.10 приведены требования, которые должна выполнять программа.

Таблица 2.10

Примеры программной поддержки агрегации

Агрегация в UML	Генерируемый код	
	Класс	C++
	Department	class Department{ private: Company department;};
	Company	class Company {}
	Department	class Department{ public: Company department;};
	Company	class Company {}
	Class_A	class Class_A{ private: Class_B a;};
	Class_B	class Class_B {}
	Class_A	class Class_A{ private: Class_B a;};
	Class_B	class Class_B {}
	Class_A	class Class_A{ private: vector<Class_B> a;};
	Class_B	class Class_B {}
	Class_A	class Class_A{ private: vector<Class_B> a;};
	Class_B	class Class_B {}
	Class_A	class Class_A{ private: vector<Class_B> a;};
	Class_B	class Class_B {}
	Class_A	class Class_A{ private: vector<Class_B> a;};
	Class_B	class Class_B { private: Class_A b; }

2.7 Вывод по разделу

Во второй главе были рассмотрены:

- основные понятия унифицированного языка программирования;
- особенности изображения элементов диаграммы;
- нотация каждого элемента диаграммы классов.

На основе этих данных были поставлены требования к генератору.

3 РАЗРАБОТКА ПРОГРАММЫ

3.1 Разработка алгоритмов

При решении данной задачи используется метод пошаговой детализации. На первом этапе описывается общая структура алгоритма без описания отдельных подзадач. На следующем – описываются подзадачи. Процесс продолжается до тех пор, пока алгоритмы подзадач не окажутся очевидными.

3.1.1 Общий алгоритм системы

На рисунке 3.1 представлена схема общего алгоритма.

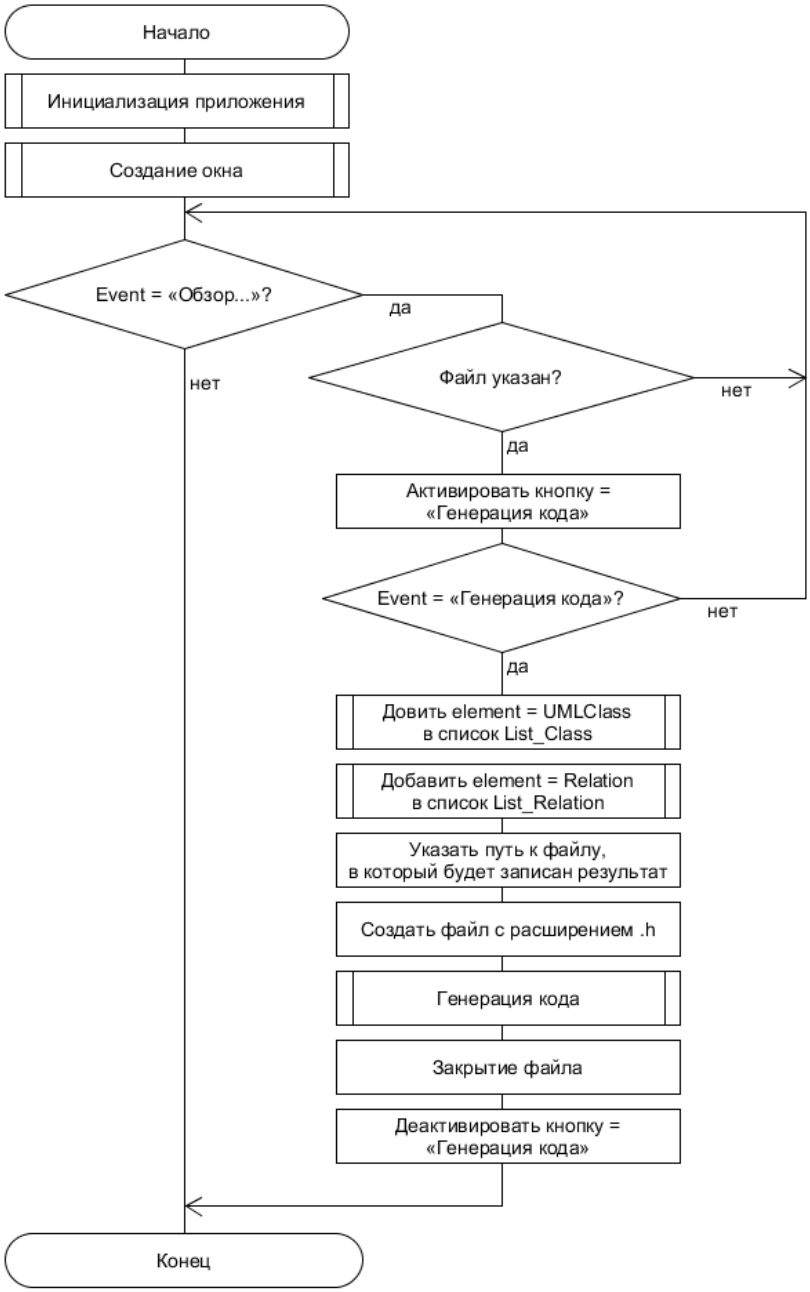


Рисунок 3.1 – Общий алгоритм программы

3.1.2 Добавление информации о классе

Ниже приведен вспомогательный алгоритм, который добавляет элемент с id = UMLClass в список List_Class (см. рисунок 3.2).

На первом шаге инициализируется class_info, в котором будет содержаться информация из файла. Далее добавляем координаты. В panel_attributes данные класса находятся в виде одного текста. С помощью метода Parse(string s) текст разбивается на соответствующие элементы: имя, поля и методы. На конечном этапе класс добавляется в список List_Class.

Рисунок 3.2 – Вспомогательный алгоритм

3.1.3 Добавление информации о связи

На рисунке 3.3 приведена схема алгоритма, которая добавляет информацию о связи в список List_ Relation.

Класс relation_info имеет дополнительные атрибуты, поэтому имеет два метода: ParsePanelAttributes(string s) и ParseAdditionalAttributes(string s).

Аналогичным образом происходит инициализация, добавление координат. Метод с названием ParsePanelAttributes разбивает текст на имя и информацию о связи. ParseAdditionalAttributes находит координаты начала и конца линии связи. После этого находим классы, которые имеют взаимоотношение. В конце добавляем информацию в список.

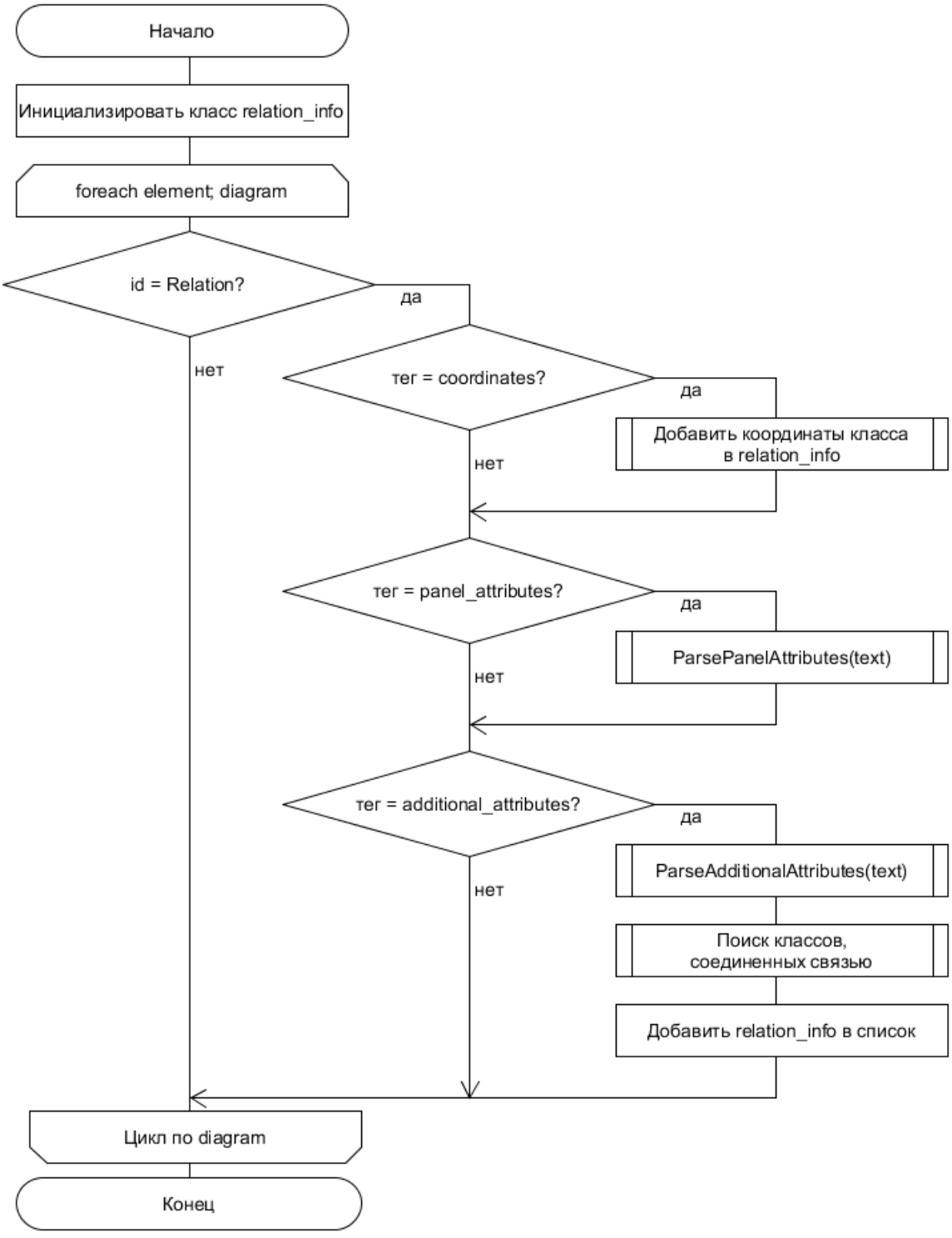


Рисунок 3.3 – Вспомогательный алгоритм

3.1.4 Поиск классов, соединенных связью

На рисунке 3.4 отображена связь между классами.

