МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий
Кафедра вычислительной техники

КУРСОВАЯ РАБОТА

По дисциплине «Объектно-ориентированное программирование»
(наименование дисциплины)

Тема курсовой работы Моделирование работы автоматического склада
(наименование темы)

Студент группы ИКБО-02-21 Хитров Никита Сергеевич
(учебная группа) *(Фамилия Имя Отчество)* *(подпись студента)*

Руководитель курсовой работы доцент каф.ВТ Путуридзе З.Ш.
(Должность, звание, ученая степень) *(подпись руководителя)*

Консультант ст.пр.каф.ВТ Данилович Е.С.
(Должность, звание, ученая степень) *(подпись консультанта)*

4(хор)

Работа представлена к защите « 7 » июня 2022 г.

Допущен к защите « 7 » мая 2022 г.

Москва 2022 г.

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий
Кафедра вычислительной техники

Утверждаю

Заведующий кафедрой _____
Подпись

Платонова О.В.
ФИО

« 14 » марта 2022г.

ЗАДАНИЕ
На выполнение курсовой работы
по дисциплине **«Объектно-ориентированное программирование»**

Студент Хитров Никита Сергеевич Группа ИКБО-02-21

Тема Моделирование работы автоматического склада

Исходные данные:

1. Описания исходной иерархии дерева объектов.
2. Описание схемы взаимодействия объектов.
3. Множество команд для управления функционированием моделируемой системы.

Перечень вопросов, подлежащих разработке, и обязательного графического материала:

1. Построение версий программ.
2. Построение и работа с деревом иерархии объектов.
3. Взаимодействия объектов посредством интерфейса сигналов и обработчиков.
4. Блок-схемы алгоритмов.
5. Управление функционированием моделируемой системы

Срок представления к защите курсовой работы: до « 16 » мая 2022 г.

Задание на курсовую работу выдал _____ (Данилович Е.С.)
Подпись *ФИО консультанта*
« 28 » февраля 2022 г.

Задание на курсовую работу получил _____ (Хитров Н.С)
Подпись *ФИО исполнителя*
« 28 » февраля 2022 г.

Москва 2022г.

ОТЗЫВ

на курсовую работу

по дисциплине «Объектно-ориентированное программирование»

Студент **Хитров Никита Сергеевич** группа ИКБО-02-21
(ФИО студента) *(Группа)*

Характеристика курсовой работы

Критерий	Да	Нет	Не полностью
1. Соответствие содержания курсовой работы указанной теме	✓		
2. Соответствие курсовой работы заданию	✓		
3. Соответствие рекомендациям по оформлению текста, таблиц, рисунков и пр.	✓		
4. Полнота выполнения всех пунктов задания		✓	
5. Логичность и системность содержания курсовой работы	✓		
6. Отсутствие фактических грубых ошибок	✓		

Замечаний: Не все пункты выполнены

Рекомендуемая оценка: 4 (хор)

доцент каф.ВТ Путуридзе З.Ш.
(Подпись руководителя) *(ФИО руководителя)*

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В заключительной части курсовой работы необходимо использовать знания, полученные за прохождение курса Объектно-ориентированного программирования, а именно:
взаимодействие системы объектов при помощи сигналов и обработчиков[5];построение дерева иерархии объектов;
проектирование системы.

А также работа с техническим заданием, грамотное составленик технической документации (а именно: написание метода решения, описание алгоритмов и оформление блок-схем согласно требованию приложению к методическому пособию [4]).

Основываясь на методе решения, алгоритме и блок-схемах реализовать поставленную задачу, используя следующие парадигмы объектно-ориентированного программирования:

1. Инкапсуляция - механизм, который объединяет данные и код, манипулирующий с этими данными, а также защищает и то, и другое от внешнего вмешательства или неправильного использования [2, с. 11];
2. Наследование - механизм, который позволяет одним классам включать в себя поля и методы других классов [1, с. 106].

1 ПОСТАНОВКА ЗАДАЧИ

Реализация сигналов и обработчиков

Для организации взаимодействия объектов вне схемы взаимосвязи используется механизм сигналов и обработчиков. Вместе с передачей сигнала еще передается определенное множество данных. Механизм сигналов и обработчиков реализует схему взаимодействия объектов один ко многим.

Реализовать механизм взаимодействия объектов с использованием сигналов и обработчиков, с передачей вместе сигналом текстового сообщения (строковой переменной).

Для организации взаимосвязи по механизму сигналов и обработчиков в базовый класс добавить три метода:

1. Установления связи между сигналом текущего объекта и обработчиком целевого объекта;

2. Удаления (разрыва) связи между сигналом текущего объекта и обработчиком целевого объекта;

3. Выдачи сигнала от текущего объекта с передачей строковой переменной. Включенный объект может выдать или обработать сигнал.

Методу установки связи передать указатель на метод сигнала текущего объекта, указатель на целевой объект и указатель на метод обработчика целевого объекта.

Методу удаления (разрыва) связи передать указатель на метод сигнала текущего объекта, указатель на целевой объект и указатель на метод обработчика целевого объекта.

Методу выдачи сигнала передать указатель на метод сигнала и строковую

переменную. В данном методе реализовать алгоритм:

1. Вызов метода сигнала с передачей строковой переменной по ссылке.

2. Цикл по всем связям сигнал-обработчик текущего объекта.

2.1. Если в очередной связи сигнал-обработчик участвует метод сигнала, переданный по параметру, то вызвать метод обработчика очередного целевого объекта и передать в качестве аргумента строковую переменную по значению.

3. Конец цикла.

Для приведения указателя на метод сигнала и на метод обработчика использовать параметризированное макроопределение препроцессора.

В базовый класс добавить метод определения абсолютного пути до текущего объекта. Этот метод возвращает абсолютный путь текущего объекта.

Состав и иерархия объектов строится посредством ввода исходных данных. Ввод организован как в версии № 3 курсовой работы.

Система содержит объекты шести классов с номерами: 1,2,3,4,5,6. Классу корневого объекта соответствует номер 1. В каждом производном классе реализовать один метод сигнала и один метод обработчика.

Каждый метод сигнала с новой строки выводит:

Signal from «абсолютная координата объекта»

Каждый метод сигнала добавляет переданной по параметру строке текста номер класса принадлежности текущего объекта по форме:

«пробел»(class: «номер класса»)

Каждый метод обработчика с новой строки выводит:

Signal to «абсолютная координата объекта» Text: «переданная строка»

Реализовать алгоритм работы системы:

1. В методе построения системы:

1.1. Построение дерева иерархии объектов согласно вводу.

1.2. Ввод и построение множества связей сигнал-обработчик для заданных пар объектов.

2. В методе отработки системы:

2.1. Привести все объекты в состоянии готовности.

2.2. Цикл до признака завершения ввода.

2.2.1. Ввод наименования объекта и текста сообщения.

2.2.2. Вызов сигнала заданного объекта и передача в качестве аргумента строковой переменной, содержащей текст сообщения.

2.3. Конец цикла.

Допускаем, что все входные данные вводятся синтаксически корректно. Контроль корректности входных данных можно реализовать для самоконтроля работы программы.

Не оговоренные, но необходимые функции и элементы классов добавляются разработчиком.

1.1 Описание входных данных

В методе построения системы.

Множество объектов, их характеристики и расположение на дереве иерархии. Структура данных для ввода согласно изложенному в версии № 3 курсовой работы.

После ввода состава дерева иерархии построчно вводится:

«координата объекта выдающего сигнал» «координата целевого объекта»

Ввод информации для построения связей завершается строкой, которая

содержит

end_of_connections

В методе запуска (отработки) системы.

Построчно вводятся множество команд в производном порядке:

EMIT «координата объекта» «текст» - выдать сигнал от заданного по координате объекта;

SET_CONNECT «координата объекта выдающего сигнал» «координата целевого объекта» - установка связи;

DELETE_CONNECT «координата объекта выдающего сигнал» «координата целевого объекта» - удаление связи;

SET_CONDITION «координата объекта» «значение состояния» - установка состояния объекта.

END – завершить функционирование системы (выполнение программы).

Команда END присутствует обязательно.

Если координата объекта задана некорректно, то соответствующая операция не выполняется и с новой строки выдается сообщение об ошибке.

Если не найден объект по координате:

Object «координата объекта» not found

Если не найден целевой объект по координате:

Handler object «координата целевого объекта» not found

Пример ввода

appls_root

/ object_s1 3

/ object_s2 2

/object_s2 object_s4 4

/ object_s13 5

/object_s2 object_s6 6

/object_s1 object_s7 2

endtree

/object_s2/object_s4 /object_s2/object_s6

/object_s2 /object_s1/object_s7

/ /object_s2/object_s4

/object_s2/object_s4 /

end_of_connections

EMIT /object_s2/object_s4 Send message 1

DELETE_CONNECT /object_s2/object_s4 /

EMIT /object_s2/object_s4 Send message 2

SET_CONDITION /object_s2/object_s4 0

EMIT /object_s2/object_s4 Send message 3

SET_CONNECT /object_s1 /object_s2/object_s6

EMIT /object_s1 Send message 4

END

1.2 Описание выходных данных

Первая строка:

Object tree

Со второй строки вывести иерархию построенного дерева.

Далее, построчно, если отработал метод сигнала:

Signal from «абсолютная координата объекта»

Если отработал метод обработчика:

Signal to «абсолютная координата объекта» Text: «переданная строка»

Пример вывода

Object tree

appls_root

    object_s1

        object_s7

    object_s2

        object_s4

        object_s6

    object_s13

Signal from /object_s2/object_s4

Signal to /object_s2/object_s6 Text: Send message 1 (class: 4)

Signal to / Text: Send message 1 (class: 4)

Signal from /object_s2/object_s4

Signal to /object_s2/object_s6 Text: Send message 2 (class: 4)

Signal from /object_s1

Signal to /object_s2/object_s6 Text: Send message 4 (class: 3)

2 МЕТОД РЕШЕНИЯ

Объекты потокового ввода и вывода cin и cout соответственно.

Оператор создания псевдонимов существующих типов данных typedef.

Тип данных TYPE_SIGNAL - тип метода сигнал объекта.

Тип данных TYPE_HANDLER - тип метода обработчика сигнала.

Параметризованные макроопределения препроцессора SIGNAL_D и HANDLER_D - необходим для получения ссылки на методы сигнала и обработчика.

Функция для ввода данных из потока до строкового разделителя - getline.

Метод проверки строки на отсутствие символов - empty.

Метод проверки контейнера на отсутствие элементов - empty.

Объект AppObj класса App.

Объекты классов Second, Third, Fourth, Fifth, Sixth, кол-во которых определяется пользовательским вводом.

Класс Base:
Поля:

Поле структуры связи:

1. Наименование - connectionStruct;
2. Тип - структура;
3. Содержит - signalMethod (метод сигнала, типTYPE_SIGNAL), handler (указатель на объект-обработчик класса Base), handlerMethod(метод обработчика, тип TYPE_HANDLER);
4. Модификатор доступа - private.

Поле хранения всех связей:

1. Наименование – connections;

2. Тип - контейнер целочисленных значений vector;
3. Модификатор доступа - protected.

Функционал:

1. Метод setConnection - установление связи между сигналом текущего объекта и обработчкиком сигналов целевого объекта;
2. Метод deleteConnection - удаление связи между сигналом текущего объекта и обработчкиком сигналов целевого объекта;
3. Метод emitSignal - выдача сигнала от текущего объекта;
4. Метод getAbsoluteCoordinate - определение абсолютного пути дотекущего объекта;
5. Метод getClassNumber - получение номера класса текущего объекта;
6. Метод SetReadyForAll – приведение всех объектов иерархии всостояние готовности.

Класс App:

Функционал:

1. Метод signalMethod - выдача сигнала;
2. Метод handlerMethod - обработка сигнала
3. Метод getSignalMethod - получение ссылки на метод сигнала нужного класса; Метод getHandlerMethod - получение ссылки на метод обработки нужного класса; Метод buildTree - постороение дерева иерархии объектов согласнопользовательскому вводу, ввод и построение множества связей "сигнал-обработчик" для заданных пар объектов;
4. Метод execute - запуск приложения (вывод дерева иерархии, обработка команд).

Класс Second:

Функционал:

1. Метод signalMethod - выдача сигнала;
2. Метод handlerMethod - обработка сигнала;
3. Конструктор - вызов конструктора класса Base, присвоение номера класса.

Класс Third:

Функционал:

1. Метод signalMethod - выдача сигнала;
2. Метод handlerMethod - обработка сигнала;
3. Конструктор - вызов конструктора класса Base, присвоение номера класса.

Класс Fourth:

Функционал:

1. Метод signalMethod - выдача сигнала;
2. Метод handlerMethod - обработка сигнала;
3. Конструктор - вызов конструктора класса Base, присвоение номера класса.

Класс Fifth:

Функционал:

1. Метод signalMethod - выдача сигнала;

2. Метод handlerMethod - обработка
   сигнала;
3. Конструктор - вызов конструктора класса Base, присвоение номера класса.

Класс Sixth:

Функционал:

1. Метод signalMethod - выдача
   сигнала;
2. Метод handlerMethod - обработка
   сигнала;
3. Конструктор - вызов конструктора класса Base, присвоение номера класса.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер	Комментарий
1	Base			Базовый класс иерархии объектов - содержит основные поля и методы		
		App	public		2	
		Secomd	public		3	
		Third	public		4	
		Fourth	public		5	
		Fifth	public		6	

		Sixth	public		7	
2	App			Класс приложения - строит дерево объектов, запускает приложение и обеспечивает его функционирование		
3	Second			Класс объектов, подчиненных базовому классу		
4	Third			Класс объектов, подчиненных базовому классу		
5	Fourth			Класс объектов, подчиненных базовому классу		

6	Fifth			Класс объектов, подчиненных базовому классу		
7	Sixth			Класс объектов, подчиненных базовому классу		

3 ОПИСАНИЕ АЛГОРИТМОВ

Согласно этапам разработки, после определения необходимого инструментария в разделе «Метод», составляются подробные описания алгоритмов для методов классов и функций.

3.1 Алгоритм конструктора класса Second

Функционал: Вызывает конструктор класса Base, присваивает номер класса.

Параметры: Base* parent - указатель на родительский объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 2.

Таблица 2 – Алгоритм конструктора класса Second

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса Base с параметрами parent, name	2
2		Присвоение полю classNumber значения 2	∅

3.2 Алгоритм метода signalMethod класса Second

Функционал: Выдача сигнала.

Параметры: string& text - ссылка на текст сообщения.

Возвращаемое значение: -.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода signalMethod класса Second

№	Предикат	Действия	№ перехода
1		Вывод с новой строки: Signal from (значение, возвращаемое методом getAbsoluteCoordinate, вызванным у текущего объекта)	2

№	Предикат	Действия	№ перехода
2		Добавляение в переменную text подстроки " (class: 2)"	∅

3.3 Алгоритм метода handlerMethod класса Second

Функционал: обработка сигнала.

Параметры: string text - текст сообщения.

Возвращаемое значение: -.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода handlerMethod класса Second

№	Предикат	Действия	№ перехода
1		Вывод с новой строки: Signal to (значение, возвращаемое методом getAbsoluteCoordinate, вызванным у текущего объекта) Text: (значение параметра text)	∅

3.4 Алгоритм конструктора класса Third

Функционал: Вызывает конструктор класса Base, присваивает номер класса.

Параметры: Base* parent - указатель на родительский объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 5.

Таблица 5 – Алгоритм конструктора класса Third

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса Base с параметрами parent, name	2
2		Присвоение полю classNumber значения 3	∅

3.5 Алгоритм метода signalMethod класса Third

Функционал: Выдача сигнала.

Параметры: string& text - ссылка на текст сообщения.

Возвращаемое значение: -.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода signalMethod класса Third

№	Предикат	Действия	№ перехода
1		Вывод с новой строки: Signal from (значение, возвращаемое методом getAbsoluteCoordinate, вызванным у текущего объекта)	2
2		Добавляение в переменную text подстроки " (class: 3)"	∅

3.6 Алгоритм метода handlerMethod класса Third

Функционал: обработка сигнала.

Параметры: string text - текст сообщения.

Возвращаемое значение: -.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода handlerMethod класса Third

№	Предикат	Действия	№ перехода
1		Вывод с новой строки: Signal to (значение, возвращаемое методом getAbsoluteCoordinate, вызванным у текущего объекта) Text: (значение параметра text)	∅

3.7 Алгоритм конструктора класса Fourth

Функционал: Вызывает конструктор класса Base, присваивает номер класса.

Параметры: Base* parent - указатель на родительский объект, string name -

имя объекта.

Алгоритм конструктора представлен в таблице 8.

Таблица 8 – Алгоритм конструктора класса Fourth

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса Base с параметрами parent, name	2
2		Присвоение полю classNumber значения 4	∅

3.8 Алгоритм конструктора класса Fifth

Функционал: Вызывает конструктор класса Base, присваивает номер класса.

Параметры: Base* parent - указатель на родительский объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 9.

Таблица 9 – Алгоритм конструктора класса Fifth

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса Base с параметрами parent, name	2
2		Присвоение полю classNumber значения 5	∅

3.9 Алгоритм конструктора класса Sixth

Функционал: Вызывает конструктор класса Base, присваивает номер класса.

Параметры: Base* parent - указатель на родительский объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 10.

Таблица 10 – Алгоритм конструктора класса Sixth

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса Base с параметрами parent, name	2

№	Предикат	Действия	№ перехода
2		Присвоение полю classNumber значения 6	∅

3.10 Алгоритм метода signalMethod класса Fourth

Функционал: Выдача сигнала.

Параметры: string& text - ссылка на текст сообщения.

Возвращаемое значение: -.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода signalMethod класса Fourth

№	Предикат	Действия	№ перехода
1		Вывод с новой строки: Signal from (значение, возвращаемое методом getAbsoluteCoordinate, вызванным у текущего объекта)	2
2		Добавляение в переменную text подстроки " (class: 4)"	∅

3.11 Алгоритм метода signalMethod класса Fifth

Функционал: Выдача сигнала.

Параметры: string& text - ссылка на текст сообщения.

Возвращаемое значение: -.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода signalMethod класса Fifth

№	Предикат	Действия	№ перехода
1		Вывод с новой строки: Signal from (значение, возвращаемое методом getAbsoluteCoordinate, вызванным у текущего объекта)	2
2		Добавляение в переменную text подстроки " (class: 5)"	∅

3.12 Алгоритм метода signalMethod класса Sixth

Функционал: Выдача сигнала.

Параметры: string& text - ссылка на текст сообщения.

Возвращаемое значение: -.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода signalMethod класса Sixth

№	Предикат	Действия	№ перехода
1		Вывод с новой строки: Signal from (значение, возвращаемое методом getAbsoluteCoordinate, вызванным у текущего объекта)	2
2		Добавляение в переменную text подстроки " (class: 6)"	∅

3.13 Алгоритм метода handlerMethod класса Sixth

Функционал: обработка сигнала.

Параметры: string text - текст сообщения.

Возвращаемое значение: -.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода handlerMethod класса Sixth

№	Предикат	Действия	№ перехода
1		Вывод с новой строки: Signal to (значение, возвращаемое методом getAbsoluteCoordinate, вызванным у текущего объекта) Text: (значение параметра text)	∅

3.14 Алгоритм метода handlerMethod класса Fourth

Функционал: обработка сигнала.

Параметры: string text - текст сообщения.

Возвращаемое значение: -.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода handlerMethod класса Fourth

№	Предикат	Действия	№ перехода
1		Вывод с новой строки: Signal to (значение, возвращаемое методом getAbsoluteCoordinate, вызванным у текущего объекта) Text: (значение параметра text)	∅

3.15 Алгоритм метода handlerMethod класса Fifth

Функционал: обработка сигнала.

Параметры: string text - текст сообщения.

Возвращаемое значение: -.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода handlerMethod класса Fifth

№	Предикат	Действия	№ перехода
1		Вывод с новой строки: Signal to (значение, возвращаемое методом getAbsoluteCoordinate, вызванным у текущего объекта) Text: (значение параметра text)	∅

3.16 Алгоритм конструктора класса App

Функционал: Вызывает конструктор класса Base, присваивает номер класса.

Параметры: Base* parent - указатель на родительский объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 17.

Таблица 17 – Алгоритм конструктора класса App

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса Base с параметрами parent, name	2
2		Присвоение полю classNumber значения 1	∅

3.17 Алгоритм метода signalMethod класса App

Функционал: Выдача сигнала.

Параметры: string& text - ссылка на текст сообщения.

Возвращаемое значение: -.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода signalMethod класса App

№	Предикат	Действия	№ перехода
1		Вывод с новой строки: Signal from (значение, возвращаемое методом getAbsoluteCoordinate, вызванным у текущего объекта)	2
2		Добавляение в переменную text подстроки " (class: 1)"	∅

3.18 Алгоритм метода handlerMethod класса App

Функционал: обработка сигнала.