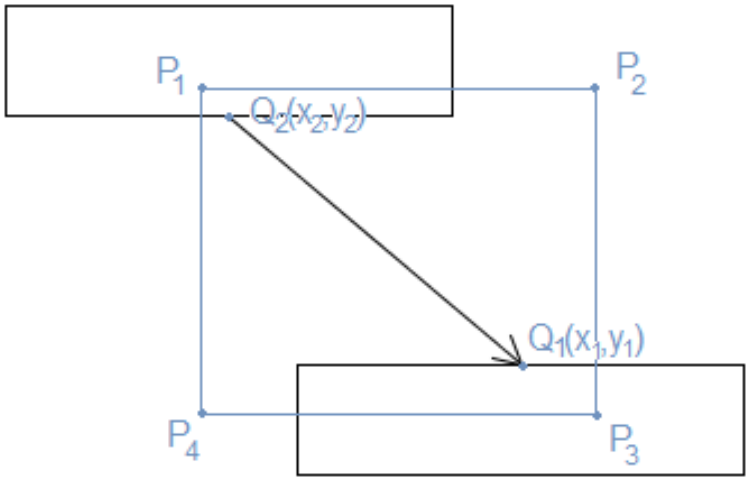


Рисунок 3.4 – Связь между классами

Начальные координаты (,), ширина ( ), высота ( ) и дополнительные атрибуты (, , ,) связи известны. Пусть , , , – вершины прямоугольника, в котором находится связь, соединяющая классы. = (,). Таким образом, координаты стрелки вычисляются по формуле:

Следовательно, и имеют следующий вид:

Если связь принимает форму изогнутой линии (см. рисунок 3.5), то количество дополнительных атрибутов увеличивается на удвоенное количество вершин связи.

Тогда точки пересечения линии связи и классов имеет следующий вид:

где - количество дополнительных атрибутов.

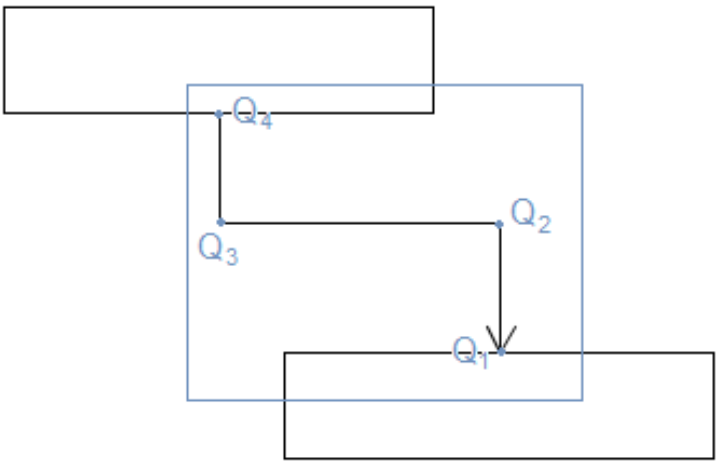


Рисунок 3.5 – Связь между классами в виде изогнутой линии

Условие поиска для первого класса, на который указывает начальная координата связи :

Условие поиска аналогично для второго класса, на который указывает конечная координата связи (см. рисунок 3.4):

Чтобы найти оба классы, содержащие общую связь, необходимо выполнение следующего условия:

Где ,) – координаты первого класса, соответственно его ширина и высота. Аналогично для второго класса. и – координаты начала и конца связи.

Ниже на рисунке 3.6 приведен алгоритм, который реализует поиск этих классов.

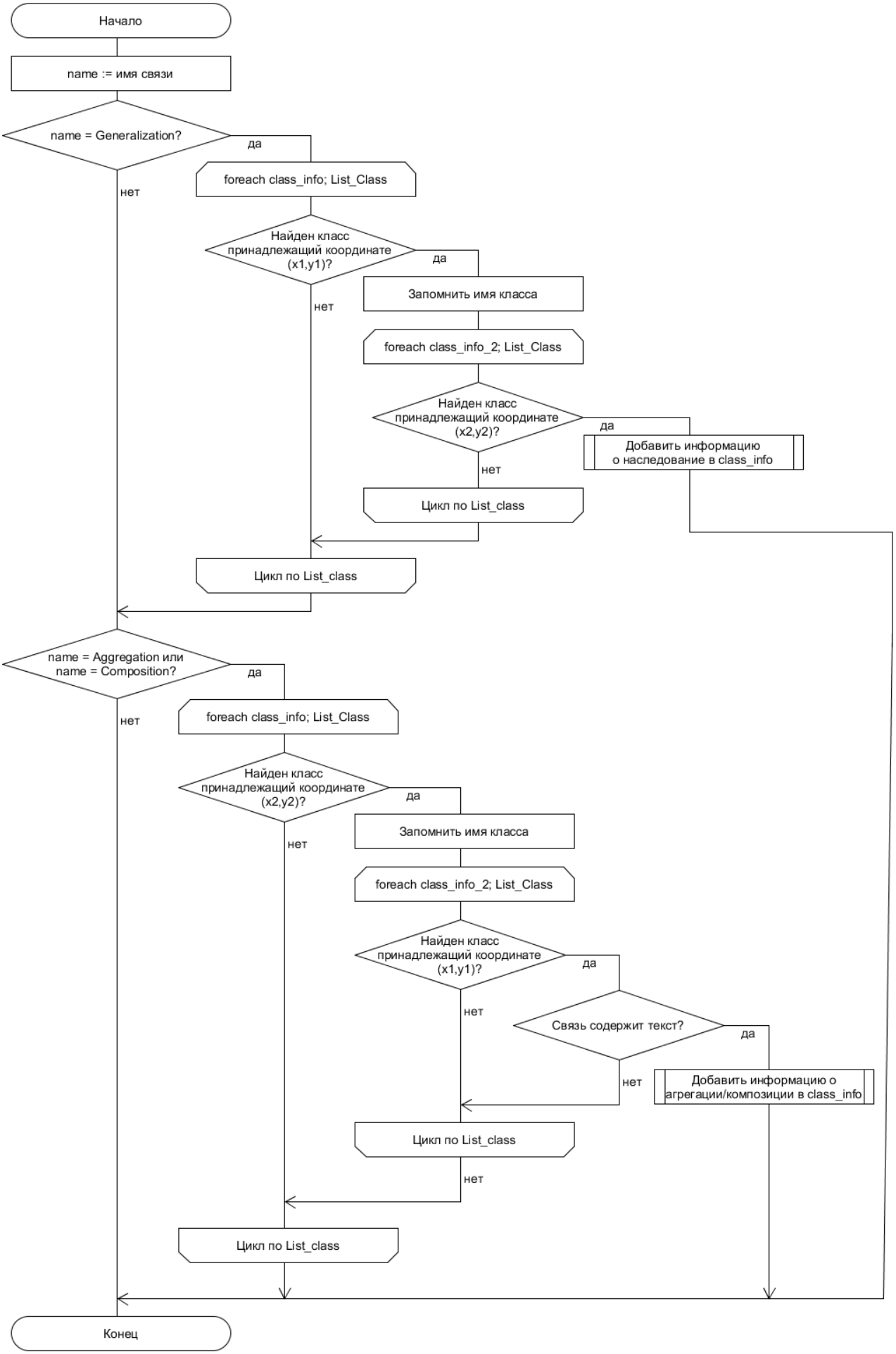


Рисунок 3.6 – Поиск классов

3.1.5 Генерация кода

После того, как вся информация о классах будет заполнена, применяем метод класса Generate(StreamWriter sw). Данный метод генерирует код для всех элементов class_info. На рисунке 3.7 представлен алгоритм метода.

Рисунок 3.7 – Алгоритм генерация кода

3.2 Вывод по разделу

В данном разделе были приведены алгоритмы по реализации программы. Сначала описали общую схему работы, потом описали вспомогательные алгоритмы. На основе этих данных была разработана программа, которая преобразовывает диаграмму классов в заголовочный файл.

4 ТЕСТИРОВАНИЕ ПРОГРАММЫ

4.1 Проверка работоспособности

В процессе тестирования необходимо было показать, что разработанная программа работает безошибочно. Ниже приведены результаты тестов.

4.1.1 Тестирование генерации классов

В редакторе Umlet создадим класс в виде простого прямоугольника (см. рисунок 4.1) и сгенерируем для него код (см. рисунок 4.2). С первым заданием программа справилась.

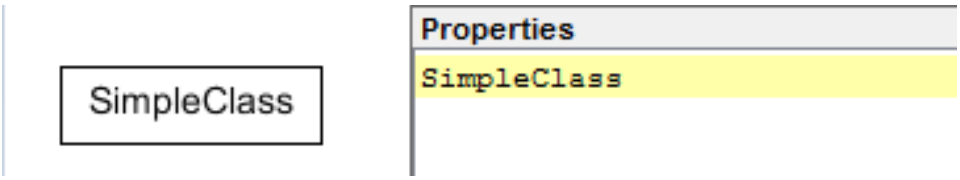


Рисунок 4.1 – Класс в виде простого прямоугольник

```
#ifndef _CLASS_H
#define _CLASS_H


class SimpleClass{
private:
public:
protected:
};


#endif
```

Рисунок 4.2 – Листинг класса без структуры

Теперь усложним задачу и добавим к классу сначала методы, секцию для полей оставим не заполненной (см. рисунок 4.3). Потом оставим секцию для методов пустой и добавим поля (см. рисунок 4.4).

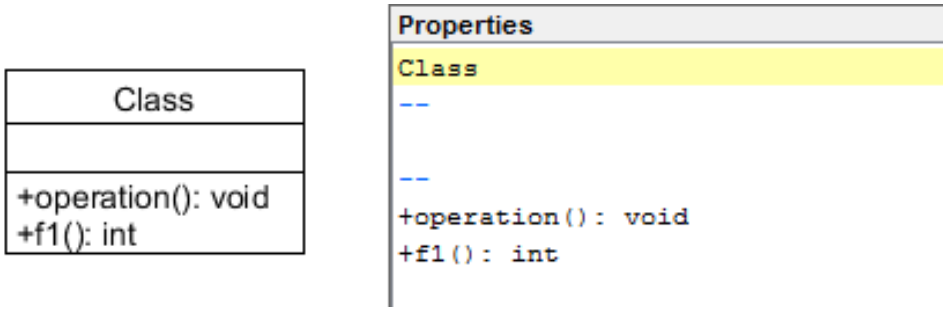


Рисунок 4.3 – Класс дополненный методами

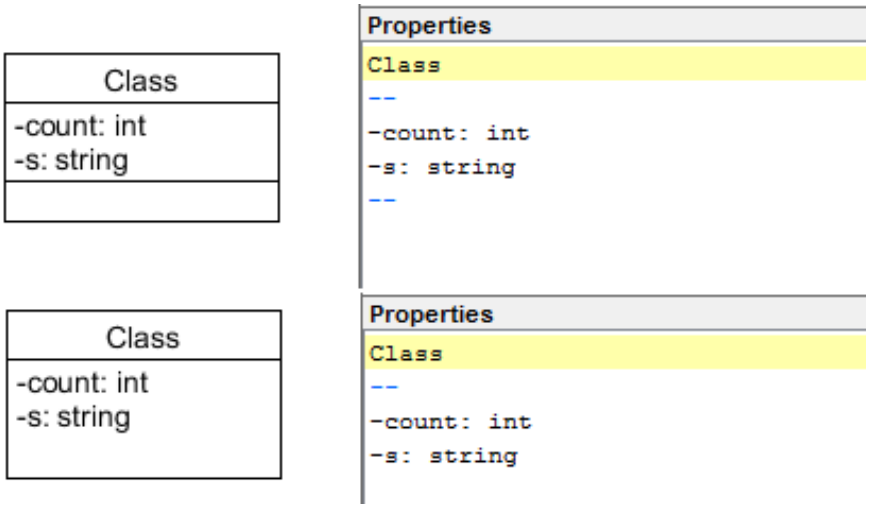


Рисунок 4.4 – Класс дополненный полями

На рисунке 4.5 представлен результат кодогенерации для класса, дополненного методами, а на рисунке 4.6 – для класса с полями.

```
#ifndef _CLASS_H
#define _CLASS_H


class Class{
private:
public:
    void operation();
    int f1();
protected:
};


#endif
```

Рисунок 4.5 – Листинг класса с методами

```
#ifndef _CLASS_H
#define _CLASS_H


class Class{
private:
     int count;
     string s;
public:
protected:
};


#endif
```

Рисунок 4.6 – Листинг класса с полями

Построим класс, который одновременно будет заполнен всеми возможными вариантами записей полей и методов. На рисунке 4.7 представлен пример. На рисунке 4.8 – соответствующий код.

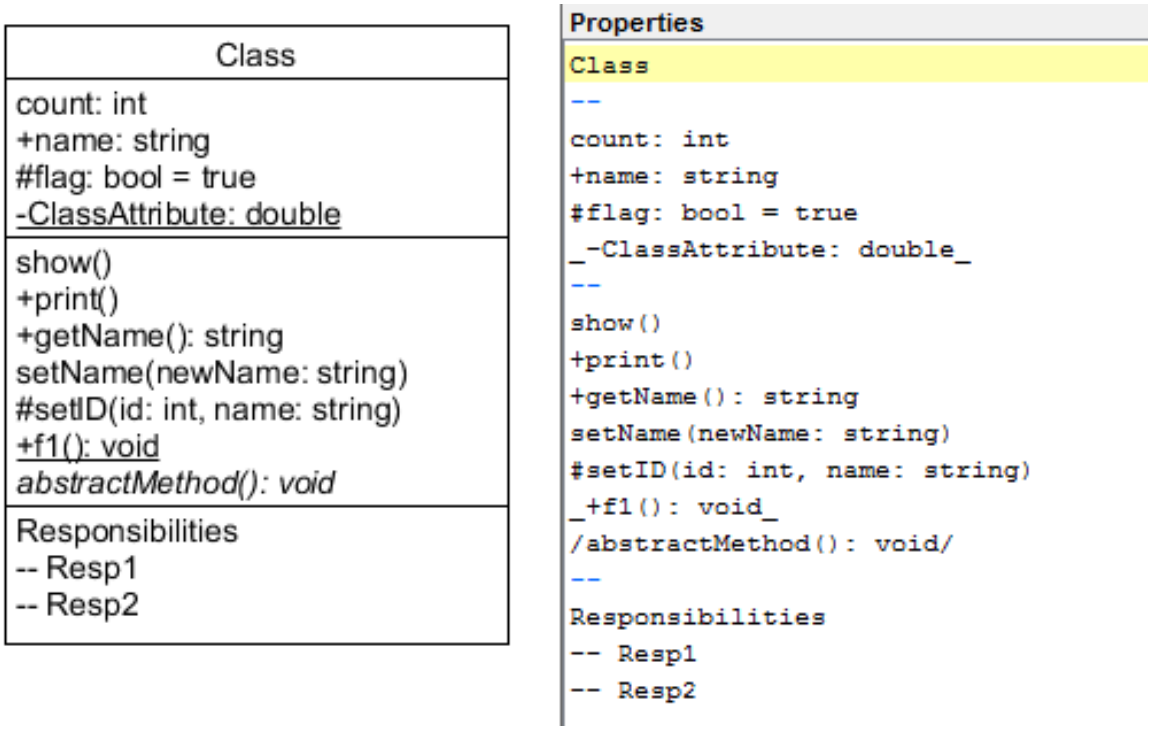


Рисунок 4.7 – Класс с заполненными секциями

```
#ifndef _CLASS_H
#define _CLASS_H


class Class{
private:
    int count;
    static double ClassAttribute;
    void show();
    void setName(string newName);
    virtual void abstractMethod() = 0;
public:
    string name;
    void print();
    string getName();
    static void f1();
protected:
    bool flag =true;
    void setID( int id, string name);
};


#endif
```

Рисунок 4.8 – Листинг примера класса

4.1.2 Тестирование генерации связей между классами

На рисунке 4.9 представлены ассоциация и зависимость соответственно. Надо убедиться, что при генерации эти связи никак не будут влиять на работоспособность программы.

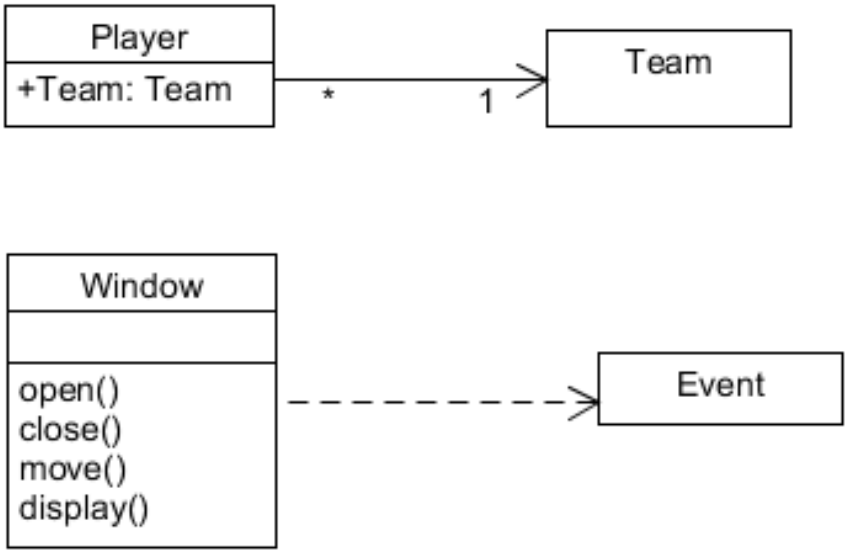


Рисунок 4.9 – Агрегация и зависимость

Ниже представлен получившийся код для первой связи (см. рисунок 4.10). На следующем рисунке 4.11 – листинг для второй связи.

```
#ifndef _CLASS_H
#define _CLASS_H


class Player{
private:
public:
     Team Team;
protected:
};


class Team{
private:
public:
protected:
};


#endif
```

Рисунок 4.10 – Листинг классов, соединенных ассоциацией

```
#ifndef _CLASS_H
#define _CLASS_H


class Window{
private:
    void open();
    void close();
    void move();
    void display();
public:
protected:
};


class Event{
private:
public:
protected:
};


#endif
```

Рисунок 4.11 – Листинг классов, соединенных зависимостью

Теперь протестируем программу, когда отношение между классами – наследование. Построим абстаркный класс GeometricObject, который наследует потомков Circle и Rectangle (см. рисунок 4.12).