Параметры: string text - текст сообщения.

Возвращаемое значение: -.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода handlerMethod класса App

№	Предикат	Действия	№ перехода
1		Вывод с новой строки: Signal to (значение, возвращаемое методом getAbsoluteCoordinate, вызванным у текущего объекта) Text: (значение параметра text)	∅

3.19 Алгоритм метода deleteConnection класса Base

Функционал: удаление связи между сигналом текущего объекта и обработчкиком сигналов целевого объекта.

Параметры: TYPE_SIGNAL signalMethod - ссылка на метод сигнала, handler - указатель на объект обработчик класса Base, TYPE_HANDLER handlerMethod - ссылка на метод обработчика.

Возвращаемое значение: -.

Алгоритм метода представлен в таблице 20.

Таблица 20 – Алгоритм метода deleteConnection класса Base

№	Предикат	Действия	№ перехода
1		Инициализация целочисленной переменной i = 0	2
2	i < кол-ва связей		3
			∅
3	i-тый элемент connections содержит передаваемые в параметрах метод сигнала текущего объекта, указатель на другой объект и метод обработчика другого объекта	Удаление i-того элемента connections	∅
		i++	2

3.20 Алгоритм метода emitSignal класса Base

Функционал: выдача сигнала от текущего объекта.

Параметры: TYPE_SIGNAL signalMethod, string text.

Возвращаемое значение: -.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода emitSignal класса Base

№	Предикат	Действия	№ перехода
1	Значение возвращаемое методом getState вызванным у данного элемента != 0		∅
			2
2	Кол-во связей равно 0		∅
			3
3		Вызов метода signalMethod текущего объекта с параметром text	4
4		Инициализация целочисленной переменной i = 0	5
5	i < кол-ва связей		6
			∅
6	signalMethod i-того элемента connections равен параметру signalMethod, и значение, возвращаемое методом getState объекта handler i-того элемента контейнера != 0	Вызов метода handlerMethod у i-того элемента connections с параметром text	7
			7
7		i++	5

3.21 Алгоритм метода setConnection класса Base

Функционал: Установление связи между сигналом текущего объекта и обработчиком сигналов целевого объекта.

Параметры: TYPE_SIGNAL signalMethod - ссылка на метод сигнала, Base* handler - указатель на объект-обработчик класса Base, TYPE_HANDLER handlerMethod - ссылка на метод обработчика.

Возвращаемое значение: -.

Алгоритм метода представлен в таблице 22.

Таблица 22 – Алгоритм метода setConnection класса Base

№	Предикат	Действия	№ перехода
1		Инициализаци целочисленной переменной i равной 0	2
2	i < кол-ва связей		3
			4
3	i-тый элемент connections содержит передаваемые в параметрах метод сигнала текущего объекта, указатель на другой объект и метод обработчика другого объекта		∅
		i++	2
4		Объявление объекта connection структуры connectionStruct	5
5		signalMethod объекта connection равно параментру signalMethod	6
6		Обработчик сигналов объекта connection = параметру handler	7
7		handlerMethod объекта connection = параметру handlerMethod	8
8		Добавление в конец контейнера connections объекта connection	∅

3.22 Алгоритм метода getAbsoluteCoordinate класса Base

Функционал: Определение абсолютного пути к объекту.

Параметры: string absoluteCoordinate - координаты объекта.

Возвращаемое значение: string, абсолютый путь к объекту.

Алгоритм метода представлен в таблице 23.

Таблица 23 – Алгоритм метода getAbsoluteCoordinate класса Base

№	Предикат	Действия	№ перехода
1	Элемент является корневым	Возвращение значения "/"	∅
			2
2		Возвращение значения метода getAbsoluteCoordinate с параметром "/" + объекта parentObj + name + absoluteCoordinate	∅

3.23 Алгоритм метода getClassNumber класса Base

Функционал: Получение номера класса текущего объекта.

Параметры: -.

Возвращаемое значение: int, номер класса.

Алгоритм метода представлен в таблице 24.

Таблица 24 – Алгоритм метода getClassNumber класса Base

№	Предикат	Действия	№ перехода
1		Возвращение значения поля classNumber	∅

3.24 Алгоритм метода setReadyForAll класса Base

Функционал: Переведение всех объектов иерархии в состояние готовности.

Параметры: int state, устанавливаемая готовность объектов.

Возвращаемое значение: -.

Алгоритм метода представлен в таблице 25.

Таблица 25 – Алгоритм метода setReadyForAll класса Base

№	Предикат	Действия	№ перехода
1		Вызов метода setState текущего объекта с параметром state	2
2		Инициализация целочисленной переменной i = 0	3
3	i < кол-ва подчиненных объектов	Вызов метода setReadyForAll у i-того дочернего объекта	4
			∅
4		i++	3

3.25 Алгоритм метода getSignalMethod класса App

Функционал: Возвращает ссылку на метод сигнала нужного класса.

Параметры: Base* pointer - указатель на нужный объект.

Возвращаемое значение: TYPE_SIGNAL, метод сигнала нужного объекта.

Алгоритм метода представлен в таблице 26.

Таблица 26 – Алгоритм метода getSignalMethod класса App

№	Предикат	Действия	№ перехода
1	Значение, возаращаемое методом getClassNumber у объекта pointer == 1	Возвращение результата параметризированного макроопределения SIGNAL_D с параметром ссылки на метод signalMethod класса App	∅
			2
2	Значение, возаращаемое методом getClassNumber у объекта pointer == 2	Возвращение результата параметризированного макроопределения SIGNAL_D с параметром ссылки на метод signalMethod класса Second	∅
			3
3	Значение, возаращаемое методом getClassNumber у объекта pointer == 3	Возвращение результата параметризированного макроопределения SIGNAL_D с параметром ссылки на метод signalMethod класса Third	∅

№	Предикат	Действия	№ перехода
			4
4	Значение, возаращаемое методом getClassNumber у объекта pointer == 4	Возвращение результата параметризированного макроопределения SIGNAL_D с параметром ссылки на метод signalMethod класса Fourth	∅
			5
5	Значение, возаращаемое методом getClassNumber у объекта pointer == 5	Возвращение результата параметризированного макроопределения SIGNAL_D с параметром ссылки на метод signalMethod класса Fifth	∅
			6
6	Значение, возаращаемое методом getClassNumber у объекта pointer == 6	Возвращение результата параметризированного макроопределения SIGNAL_D с параметром ссылки на метод signalMethod класса Sixth	∅
			∅

3.26 Алгоритм метода getHandlerMethod класса App

Функционал: Получение ссылки на метод обработки нужного класса.

Параметры: Base* pointer - указатель на нужный объект.

Возвращаемое значение: TYPE_HANDLER, метод обработки нужного объекта.

Алгоритм метода представлен в таблице 27.

Таблица 27 – Алгоритм метода getHandlerMethod класса App

№	Предикат	Действия	№ перехода
1	Значение, возаращаемое методом getClassNumber у объекта pointer == 1	Возвращение результата параметризированного макроопределения HANDLER_D с параметром ссылки на метод handlerMethod класса App	∅
			2
2	Значение, возаращаемое	Возвращение результата параметризированного	∅

№	Предикат	Действия	№ перехода
	методом getClassNumber у объекта pointer == 2	макроопределения HANDLER_D с параметром ссылки на метод handlerMethod класса Second	
			3
3	Значение, возаращаемое методом getClassNumber у объекта pointer == 3	Возвращение результата параметризированного макроопределения HANDLER_D с параметром ссылки на метод handlerMethod класса Third	∅
			4
4	Значение, возаращаемое методом getClassNumber у объекта pointer == 4	Возвращение результата параметризированного макроопределения HANDLER_D с параметром ссылки на метод handlerMethod класса Fourth	∅
			5
5	Значение, возаращаемое методом getClassNumber у объекта pointer == 5	Возвращение результата параметризированного макроопределения HANDLER_D с параметром ссылки на метод handlerMethod класса Fifth	∅
			6
6	Значение, возаращаемое методом getClassNumber у объекта pointer == 6	Возвращение результата параметризированного макроопределения HANDLER_D с параметром ссылки на метод handlerMethod класса Sixth	∅
			∅

3.27 Алгоритм метода buildTree класса App

Функционал: Построение дерева иерархии объектов согласно пользовательскому вводу, ввод и построение множества связей "сигнал-обработчик" для заданных пар объектов.

Параметры: -.

Возвращаемое значение: -.

Алгоритм метода представлен в таблице 28.

Таблица 28 – Алгоритм метода buildTree класса App

№	Предикат	Действия	№ перехода
1		Обявление строковых переменных path и childName	2
2		Ввод значения path с клавиатуры	3
3		Вызов метода setName с параметром path	4
4		Объявление целочисленной переменной classNum	5
5		Объявление указателя parentObject на объект класса Base	6
6	Значение переменной path вводится с клавиатуры		7
			19
7	Значение переменной path равно "endtree"		19
			8
8		Присвоение parentObject значения возвращаемого методом findObjByPath с параметром path	9
9	Текущий объект - корневой	Вывод: Object tree(переход на новую строку)	10
			13
10		Вызов метода printHierarchyDepth	11
11		Вывод с новой строки: The head object (значение переменной path) is not found	12
12		Вызов функции exit с параметром 0	13
13		Ввод значений переменных childName и classNum с клавиатуры через разделитель	14
14	Значение classNum равно 2	Создание объекта класса Second с параметрами parentObject и childName передаваемыми в конструктор класса	19
			15

№	Предикат	Действия	№ перехода
15	Значение classNum равно 3	Создание объекта класса Third с параметрами parentObject и childName передаваемыми в конструктор класса	19
			16
16	Значение classNum равно 4	Создание объекта класса Fourth с параметрами parentObject и childName передаваемыми в конструктор класса	19
			17
17	Значение classNum равно 5	Создание объектакласса Fifth с параметрами parentObject и childName передаваемыми в конструктор класса	19
			18
18	Значение classNum равно 6	Создание объекта класса Sixth с параметрами parentObject и childName передаваемыми в конструктор класса	19
			19
19		Объявление строковых переменных signalCoordinate и handlerCoordinate	20
20		Объявление указателей signalObj и handlerObj на объекты класса Base	21
21		Ввод значения переменной signalCoordinate с клавиатуры	22
22	значение переменной signalCoordinate не равно "end_of_connection"	Ввод значения переменной handlerCoordinate с клавиатуры	23
			∅

№	Предикат	Действия	№ перехода
23		Присвоение signalObj значения возвращаемого методом findObjByPath с параметром signalCoordinate	24
24	значение signalObj не равно	Присвоение handlerObj значения, возвращаемого методом findObjByPath с параметром handlerCoordinate	25
		Вывод с новой строки: Object (значение переменной signalCoordinate) not found	26
25	значение handlerObj не равнонулевому указателю	Вызов метода setConnection у объекта signalObj с параметрами: значение возвращаемое методом getSignalMethod с параметром signalObj, handlerObj, значение, возвращаемое метдом getHandlerMethod с параметром handlerObj	26
		Вывод с новой строки: Handler object (значение переменной handlerCoordinate) not found	26
26		Ввод значения переменной signalCoordinate с клавиатуры	∅

3.28 Алгоритм метода execute класса App

Функционал: Вывод дерева иерархии, обработка команд пользователя.