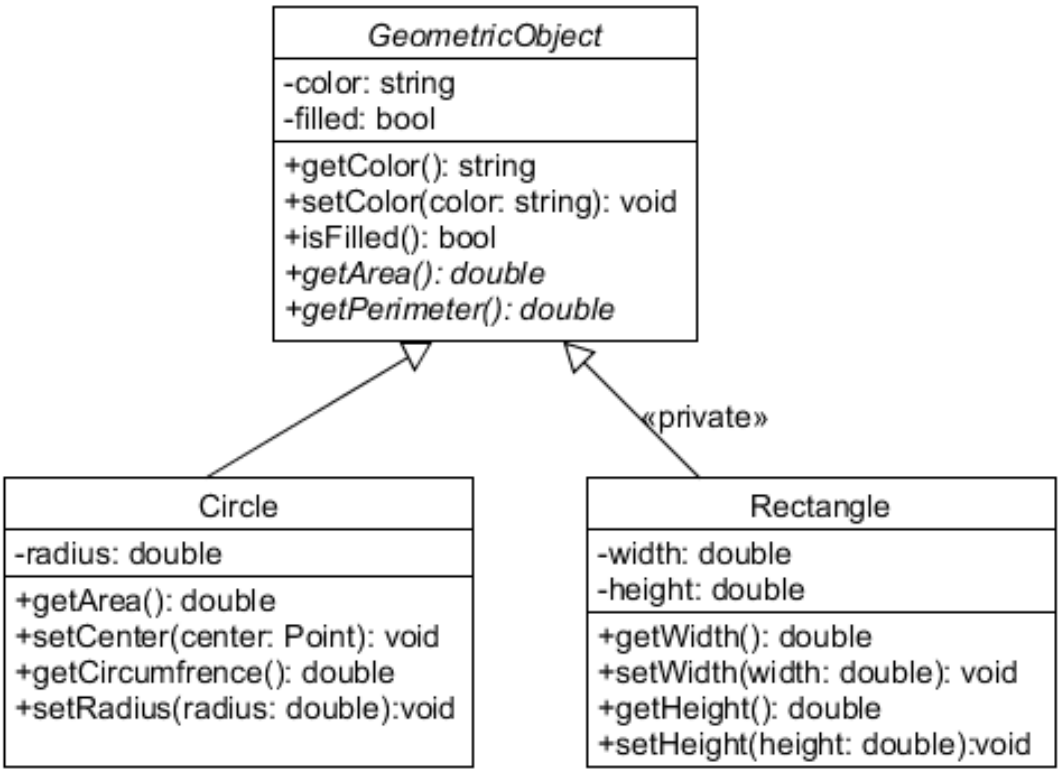


Рисунок 4.12 – Наследование классов

В результате был получен следующий код для данных классов (см. рисунок 4.13).

```
#ifndef _CLASS_H
#define _CLASS_H


class Circle: public GeometricObject{
private:
    double radius;
public:
    double getArea();
    void setCenter(Point center);
    double getCircumfrence();
    void setRadius(double radius);
protected:
};


class GeometricObject{
private:
    string color;
    bool filled;
public:
    string getColor();
    void setColor(string color);
    bool isFilled();
    virtual double getArea() = 0;
    virtual double getPerimeter() = 0;
protected:
};


class Rectangle: private GeometricObject{
private:
    double width;
    double height;
public:
    double getWidth();
    void setWidth(double width);
    double getHeight();
    void setHeight(double height);
protected:
};


#endif
```

Рисунок 4.13 – Листинг наслелование классов

Рассмотрим следующий тест. Агрегация и композиция на языке C++ имеет одинаковое представление кода, поэтому будем рассматривать эти связи вместе. Приведем несколько примеров (см. рисунок 4.14), когда связь имеет:

- роли и кратность с двух концов;
- на одном конце роль, а на другом – кратность;
- на одном конце роль и кратность, на другом – только роль;
- только обычный текст;
- только кратности с двух сторон;
- на одном конце роль имеет модификатор доступа, а на другом – нет.

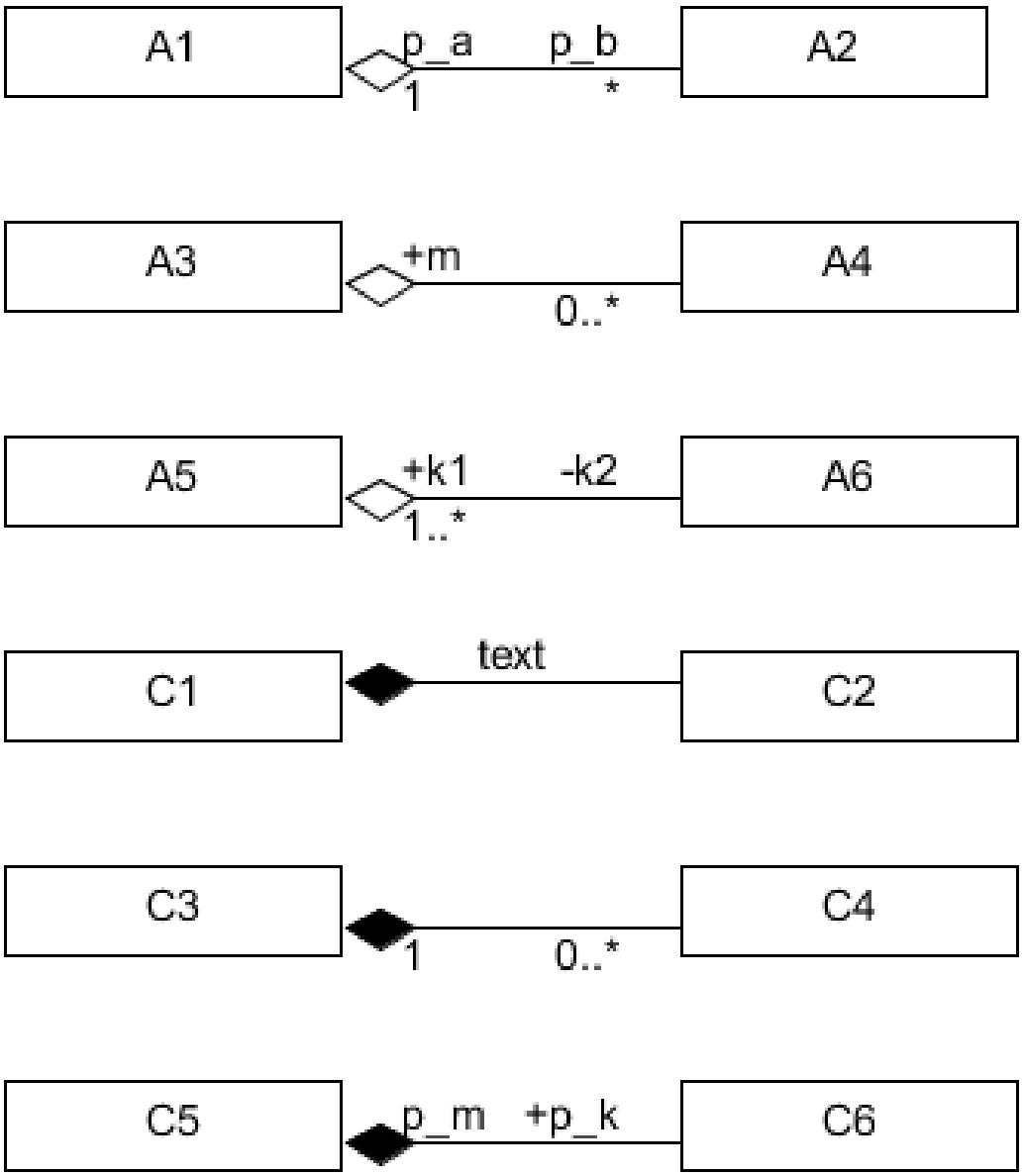


Рисунок 4.14 – Примеры агрегации и композиции

При генерации связи, имеющие кратности без роли и простой текст, никак не влияют на структуру класса. На рисунке 4.15 приведен результат работы программы.

```
#ifndef _CLASS_H
#define _CLASS_H


class A1{
private:
    vector<A2> p_b;
public:
protected:
};


class A2{
private:
    A1 p_a;
public:
protected:
};
class A3{
private:
public:
protected:
};


class A4{
private:
public:
    A3 m;
protected:
};


class A5{
private:
    A6 k2;
public:
protected:
};

class A6{
private:
public:
    vector<A5> k1;
protected:
};
```

```
class C1{
private:
public:
protected:
};


class C2{
private:
public:
protected:
};


class C3{
private:
public:
protected:
};


class C4{
private:
public:
protected:
};


class C5{
private:
public:
    C6 p_k;
protected:
};


class C6{
private:
    C5 p_m;
public:
protected:
};


#endif
```

Рисунок 4.15 – Листинг примеров агрегации и композиции

4.1.3 Общие примеры тестирования программы

В этом разделе рассмотрим пару примеров паттернов проектирования:
- одиночка (см. рисунок 4.16);
- компоновщик (см. рисунок 4.17).

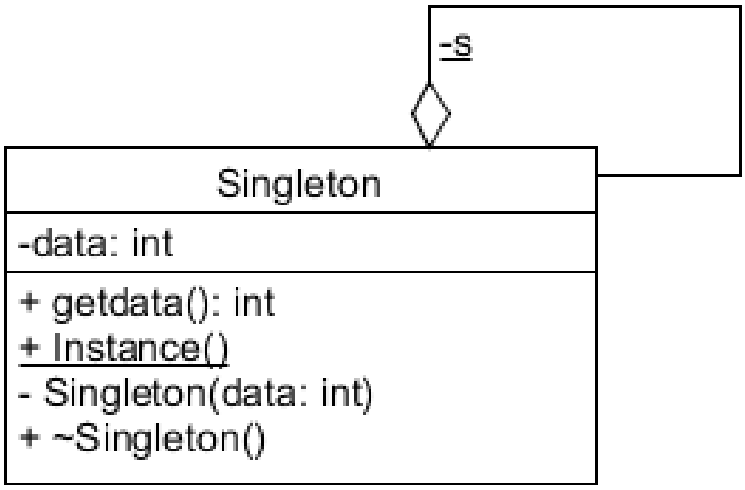


Рисунок 4.16 – Одиночка

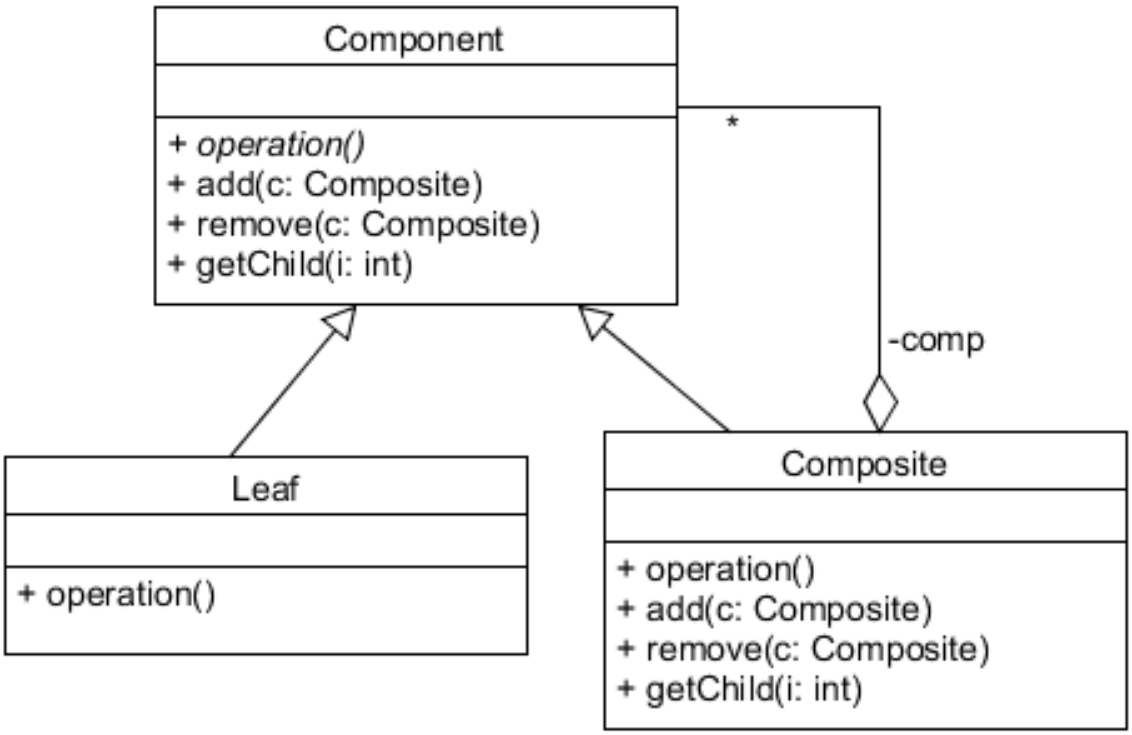


Рисунок 4.17 – Компоновщик

На рисунке 4.18 реализация кода для первого паттерна, на рисунке 4.19 – для второго паттерна проектирования.

```
#ifndef _CLASS_H
#define _CLASS_H


class Singleton{
private:
    int data;
    static Singleton s;
     Singleton(int data);
public:
    int getdata();
    static void Instance();
     ~Singleton();
protected:
};


#endif
```

Рисунок 4.18 – Пример1

```
#ifndef _CLASS_H
#define _CLASS_H


class Leaf: public Component{
private:
public:
    void operation();
protected:
};


class Composite: public Component{
private:
public:
    void operation();
    void add(Composite c);
    void remove(Composite c);
    void getChild(int i);
protected:
};
class Component{
private:
    Composite comp;
public:
    virtual void operation() = 0;
    void add(Composite c);
    void remove(Composite c);
    void getChild(int i);
protected:
};


#endif
```

Рисунок 4.19 – Пример 2

4.2 Вывод по разделу

В данном разделе приведены тесты работоспособности программы. Были протестированы следующие компоненты:

- класс со сложной и простой структурой;
- поля;
- методы;
- связи между классами.

Также привели пару общих примеров с использованием моделей диаграмм. В результате код, полученной с помощью генератора, полностью соответствует поставленным требованиям во второй главе.

ЗАКЛЮЧЕНИЕ

Целью работы была разработка программы построения интерфейса классов по UML-диаграмме для редактора Umlet.

В данной работе был проведен обзор существующих программ, способных генерировать код по диаграмме UML. Были отмечены их достоинства и недостатки. Рассмотрели структура файла Umlet, который использовался для разработки программы.

В ходе квалификационной работы решены следующие задачи:

- разработали требования к генератору;
- разработали алгоритмы работы программы;
- реализовали программу и выполнили её проверку.

В результате был получен генератор программного кода на языке C++ из моделей диаграмм класса. В дальнейшем можно добавить генерацию для шаблонного класса.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Гома, Х. UML. Проектирование систем реального времени, параллельных и распределенных приложений. / Х. Гома.– пер. с англ. – М.: ДМК Пресс, 2011. – 704 с.

2 StarUML. Руководство пользователя [Электронный ресурс] / Режим доступа: https://studfiles.net/preview/6064280/page:2, свободный. – Загл. с экрана.

3 Visual Studio 2013 [Электронный ресурс] / Режим доступа: https://msdn.microsoft.com/ru-ru/library/dd831853(v=vs.120).aspx, свободный. – Загл. с экрана.

4 Описание формата UXF [Электронный ресурс]/ Режим доступа: https://en.wikipedia.org/wiki/UXF, свободный. – Загл. с экрана.

5 XML [Электронный ресурс] / Режим доступа: https://ru.wikipedia.org/wiki/XML, свободный. – Загл. с экрана.

6 Иванов, Д. Ю. Унифицированный язык моделирования UML / Д. Ю. Иванов, Ф. А. Новиков. – Санкт-Петербург: СПбГУ ИТМО, 2010. – 200 с.

7 Буч, Г. Язык UML. Руководство пользователя. / Г. Буч, Д. Рамбо, И. Якобсон. – М.: ДМК Пресс, 2008. – 496 с.

8 Токмаков, Г. П.Автоматизированное проектирование информационных систем: учебное пособие / Г. П. Токмаков. – Ульяновск : УлГТУ, 2015. − 121 с.

9 Синтаксис и семантика основных объектов UML [Электронный ресурс] / Режим доступа: https://www.intuit.ru/studies/courses/2195/55/lecture/1638?page=2, свободный. – Загл. с экрана.

10 Язык UML. Руководство пользователя [Электронный ресурс] / Режим доступа:http://alice.pnzgu.ru:8080/~zsa/sql/titan_zsa/uml_htm_gol/gl_04.htm, свободный. – Загл. с экрана.

11 Павловская, Т.А. С/С++. Структурное и объектно-ориентированное программирование: практикум. / Т.А. Павловская, Ю.А Щупак. - СПб: Питер, 2010 – 352 с.

12 Язык UML Руководство пользователя [Электронный ресурс] / Режим доступа: http://bourabai.ru/dbt/uml/ch5.htm. – Загл. с экрана.

13 Фаулер, М.UML. Основы, 3 е издание.– пер. с англ. – СПб: Символ Плюс, 2018. – 192 с.,

14 Гамма Э.Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж.Влиссидес. - СПб: Питер, 2016. – 368 с.

15 Павловская, Т.А. С++. Объектно-ориентированное программирование: практикум: учебное пособие для вузов / Т.А. Павловская, Ю.А Щупак. – СПб.: Питер, 2008. – 464 с.

16 Леоненков, А. В. Объектно-ориентированный анализ и проектирование с использованием UML и IBM RATIONAL ROSE : Учебное пособие для вузов / А. В. Леоненков. - М. : Интернет-Университет Ин-117 формационных Технологий, 2009. - 318 с.

17 Новиков, Ф.А. Учебно-методическое пособие по дисциплине «Анализ и проектирование на UML» / Ф.А. Новиков. - СПб: СПбГУИТМО, 2008. - 286 с.
18 UML-диаграммы классов - Программирование [Электронный ресурс]/Режим доступа: https://prog-cpp.ru/uml-classes, свободный. – Загл. с экрана.
19 Основные сведения о UML и BOUML [Электронный ресурс] / Режим доступа: http://window.edu.ru/resource/114/80114/files/abramova.pdf#1, свободный. – Загл. с экрана.
20 ГОСТ 19.201 – 78. Техническое задание. Требование к содержанию и оформлению. – М.: Изд-во стандартов, 2004. – 34 с.
21 ГОСТ 19.402 – 78. Описание программы. – М.: Изд-во стандартов, 2000. – 49 с.

ПРИЛОЖЕНИЕ 1 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Назначение и условия применения системы

Программа GenerateC++ предназначена для генерации кода для классов UML. Пользователь должен иметь установленное приложение Umlet.

Подготовка системы к работе

Чтобы начать работу, необходимо скопировать папку с ПО на компьютер и запустить приложение.

Перед тем как начать использовать программу, необходимо создать файл в редактору Umlet. Создаем элементы диаграммы в поле редактора (см. рисунок 5.1), потом сохраняем в формате uxf.

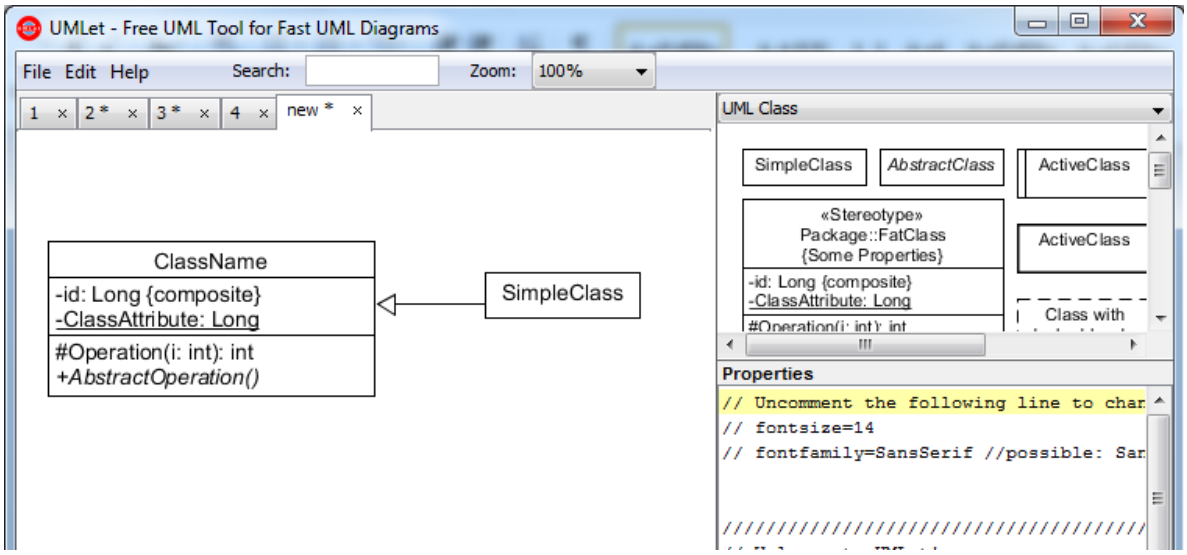


Рисунок 5.1 – Редактор Umlet

Выполнение программы

Запускаем приложение GenetrateC++. Нажимаем на кнопку «Обзор…» (см. рисунок 5.2) и выбираем ранее созданный файл в Umlet. В текстовой строке появляется путь к указанному файлу. После этого нажимаем на кнопку «Сгенерировать код». Открывается диалоговое окно, в котором указываем имя и путь, куда будет сохранен файл.