Параметры: -.

Возвращаемое значение: int, код ошибки.

Алгоритм метода представлен в таблице 29.

Таблица 29 – Алгоритм метода execute класса App

№	Предикат	Действия	№ перехода
1		Вывод: Object tree (переход на новую строку)	2
2		Вызов метода printHierarchyDepth	3
3		Вызов метода setReadyForAll с параметром 1	4
4		Объявление строковых переменных signalCoordinate, handlerCoordinate, command, text	5
5		Объявление указателей signalObj, handlerObj на	6

№	Предикат	Действия	№ перехода
		объекты класса Base	
6		Объявление целочисленной переменной ready	7
7		Ввод значения переменной command с клавиатуры	8
8	значение переменной command не равно "END"	Ввод значения переменной signalCoordinate	9
			23
9		signalObj присваивается значение, возвращаемое методом findObjByPath с параметром signalCoordinate	10
10	значение переменной command равно "EMIT"		11
			13
11	signalObj != nullptr	Вызов функции getline лдя извлечения данных из входного потока до строкового разделителя в переменную text	12
		Вывод с новой строки: Object (значение переменной signalCoordinte) not found	22
12		Вызов метода emitSignal у объекта signalObj с параметрами: возвращаемое значение метода getSignalMethod с параметром signalObj, text	22
13	значение переменной command равно "SET_CONNECT"	Ввод значения переменной handler coordinate	14
			16
14	signalObj != nullptr	HandlerObj присваивается значение возвращаемое методом findObjByPath с параметром handlerCoordinate	15
		Вывод с новой строки: Object (значение переменной signalCoordinate) not found	22
15	handlerObj != nullptr	Вызов метода setConnection у объекта signalObj с	22

№	Предикат	Действия	№ перехода
		параметрами: возвращаемое значение метода getSignalMethod с параметром signalObj, handlerObj, возвращаемое значение метода getHandlerMethod с параметром handlerObj	
		Вывод с новой строки: Handler object (значение переменной handlerCoordinate) not found	22
16	значение переменной command равно "DELETE_CONNECT"	Ввод значения переменной handlerCoordinate с клавиатуры	17
			19
17	signalObj != nullptr	присвоение handlerObj значения, возвращаемого методом findObjByPath с параметром handlerCoordinate	18
		Вывод с новой строки: Object (значение переменной signalCoordinate) not found	22
18	handlerObj != nullptr	Вызов метода deleteConnection у объекта signalObj с параметрами: возвращаемое значение метода getSignalMethod с параметром signalObj, handlerObj, возвращаеое значение метода getHandlerMethod с параметром handlerObj	22
		Вывод с новой строки: Handler object (значение переменно handlerCoordinate) not found	22
19	значение переменной command равно "SET_CONDITION"		20
			22
20	signalObj != nullptr	Ввод значения переменной ready	21
		Вывод с новой строки: Object (значение переменной signalCoordinate) not found	22

№	Предикат	Действия	№ перехода
21		Вызво метода setState у объекта signalObj с параметром ready	22
22		Ввод значения переменной command с клавиатуры	8
23		Возвращение значения метода 0	∅

3.29 Алгоритм функции main

Функционал: Создает объект приложения, вызывает у него метод построения дерева иерархии и возвращает значение метода запуска приложения.

Параметры: -.

Возвращаемое значение: int, код ошибки.

Алгоритм функции представлен в таблице 30.

Таблица 30 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создание объекта AppObj класса App с параметром nullptr	2
2		Вызов метода построения дерева иерархии у объекта AppObj	3
3		Возвращение значения метода execute, вызываемого у объекта AppObj	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-22.