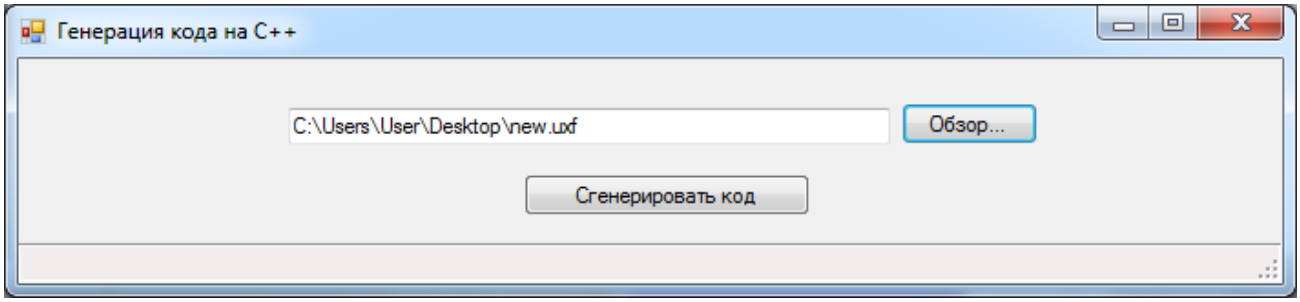


Рисунок 5.2 – Приложение GenerateC++

ПРИЛОЖЕНИЕ 2 ТЕКСТ ПРОГРАММЫ

Main.cs – класс главного окна программы, в котором происходит обработка информации из XML-документа.

```
//-----------------------------------------------------------------------------
//                      Main.cs
//-----------------------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Xml;
using System.Xml.Linq;

namespace WindowsFormsApplication1
{
    public partial class MainForm : Form
    {
       public MainForm()
       {
              InitializeComponent();
              openFileDialog1.Filter = "Text files(*.uxf)|*.uxf";
              openFileDialog1.FileName = "";
        }

    //список классов
    private List<ClassInfo> List_Class = new List<ClassInfo>();
    //список связей между классами
    private List<RelationInfo> List_Relation = new List<RelationInfo>();

//-----------------------------------------------------------------------------
//                      Кнопка "Обзор..."
//-----------------------------------------------------------------------------

    private void button1_Click(object sender, EventArgs e)
    {
       if (openFileDialog1.ShowDialog() == DialogResult.Cancel)
       {
              return;
       }
       string filename = openFileDialog1.FileName;
       textBox2.Text = filename; //Расположение файла
       if (filename != "")
       {
              //Если файл был выбран, то активируем кнопку
              button2.Enabled = true;
        }
    }

//-----------------------------------------------------------------------------
//          Чтение из файла элементов UMLClass, добавление их в список
//-----------------------------------------------------------------------------
```

```
      private void ReadXmlClass()
      {
            try
            {
                  XmlDocument xDoc = new XmlDocument();
                  xDoc.Load(textBox2.Text);
                  toolStripStatusLabel1.Text = "Файл открыт";


                  // получим корневой элемент
                  XmlElement xRoot = xDoc.DocumentElement;
                  string str_id;

                  // обход всех узлов в корневом элементе
                  foreach (XmlNode xnode in xRoot)
                  {
                        str_id = "";
                        ClassInfo class_info = new ClassInfo();

                        // обходим все дочерние узлы элемента element
                        if (xnode.Name == "element")
                        {
                            foreach (XmlNode childnode in xnode.ChildNodes)
                            {
                                if (childnode.Name == "id")
                                    str_id = childnode.InnerXml;
                                //находим все структуры класса
                                if (str_id == "UMLClass")
                                {
                                    if (childnode.Name == "coordinates")
                                    {
                                        foreach (XmlNode child1 in
childnode.ChildNodes)
                                        {
                                            if (child1.Name == "x")
                                                class_info.x=
Convert.ToInt32(child1.InnerXml);
                                            if (child1.Name == "y")
                                                class_info.y =
Convert.ToInt32(child1.InnerXml);
                                            if (child1.Name == "w")
                                                class_info.w =
Convert.ToInt32(child1.InnerXml);
                                            if (child1.Name == "h")
                                                class_info.h =
Convert.ToInt32(child1.InnerXml);
                                        }
                                    }
                                    if (childnode.Name == "panel_attributes")
                                    {
                                        class_info.Parse(childnode.InnerXml);
                                    }
                                    if (childnode.Name == "additional_attributes")
                                    {
                                        //Добавляем в список новый класс
                                        List_Class.Add(new ClassInfo()
                                        {
                                            x = class_info.x,
                                            y = class_info.y,
                                            w = class_info.w,
                                            h = class_info.h,
                                            name = class_info.name,
```

```
                                        fi = class_info.fi,
                                        mi = class_info.mi
                                    });
                                }
                            }
                        }
                    }
                }
                catch (Exception ex)
                {
                    toolStripStatusLabel1.Text = ex.Message;
                }
            }

//-----------------------------------------------------------------------------
//                      Поиск классов для связи
//                       агрегации и композиции
//-----------------------------------------------------------------------------

            private void classSearchAggregationAndComposition(int x1, int x2, int
y1, int y2, string m, string r)
            {
                foreach (ClassInfo class_info in List_Class)
                {
                    //аналогично как при наследовании
                    if (class_info.x <= x2 &&
                        x2 <= (class_info.x + class_info.w) &&
                        class_info.y <= y2 &&
                        y2 <= (class_info.y + class_info.h))
                    {
                        foreach (ClassInfo class_info_2 in List_Class)
                        {
                            if (class_info_2.x <= x1 &&
                              x1 <= (class_info_2.x + class_info_2.w) &&
                              class_info_2.y <= y1 &&
                              y1 <= (class_info_2.y + class_info_2.h))
                            {
                                //добавляем информацию о связи в класс
                                ClassInfo.AggregationAndCompositionInfo
AggregationAndComposition = new ClassInfo.AggregationAndCompositionInfo();
                                AggregationAndComposition.Parse(class_info.name, r,
m);
                                class_info_2.riCA.Add(new
ClassInfo.AggregationAndCompositionInfo()
                                {
                                    resClass = AggregationAndComposition.resClass,
                                    acs = AggregationAndComposition.acs,
                                    text = AggregationAndComposition.text,
                                    multiplicity =
AggregationAndComposition.multiplicity,
                                    knd = AggregationAndComposition.knd
                                });
                            }
                        }
                    }
                }
            }
//-----------------------------------------------------------------------------
//                      Поиск классов для связи наследования
//-----------------------------------------------------------------------------
```

```
            private void classSearchGeneralization(int x1, int x2, int y1, int y2,
string text)
            {
                foreach (ClassInfo class_info in List_Class)
                {
                    if (class_info.x <= x1 &&
                        x1 <= (class_info.x + class_info.w) &&
                        class_info.y <= y1 &&
                        y1 <= (class_info.y + class_info.h))
                    {
                        foreach (ClassInfo class_info_2 in List_Class)
                        {
                            if (class_info_2.x <= x2 &&
                              x2 <= (class_info_2.x + class_info_2.w) &&
                              class_info_2.y <= y2 &&
                              y2 <= (class_info_2.y + class_info_2.h))
                            {
                                //добавляем информацию о наследовании в класс
                                ClassInfo.GeneralizationInfo  generalization  =  new
ClassInfo.GeneralizationInfo();
                                generalization.Parse(class_info.name, text);
                                class_info_2.riG.Add(new
ClassInfo.GeneralizationInfo()
                                {
                                    resClass = generalization.resClass,
                                    acs = generalization.acs
                                });
                            }
                        }
                    }
                }
            }

//-------------------------------------------------------------------------------
//          Чтение из файла элементов Relation, добавление их в список
//-------------------------------------------------------------------------------


            private void ReadXmlRelation()
            {
                try
                {
                    XmlDocument xDoc = new XmlDocument();
                    xDoc.Load(textBox2.Text);
                    // получим корневой элемент
                    XmlElement xRoot = xDoc.DocumentElement;
                    // обход всех узлов в корневом элементе

                    foreach (XmlNode xnode in xRoot)
                    {
                        string str_id = "";
                        RelationInfo relation_info = new RelationInfo();
                        // обходим все дочерние узлы элемента element
                        foreach (XmlNode childnode in xnode.ChildNodes)
                        {
                            if (childnode.Name == "id")
                                str_id = childnode.InnerXml;
                            if (str_id == "Relation")
                            {
                                if (childnode.Name == "coordinates")
                                {
```

```
                                    foreach (XmlNode child_coordinates in
childnode.ChildNodes)
                                    {
                                        if (child_coordinates.Name == "x")
                                            relation_info.x =
Convert.ToInt32(child_coordinates.InnerXml);
                                        if (child_coordinates.Name == "y")
                                            relation_info.y =
Convert.ToInt32(child_coordinates.InnerXml);
                                        if (child_coordinates.Name == "w")
                                            relation_info.w =
Convert.ToInt32(child_coordinates.InnerXml);
                                        if (child_coordinates.Name == "h")
                                            relation_info.h =
Convert.ToInt32(child_coordinates.InnerXml);
                                    }
                                }
                                if (childnode.Name == "panel_attributes")
                                {

relation_info.ParsePanelAttributes(childnode.InnerXml);
                                    //если связь = ассоциация/зависимость, то на
коде никак не влияет
                                    if   (Convert.ToString(relation_info.name)   ==
"Association" || Convert.ToString(relation_info.name) == "Dependency") break;
                                }
                                if (childnode.Name == "additional_attributes")
                                {

relation_info.ParseAdditionalAttributes(childnode.InnerXml);
                                    int x1 = relation_info.x1;
                                    int x2 = relation_info.x2;
                                    int y1 = relation_info.y1;
                                    int y2 = relation_info.y2;
                                    //если связь = наследование,
                                    //то сначала находим класс от которого
                                    //идёт связь
                                    //после ищем класс
                                    //в котором будет эта связь добавлена
                                    if   (Convert.ToString(relation_info.name)   ==
"Generalization")
                                    {
                                        string text = relation_info.text;
                                        classSearchGeneralization(x1,  x2,  y1,  y2,
relation_info.text);
                                    }
                                    else
                                    {
                                    //если не наследование,
                                    //то значит это композиция или агрегация
                                    //если связь не несет информацию,
                                    //то она не влияет на код
                                        if (relation_info.r1 != null)
                                        {
                                          //множественность
                                            string m1 = relation_info.m1;
                                          //роль
                                            string r1 = relation_info.r1;

classSearchAggregationAndComposition(x2, x1, y2, y1, m1, r1);
                                        }
```

```
                                        if (relation_info.r2 != null)
                                        {
                                          //множественность
                                            string m2 = relation_info.m2;
                                          //роль
                                            string r2 = relation_info.r2;

classSearchAggregationAndComposition(x1, x2, y1, y2, m2, r2);
                                        }

                                    }
                                    //Добавляем всю информацию в список связей
                                    List_Relation.Add(new RelationInfo()
                                    {
                                        x = relation_info.x,
                                        y = relation_info.y,
                                        w = relation_info.w,
                                        h = relation_info.h,
                                        name = relation_info.name,
                                        x1 = relation_info.x1,
                                        x2 = relation_info.x2,
                                        y1 = relation_info.y1,
                                        y2 = relation_info.y2,
                                        a = relation_info.a,
                                        b = relation_info.b,
                                        c = relation_info.c,
                                        d = relation_info.d
                                    });
                                }
                            }
                        }
                    }
                    catch (Exception ex)
                    {
                        toolStripStatusLabel1.Text = ex.Message;
                    }
            }

//-----------------------------------------------------------------------------
//                            Кнопка "Генерация кода"
//-----------------------------------------------------------------------------


            private void button2_Click(object sender, EventArgs e)
            {
                try
                {
                        string sfilename=""; //путь к файлу
                        saveFileDialog1.Filter = "Заголовочные файлы (*.h)|*.h|All
files (*.*)|*.*";
                        if (saveFileDialog1.ShowDialog() == DialogResult.OK)
                        {
                        //имя файла, в который будет записан результат
                            sfilename = saveFileDialog1.FileName;
                        }
                        if (sfilename != "") //если файл выбрана
                        {
                            List_Class.Clear();
                            List_Relation.Clear();

                        //чтение классов
```

```
                        ReadXmlClass();
                    //чтение взаимосвязей между классами
                        ReadXmlRelation();

                    //создание файла
                        StreamWriter sw = new StreamWriter(sfilename);
                        sw.WriteLine("#ifndef _CLASS_H");
                        sw.WriteLine("#define _CLASS_H");
                        sw.WriteLine();
                        sw.WriteLine();
                        foreach (ClassInfo class_info in List_Class)
                        {
                          //генерируем код для всех элементов класса
                            class_info.Generate(sw);
                        }
                        sw.WriteLine("#endif");
                        sw.Close();
                        toolStripStatusLabel1.Text = "Готово!";
                        button2.Enabled = false;
                        sfilename = "";
                    }

                else toolStripStatusLabel1.Text = "Файл не выбран.";
            }
            catch (Exception ex)
            {
                toolStripStatusLabel1.Text = ex.Message;
            }
        }
    }
}
```

ClassInfo.cs – содержит информацию о классе.

```
//---------------------------------------------------------------------------
//                      ClassInfo.cs
//---------------------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;

   namespace WindowsFormsApplication1
   {
       class ClassInfo
       {
           public int x, y, w, h;
           public string name;
           public enum Access { Public, Protected, Private }
           public enum Kind { Abstract, Static, Null }

//---------------------------------------------------------------------------
//                      Внутренний класс FieldInfo
//---------------------------------------------------------------------------

           public class FieldInfo
           {
```

```
                public Access acs;
                public Kind knd;
                public string name;
                public string type;
                public string defaultValue;

//--------------------------------------------------------------------------
//                      метод класса FieldInfo
//--------------------------------------------------------------------------

                private Access getacs(char s,ref int refindex)
                {
                    if (s == '-')
                    {
                        refindex++;
                        return Access.Private;
                    }
                    else
                    {
                        if (s == '+')
                        {
                            refindex++;
                            return Access.Public;
                        }
                        else
                        {
                            if (s == '#')
                            {
                                refindex++;
                                return Access.Protected;
                            }
                            else return Access.Private;
                        }
                    }
                }

//--------------------------------------------------------------------------
//                      метод класса FieldInfo
//--------------------------------------------------------------------------

                public void Parse(string s)
                {
                    int index = 0;
                    string str = "";

                  //из входной строчки удаляем все пробелы
                    s = s.Replace(" ", "");
                  //длина строчки
                    int len = s.Length;

                    //если первый символ строки "_", вид = статик, а иначе не имеет
                    if (s[0] == '_')
                        {
                            this.knd = Kind.Static;
                            index++;
                            len--;
                        }
                        else this.knd = Kind.Null;

                    //определеяем тип доступа
                    this.acs = getacs(s[index],ref index);
```

```
                    //считывает имя переменной
                    for (int i = index; s[i]!=':'; i++)
                    {
                        str += s[i];
                        index++;
                    }

                    //записывам имя
                    this.name = str;
                    str = "";

                    //считываем тип переменной
                    for (int i = index + 1; i != len && s[i]!='='; i++)
                    {
                        str += s[i];
                        index++;
                    }

                    //записываем
                    this.type = str;
                    str = "";

                    //считываем начальное значение
                    for (int i = index + 1; i != len ; i++)
                    {
                        str += s[i];
                        index++;
                    }
                    this.defaultValue = str;

                }

//-------------------------------------------------------------------------
//                      метод класса FieldInfo
//-------------------------------------------------------------------------

                public void Generate(StreamWriter sw) {
                    string s;
                    if (this.knd == Kind.Static) s = "static ";
                    else s = "";
                    s = "\t"+ s +this.type + " " +this.name+this.defaultValue+ ";";
                    sw.WriteLine(s);
                }
            }

//-------------------------------------------------------------------------
//                      Внутренний класс MethodInfo
//-------------------------------------------------------------------------

            public List<FieldInfo> fi = new List<FieldInfo>();
            public class MethodInfo
            {
                public Access acs;
                public Kind knd;
                public string text;
                public string method;
                public string type;

//-------------------------------------------------------------------------
//                      метод класса MethodInfo
//-------------------------------------------------------------------------
```

```
                private Access getacs(char s, ref int refindex)
                {
                    if (s == '-')
                    {
                        refindex++;
                        return Access.Private;
                    }
                    else
                    {
                        if (s == '+')
                        {
                            refindex++;
                            return Access.Public;
                        }
                        else
                        {
                            if (s == '#')
                            {
                                refindex++;
                                return Access.Protected;
                            }
                            else return Access.Private;
                        }
                    }
                }

//-------------------------------------------------------------------------
//                      метод класса MethodInfo
//-------------------------------------------------------------------------

                private Kind getaknd(char s, ref int refindex, ref int reflen)
                {
                    if (s == '/')
                    {
                        refindex++;
                        reflen--;
                        return Kind.Abstract;
                    }
                    else if (s == '_')
                    {
                        refindex++;
                        reflen--;
                        return Kind.Static;
                    }else return Kind.Null;
                }

//-------------------------------------------------------------------------
//                      метод класса MethodInfo
//-------------------------------------------------------------------------


                public void Parse(string s, string nameclass)
                {
                    int index = 0;
                    string str = "";
                    string namemethod="";

                    s = s.Replace(" ", "");
                    int len;
                    len = s.Length;

                    this.knd = getaknd(s[0], ref index, ref len);
```

```
                this.acs = getacs(s[index],ref index);
                //считываем название метода
                for (int i = index; s[i] != '('; i++)
                {
                    namemethod += s[i];
                    index++;
                }

                this.method = namemethod;
                //считываем переменные метода
                for (int i = index + 1; s[i] != ')'; i++)
                {
                    str += s[i];
                    index++;
                }
                //приводим переменные к нормальному виду
                if (str != "")
                {
                    string[] split = str.Split(new Char[] { ',', ':' });
                    str = "";

                    for (int i = 0; i < split.Count(); i = i + 2)
                        if (i == 0) str += split[i + 1] + " " + split[i] + ",";
                        else str += " " + split[i + 1] + " " + split[i] + ",";
                }
               //удаляем последнюю запятую,
              //где происходит перечисление переменных
                this.text = str.TrimEnd(',');
                str = "";

                for (int i = index + 2; i != len; i++)
                {
                        if(s[i]!=':') str += s[i];
                }
                if (str == "")
                {
                    if(namemethod==nameclass|| ("~"+nameclass == namemethod))
                        this.type = "";
                    else this.type = "void";
                }
                else this.type = str;
            }

//-------------------------------------------------------------------------
//                    метод класса MethodInfo
//-------------------------------------------------------------------------

            public void Generate(StreamWriter sw)
            {
                string s;
                if (this.knd == Kind.Static)
                {
                    s = "\t"+"static "+ this.type + " "
                          + this.method + "(" + this.text + ")" + ";";
                }else
                    if (this.knd == Kind.Abstract)
                    {
                        s = "\t" + "virtual " + this.type + " "
                          + this.method + "(" + this.text + ") = 0" + ";";
                    } else s = "\t" + this.type + " " + this.method
                          + "(" + this.text + ")" + ";";
                sw.WriteLine(s);
```

```
            }
        }
        public List<MethodInfo> mi = new List<MethodInfo>();

//-----------------------------------------------------------------------------
//                      Внутренний класс GeneralizationInfo
//-----------------------------------------------------------------------------


        public List<GeneralizationInfo> riG = new List<GeneralizationInfo>();
        public class GeneralizationInfo
        {
            public string resClass;
            public string acs;

//-----------------------------------------------------------------------------
//                      метод класса GeneralizationInfo
//-----------------------------------------------------------------------------

            public void Parse(string nameclass,string text)
            {
                //nameclass - имя класса, от которого идёт связь
                this.resClass = nameclass;
                //если идентификатор доступа не указан, то по умолчанию public
                if (text == null || text == "")
                {
                    this.acs = "public";
                }
                else
                {
                        if (text == "&lt;&lt;private&gt;&gt;")
                          this.acs = "private";
                        if (text == "&lt;&lt;public&gt;&gt;")
                          this.acs = "public";
                        if (text == "&lt;&lt;protected&gt;&gt;")
                          this.acs = "protected";
                }
            }
//-----------------------------------------------------------------------------
//                      Метод класса GeneralizationInfo
//-----------------------------------------------------------------------------

            public void Generate(StreamWriter sw)
            {
                string s;
                s = this.acs + " " + this.resClass;
                sw.Write(s);
            }
        }

//-----------------------------------------------------------------------------
//                Внутренний класс AggregationAndCompositionInfo
//-----------------------------------------------------------------------------

        public List<AggregationAndCompositionInfo> riCA =
                          new List<AggregationAndCompositionInfo>();
        public class AggregationAndCompositionInfo
        {
            public string resClass;
            public string text;
            public Access acs;
            public string knd;
```

```
                public string multiplicity;

//-----------------------------------------------------------------------------
//                Метод класса AggregationAndCompositionInfo
//-----------------------------------------------------------------------------

                public void Parse(string classname, string r_text, string m)
                {
                    this.resClass = classname;
                    if (r_text != null)
                    {
                        string str = r_text;
                        r_text = r_text.Replace("_", "");
                        if (str != r_text)
                            this.knd = "static ";
                        else this.knd = "";

                        //из входной строчки удаляем все пробелы
                        r_text = r_text.Replace(" ", "");
                        if (r_text[0] == '-')
                        {
                            this.acs = Access.Private;
                            this.text = r_text.Remove(0, 1);
                        }
                        else
                        {
                            if (r_text[0] == '+')
                            {
                                this.acs = Access.Public;
                                this.text = r_text.Remove(0, 1);
                            }
                            else
                            {
                                if (r_text[0] == '#')
                                {
                                    this.acs = Access.Protected;
                                    this.text = r_text.Remove(0, 1);
                                }
                                else
                                {
                                    this.acs = Access.Private;
                                    this.text = r_text;
                                }
                            }
                        }
                    }
                    if (m != null)
                    {
                        //из входной строчки удаляем все пробелы
                        m = m.Replace(" ", "");
                        if (m == "0..1" || m == "1")
                        {
                            this.multiplicity = classname + " ";
                        }
                        if (m == "0..*" || m == "1..*" || m == "*")
                        {
                            this.multiplicity = "vector<"+classname+"> ";
                        }
                    }
                    else this.multiplicity = classname + " ";
                }
//-----------------------------------------------------------------------------
```

```
//                Метод класса AggregationAndCompositionInfo
//-----------------------------------------------------------------------------


                public void Generate(StreamWriter sw)
                {
                    string s;
                    s = "\t"+ this.knd + this.multiplicity + this.text+";";
                    sw.WriteLine(s);
                }
            }

//-----------------------------------------------------------------------------
//                Метод класса ClassInfo
//-----------------------------------------------------------------------------


            public void Parse(string  s) {
                String pattern = @"--";
                String nameclass;

                String[] elements = Regex.Split(s, pattern);

             //удаляем начальные и конечные пробелы
                nameclass = elements[0].Trim();
               nameclass = nameclass.Replace("/", "");
                this.name = nameclass;

                if(elements.ElementAtOrDefault(1) != null){
                    FieldInfo f = new FieldInfo();
                    String p = @"\r\n";
                    String[] fields = Regex.Split(elements[1], p);

                    foreach (string str in fields)
                    {
                        if (str != "")
                        {
                            f.Parse(str);
                            fi.Add(new FieldInfo() {
                                acs = f.acs,
                                knd = f.knd,
                                name = f.name,
                                type = f.type,
                                defaultValue = f.defaultValue
                            });
                        }
                    }
                }

                if (elements.ElementAtOrDefault(2) != null)
                {
                    MethodInfo m = new MethodInfo();
                    String p = @"\r\n";
                    String[] methods = Regex.Split(elements[2], p);

                    foreach (string str in methods)
                    {
                        if (str != "")
                        {
                            m.Parse(str, nameclass);
                            mi.Add(new MethodInfo()
                            {
```

```
                                acs = m.acs,
                                knd = m.knd,
                                text = m.text,
                                method = m.method,
                                type = m.type
                            });
                        }
                    }
                }
            }

//-----------------------------------------------------------------------------
//                      Метод класса ClassInfo
//-----------------------------------------------------------------------------

            public void GenerateForFieldsAndMethods(Access a,StreamWriter sw)
            {
                foreach(FieldInfo f in fi)
                {
                    if(f.acs==a)
                        f.Generate(sw);
                }
                foreach (AggregationAndCompositionInfo r in riCA)
                {
                    if (r.acs == a)
                        r.Generate(sw);
                }
                foreach (MethodInfo f in mi)
                {
                    if (f.acs == a)
                        f.Generate(sw);
                }
            }

//-----------------------------------------------------------------------------
//                      Метод класса ClassInfo
//-----------------------------------------------------------------------------

            public void GenerateForGeneralization( StreamWriter sw)
            {
                if(riG.Count()!=0)
                {
                    sw.Write(": ");
                    int count = 0; //количество сколько уже записали
                    int countrig=riG.Count(); //количество наследований
                    foreach (GeneralizationInfo r in riG)
                    {
                            r.Generate(sw);
                            count++;
                            if (count != countrig) sw.Write(", ");
                    }
                }
            }

//-----------------------------------------------------------------------------
//                      Метод класса ClassInfo
//-----------------------------------------------------------------------------

            public void Generate(StreamWriter sw)
            {
                sw.Write("class ");
                sw.Write(name);
```

```
                GenerateForGeneralization(sw);
                sw.WriteLine("{");
                sw.WriteLine("private:");
                GenerateForFieldsAndMethods(Access.Private, sw);
                sw.WriteLine("public:");
                GenerateForFieldsAndMethods(Access.Public, sw);
                sw.WriteLine("protected:");
                GenerateForFieldsAndMethods(Access.Protected, sw);
                sw.WriteLine("};");
                sw.WriteLine();
                sw.WriteLine();
            }
        }
    }
```