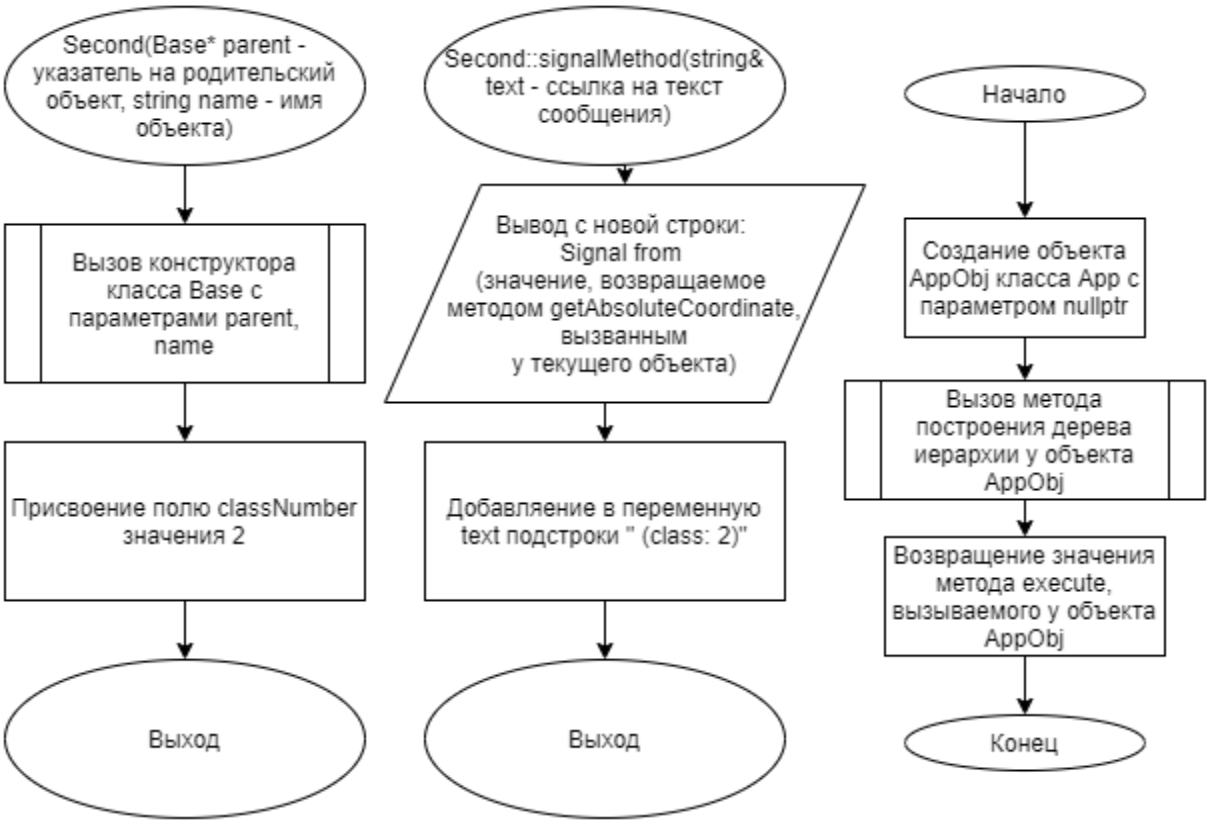

Рисунок 1 – Блок-схема алгоритма

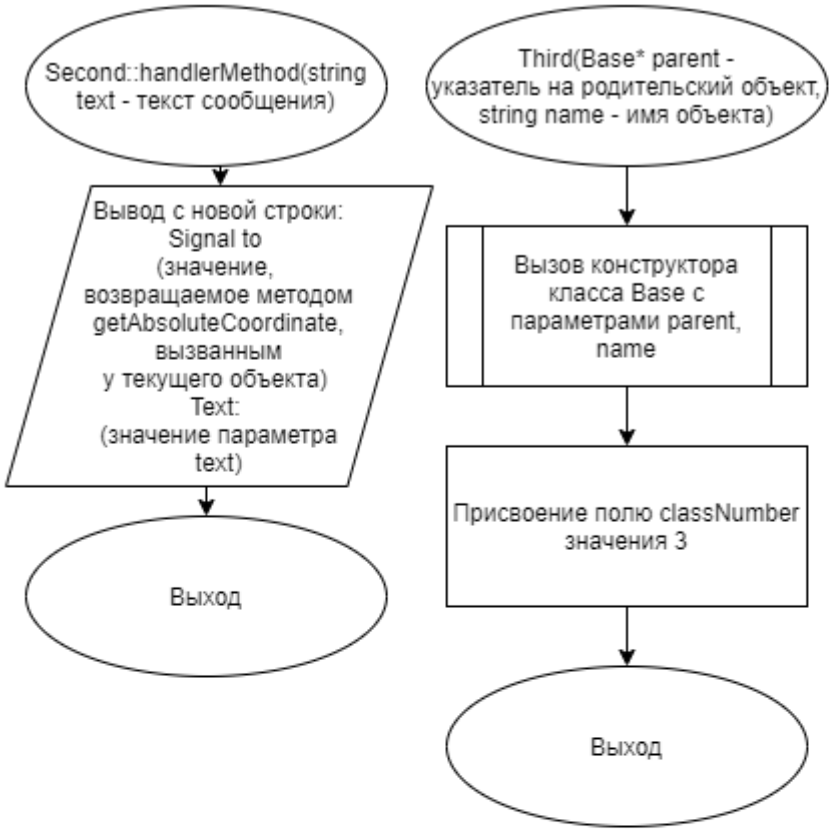

Рисунок 2 – Блок-схема алгоритма

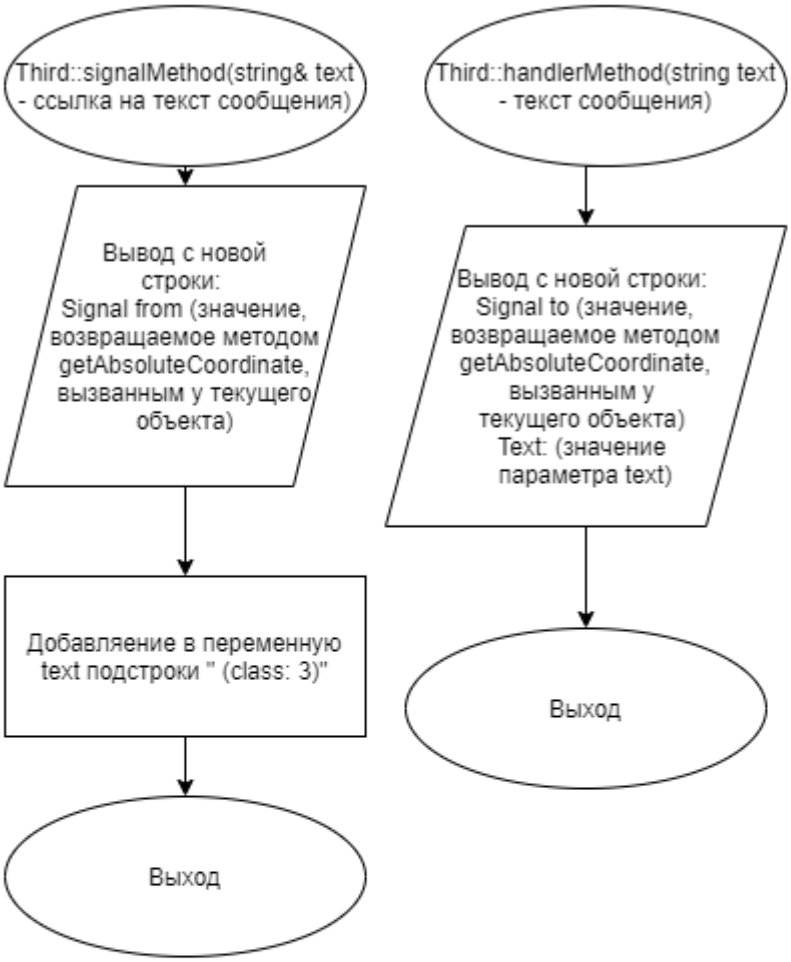

Рисунок 3 – Блок-схема алгоритма

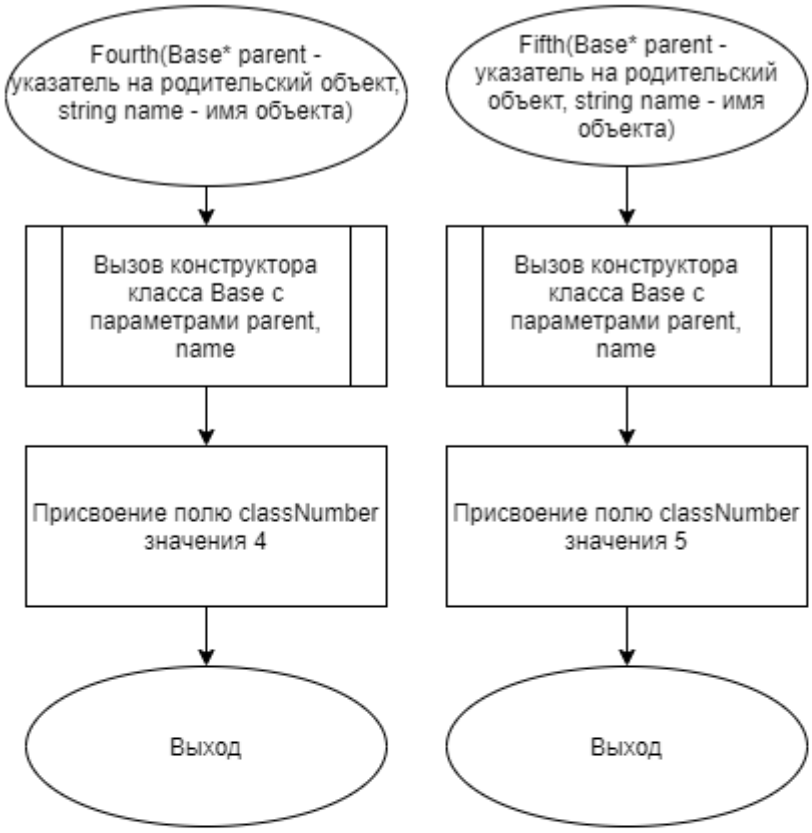

Рисунок 4 – Блок-схема алгоритма