RelationInfo.cs – класс, который содержит информацию о связи.

```
//-------------------------------------------------------------------------
//                      RelationInfo.cs
//-------------------------------------------------------------------------

    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using System.Text.RegularExpressions;
    using System.Threading.Tasks;

    namespace WindowsFormsApplication1
    {
        class RelationInfo
        {
            public int x, y, w, h;
            public int a, b, c, d;
            public int x1, x2,y1, y2;
            public string text;
            public string m1;
            public string m2;
            public string r1;
            public string r2;
            public string r;
            public Kind name;
            public enum Kind { Association, Dependency, Aggregation, Composition,
Generalization };

//-------------------------------------------------------------------------
//                      метод ParseTextRelation
//-------------------------------------------------------------------------

            private void ParseTextRelation(string s)
            {
                int n = s.Length; // длина строки до
                s = s.Replace("m1=", "");
                int m = s.Length; //длина строки после

             //если после удаления элементов ничего не изменилось,
             //то нет m1 или m2
                if (m != n) this.m1 = s;
                else
                {
```

```
                    s = s.Replace("m2=", "");
                    m = s.Length;

                    if (m != n) this.m2 = s;
                    else
                    {
                        s = s.Replace("r1=", "");
                        m = s.Length;

                        if (m != n) this.r1 = s;
                        else
                        {
                            s = s.Replace("r2=", "");
                            m = s.Length;

                            if (m != n) this.r2 = s;
                            else this.text = s;
                        }

                    }
                }
            }

//-------------------------------------------------------------------------------
//                      метод ParsePanelAttributes
//-------------------------------------------------------------------------------

            public void ParsePanelAttributes(string s)
            {
                string pattern = @"\r\n";

                //Содержимое panel_attributes делим на
                //элементы по разделителю "перенос строки"
                string[] elements = Regex.Split(s, pattern);

                //первый элемент содержит информацию о виде связи
                if (elements[0].Trim() == "lt=&lt;-" ||
                  elements[0].Trim() == "lt=-")
                  this.name = Kind.Association;

                if (elements[0].Trim() == "lt=&lt;.")
                  this.name = Kind.Dependency;

                if (elements[0].Trim() == "lt=&lt;&lt;&lt;&lt;-")
                  this.name = Kind.Aggregation;

                if (elements[0].Trim() == "lt=&lt;&lt;&lt;&lt;&lt;-")
                  this.name = Kind.Composition;

                if (elements[0].Trim() == "lt=&lt;&lt;-")
                  this.name = Kind.Generalization;

                //остальные элементы содержат, что несет в себе связь
                if (this.name == Kind.Generalization)
                {
                     if (elements.ElementAtOrDefault(1) != null)
                         this.text = elements[1];
                }

                if (this.name == Kind.Composition || this.name == Kind.Aggregation)
                {
                    if (elements.ElementAtOrDefault(1) != null)
```

```
                    ParseTextRelation(elements[1]);
                if (elements.ElementAtOrDefault(2) != null)
                    ParseTextRelation(elements[2]);
                if (elements.ElementAtOrDefault(3) != null)
                    ParseTextRelation(elements[3]);
                if (elements.ElementAtOrDefault(4) != null)
                    ParseTextRelation(elements[4]);
                if (elements.ElementAtOrDefault(5) != null)
                    ParseTextRelation(elements[5]);
            }
        }

//-------------------------------------------------------------------------
//                    метод ParseAdditionalAttributes
//-------------------------------------------------------------------------

        public void ParseAdditionalAttributes(string s)
        {
            //разделяем additional_attributes на элементы по разделителю ";"
            string pattern = @";";
            string[] elements = Regex.Split(s, pattern);
            int count = elements.Count(); //количество элементов

            double a1 = Convert.ToDouble(elements[0],
                                         CultureInfo.GetCultureInfo("en-US"));
            double b1 = Convert.ToDouble(elements[1],
                                         CultureInfo.GetCultureInfo("en-US"));
            double c1 = Convert.ToDouble(elements[count - 2],
                                         CultureInfo.GetCultureInfo("en-US"));
            double d1 = Convert.ToDouble(elements[count - 1],
                                         CultureInfo.GetCultureInfo("en-US"));

            this.a = Convert.ToInt32(a1);
            this.b = Convert.ToInt32(b1);
            this.c = Convert.ToInt32(c1);
            this.d = Convert.ToInt32(d1);

            this.x1 = this.x + this.a;
            this.y1 = this.y + this.b;
            this.x2 = this.x + this.c;
            this.y2 = this.y + this.d;
        }
    }
}
```