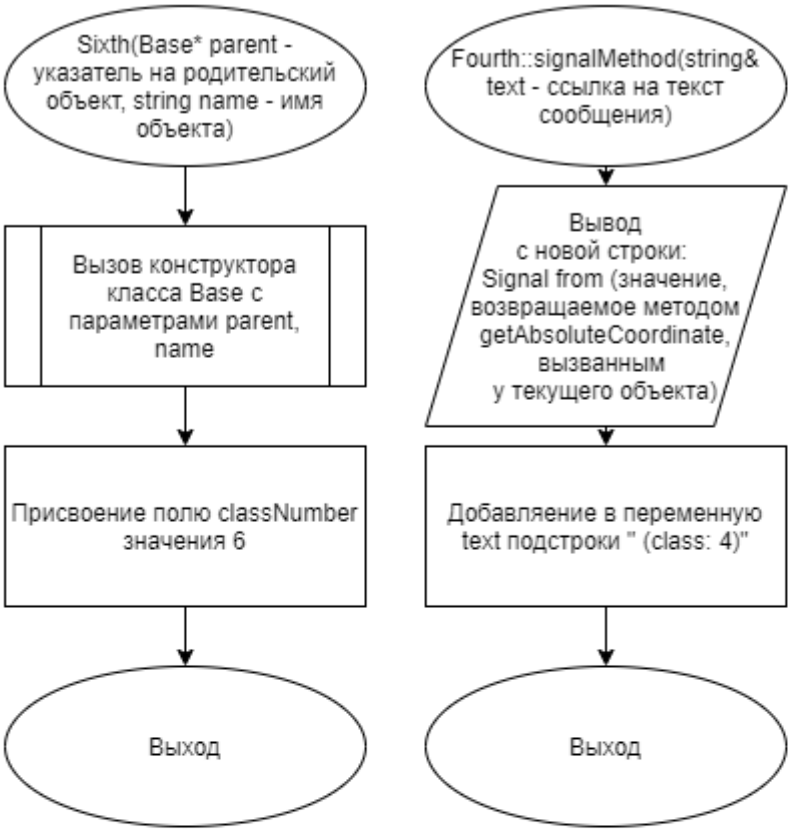

Рисунок 5 – Блок-схема алгоритма

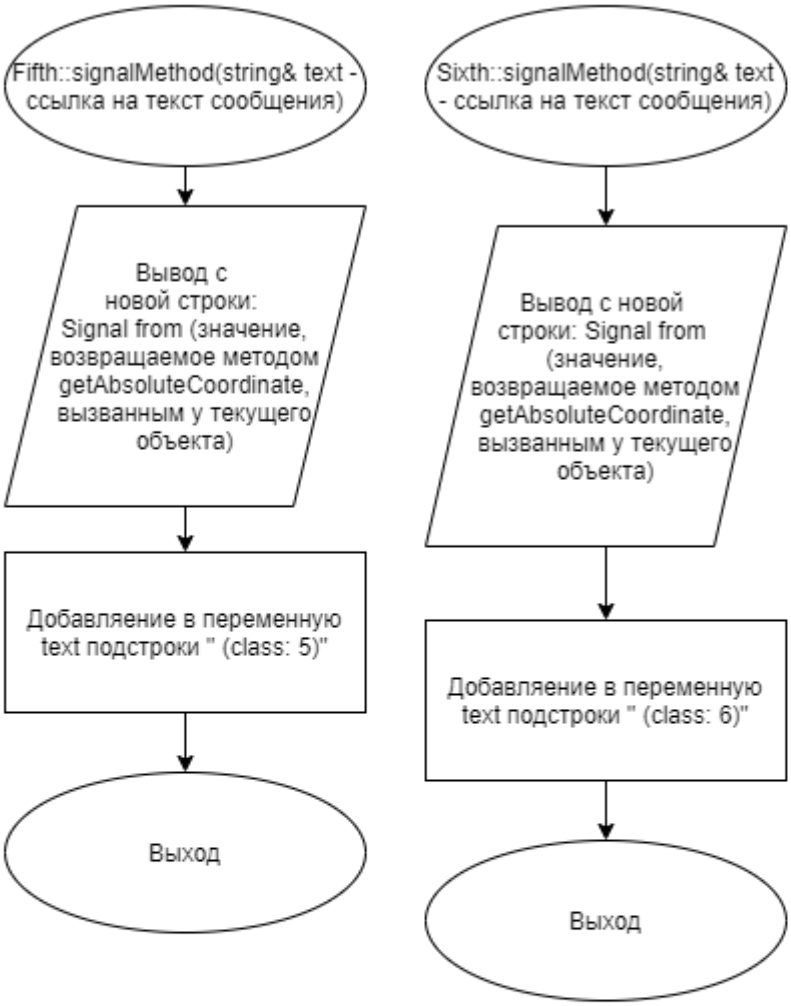

Рисунок 6 – Блок-схема алгоритма

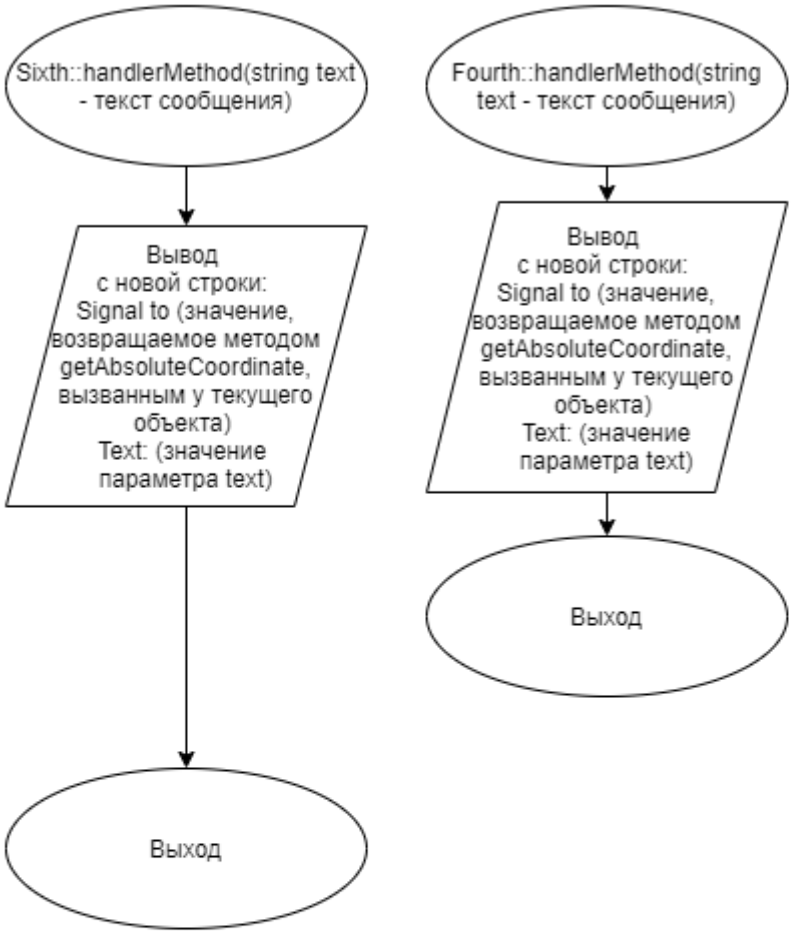

Рисунок 7 – Блок-схема алгоритма

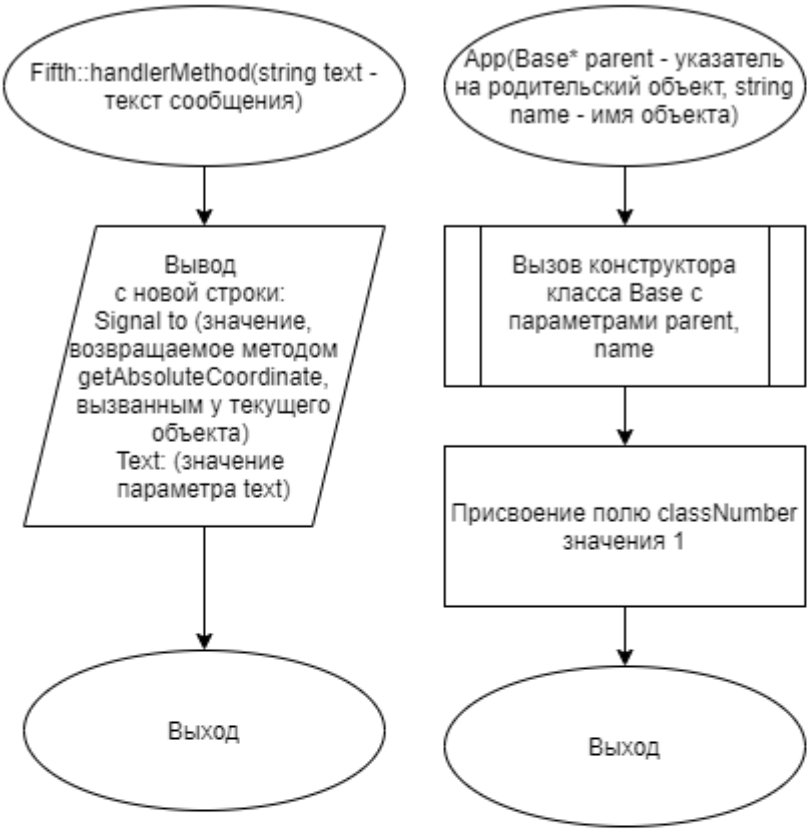

Рисунок 8 – Блок-схема алгоритма

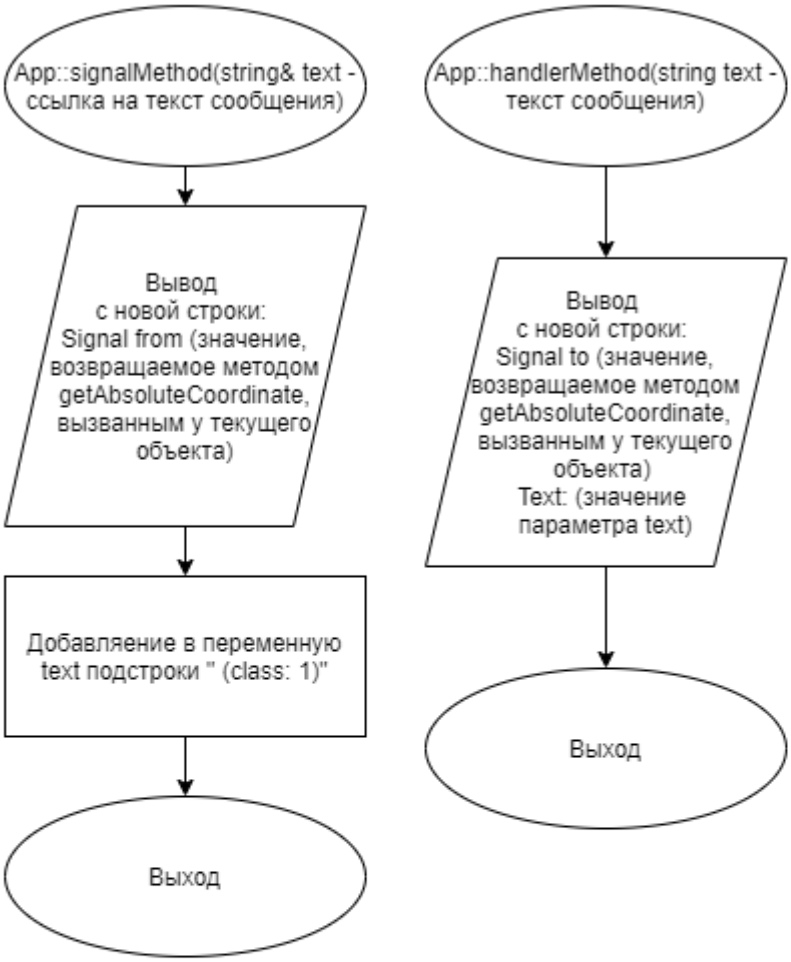

Рисунок 9 – Блок-схема алгоритма

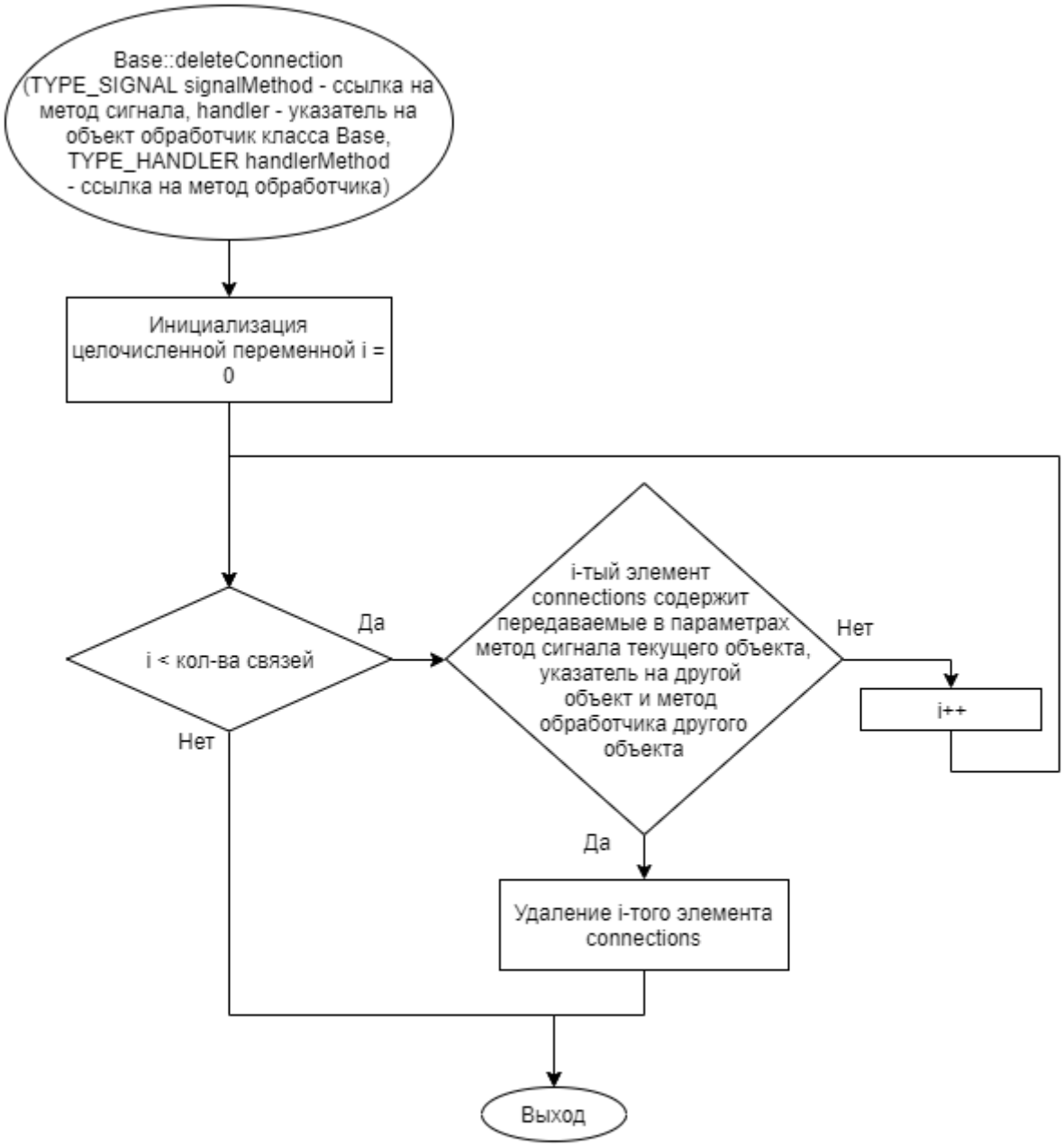

Рисунок 10 – Блок-схема алгоритма

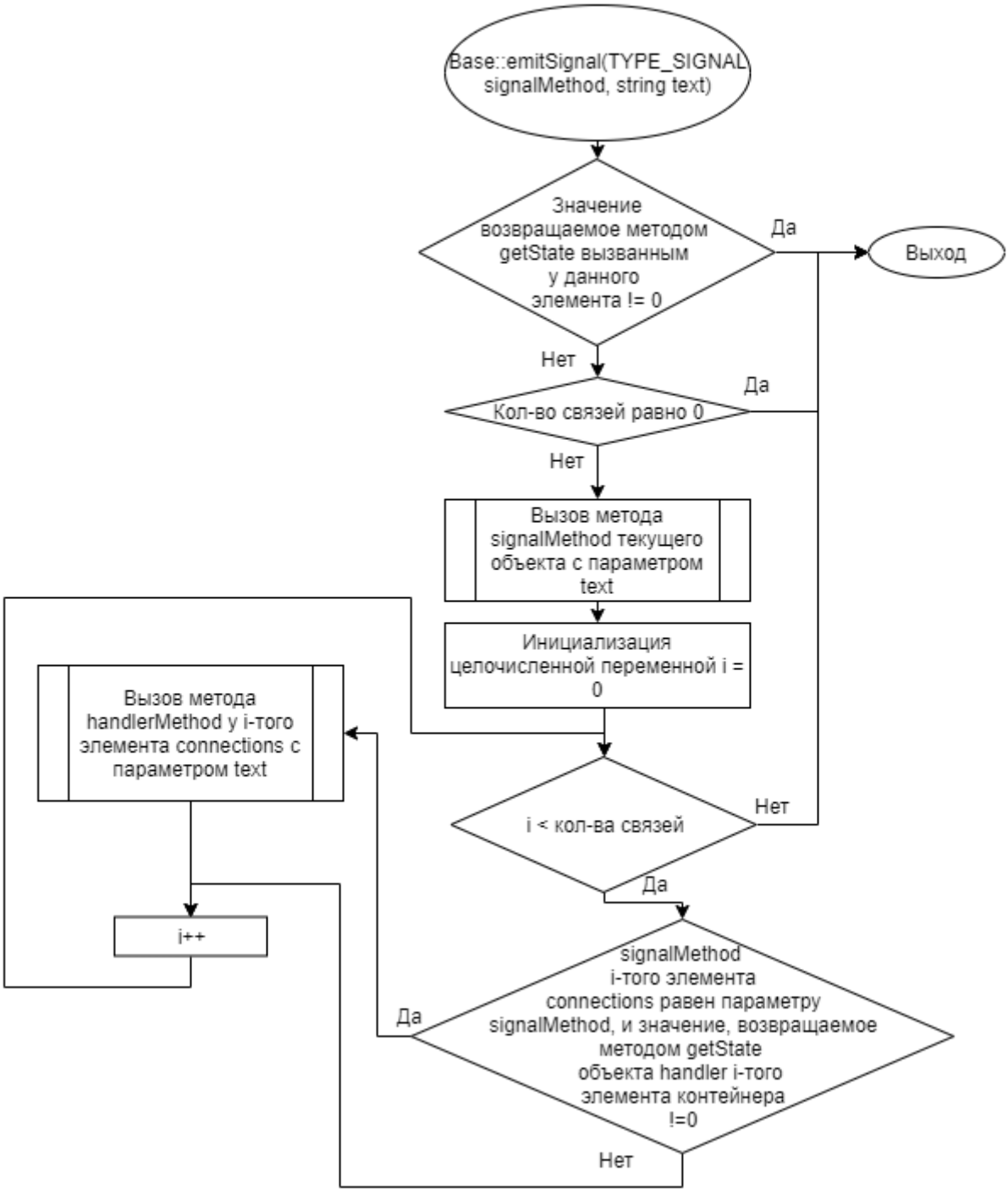

Рисунок 11 – Блок-схема алгоритма

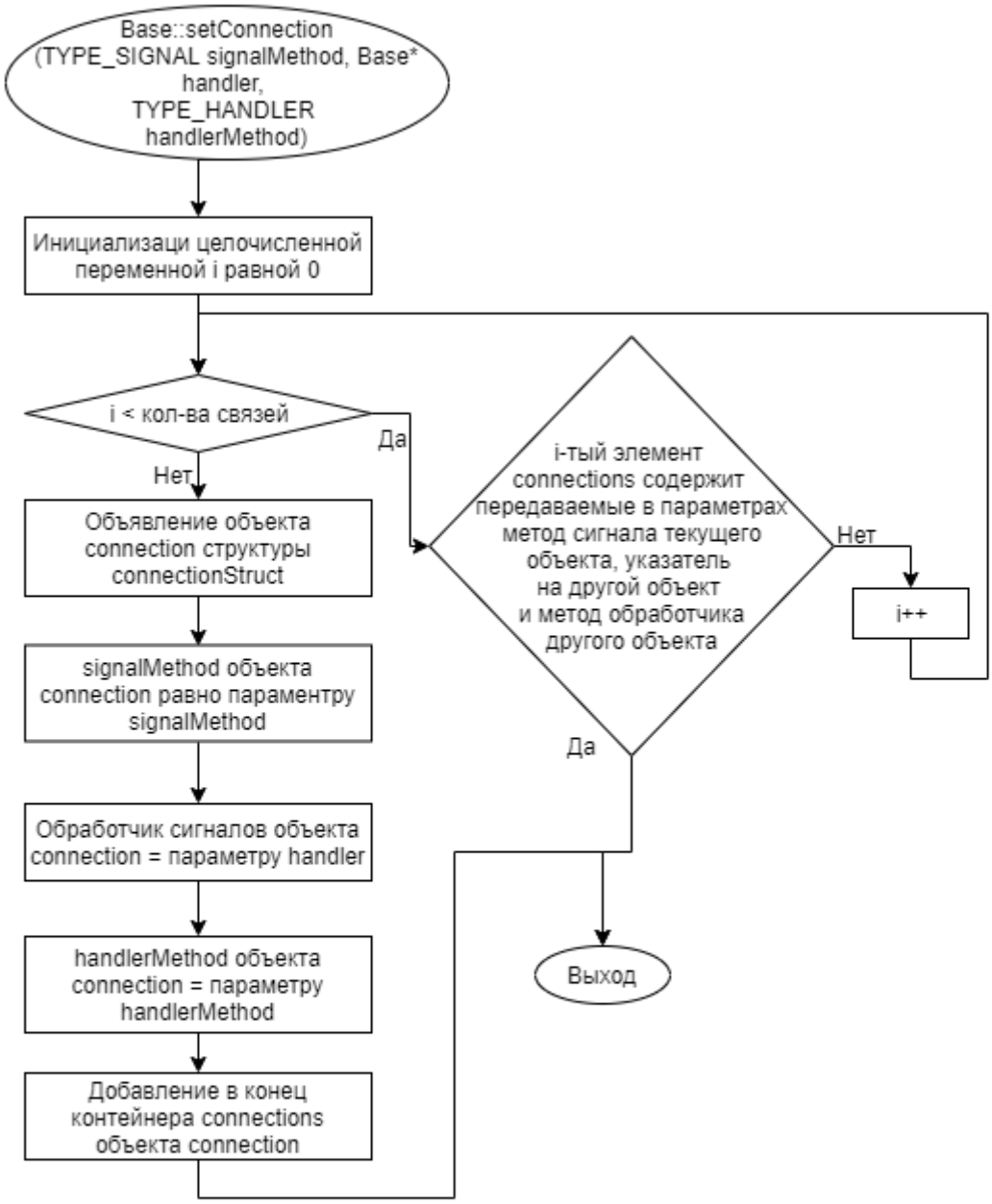

Рисунок 12 – Блок-схема алгоритма

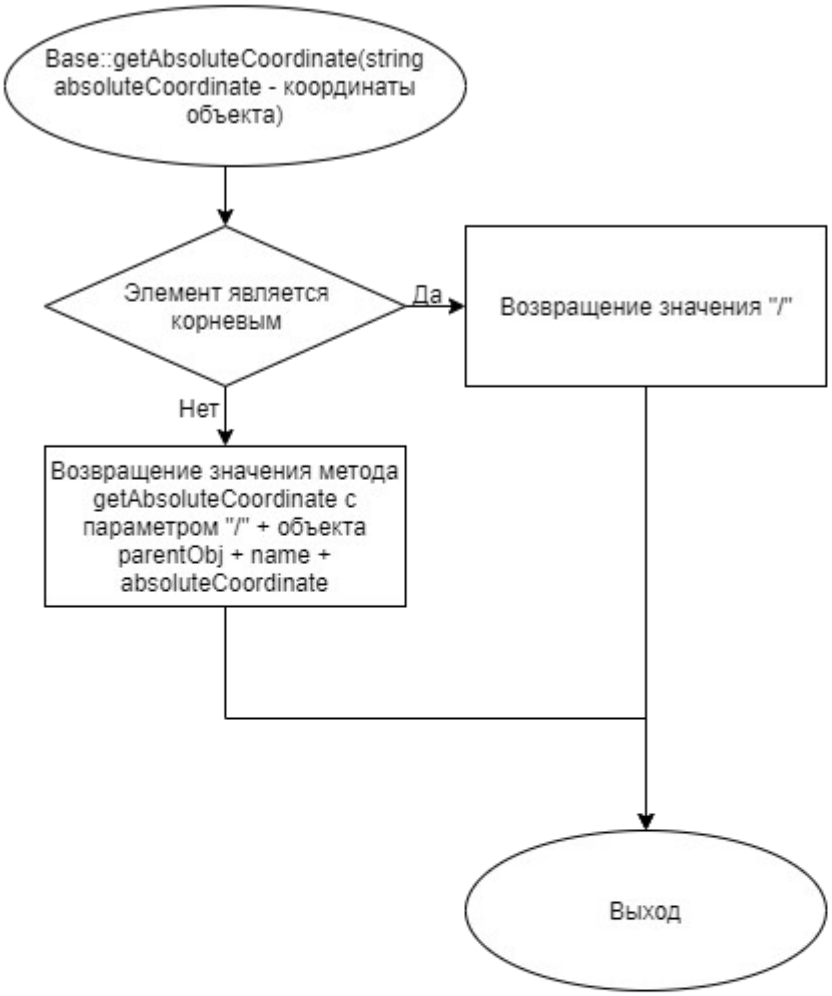

Рисунок 13 – Блок-схема алгоритма

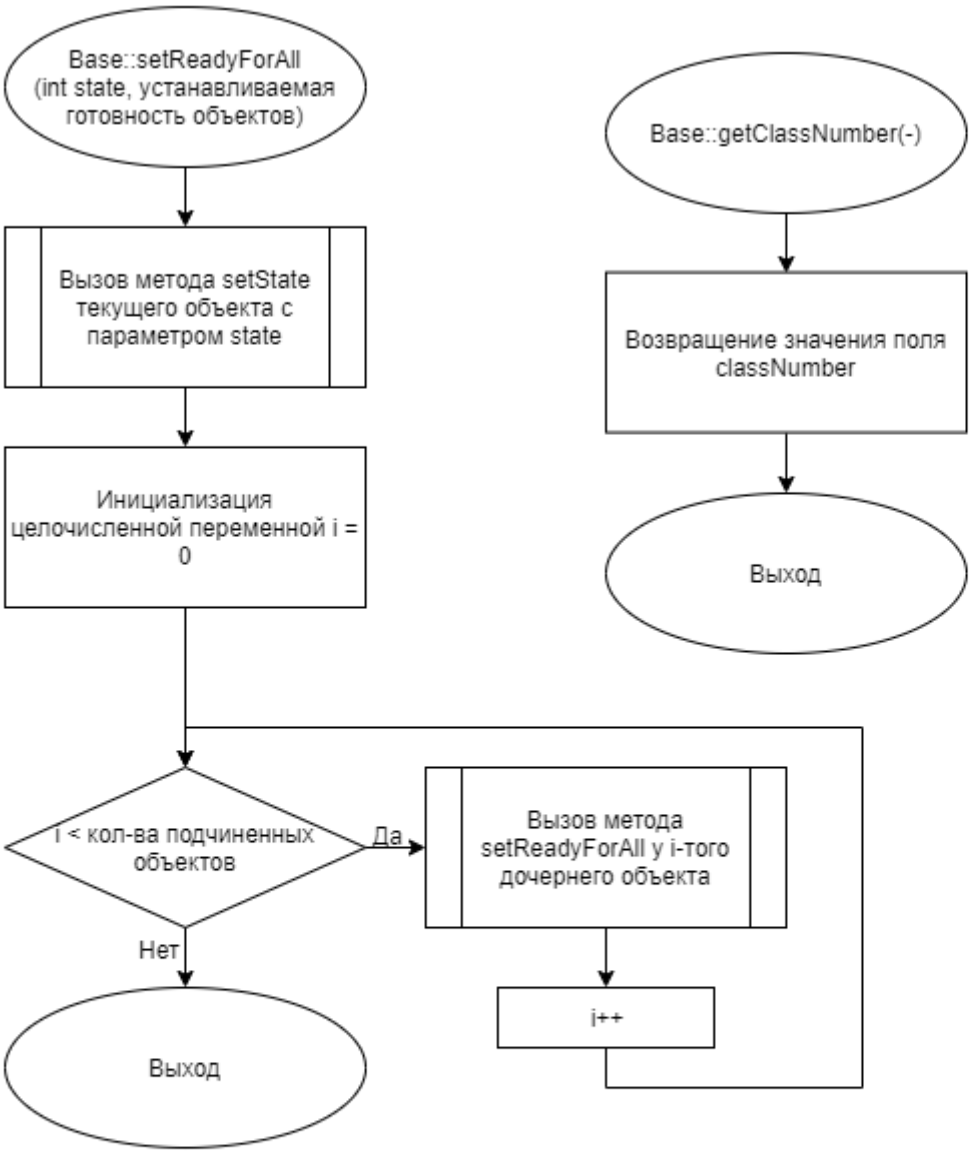

Рисунок 14 – Блок-схема алгоритма

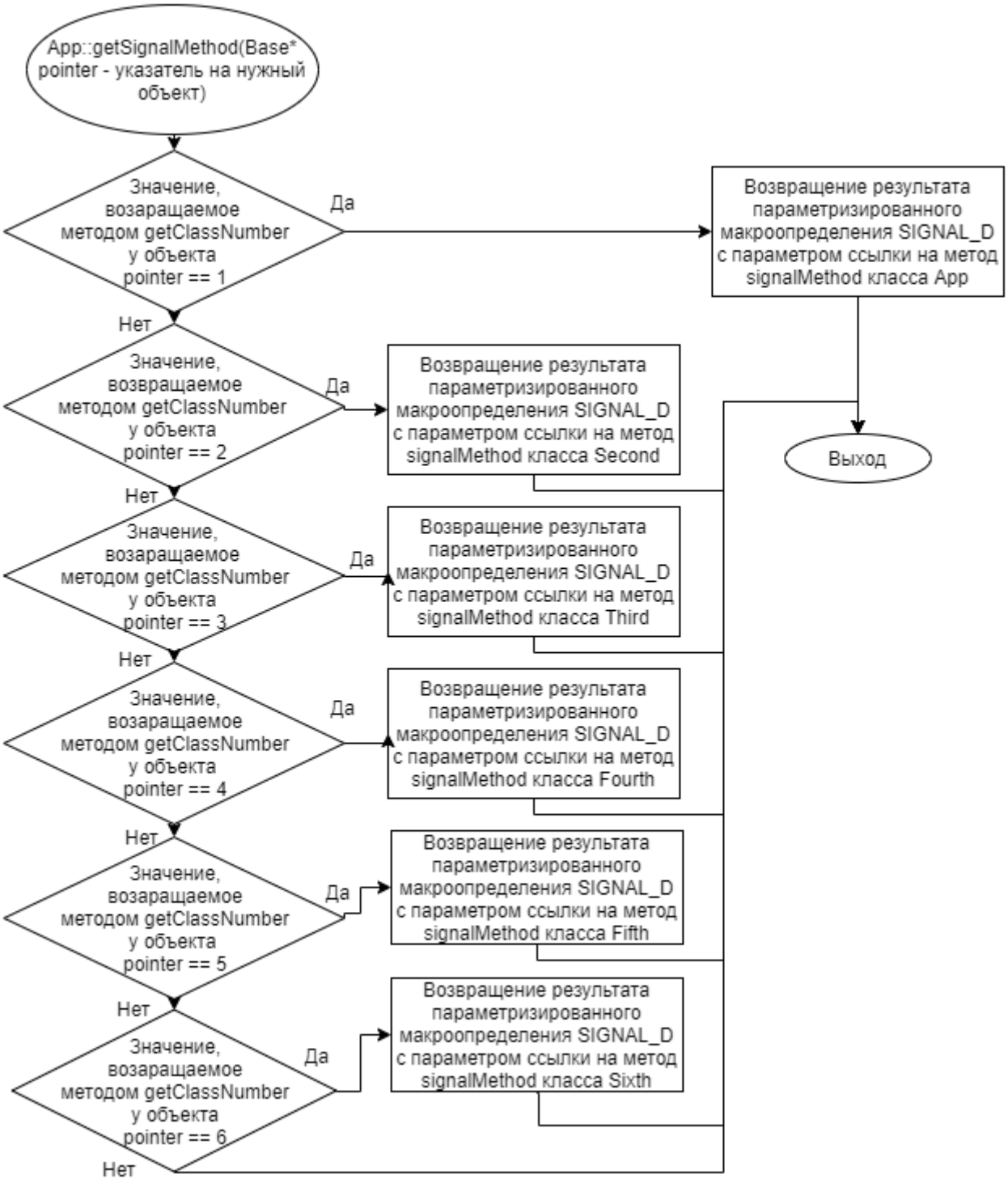

Рисунок 15 – Блок-схема алгоритма

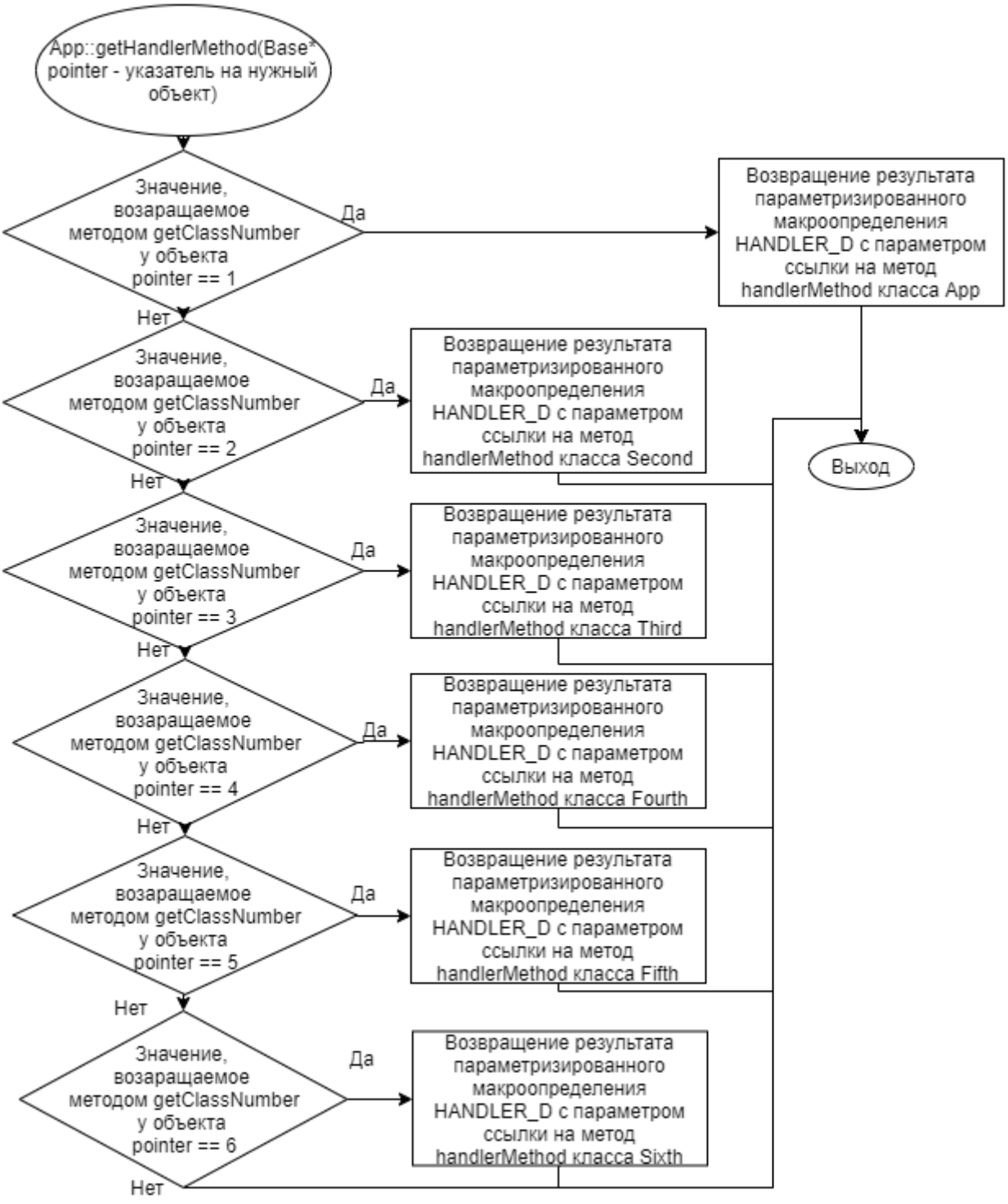

Рисунок 16 – Блок-схема алгоритма

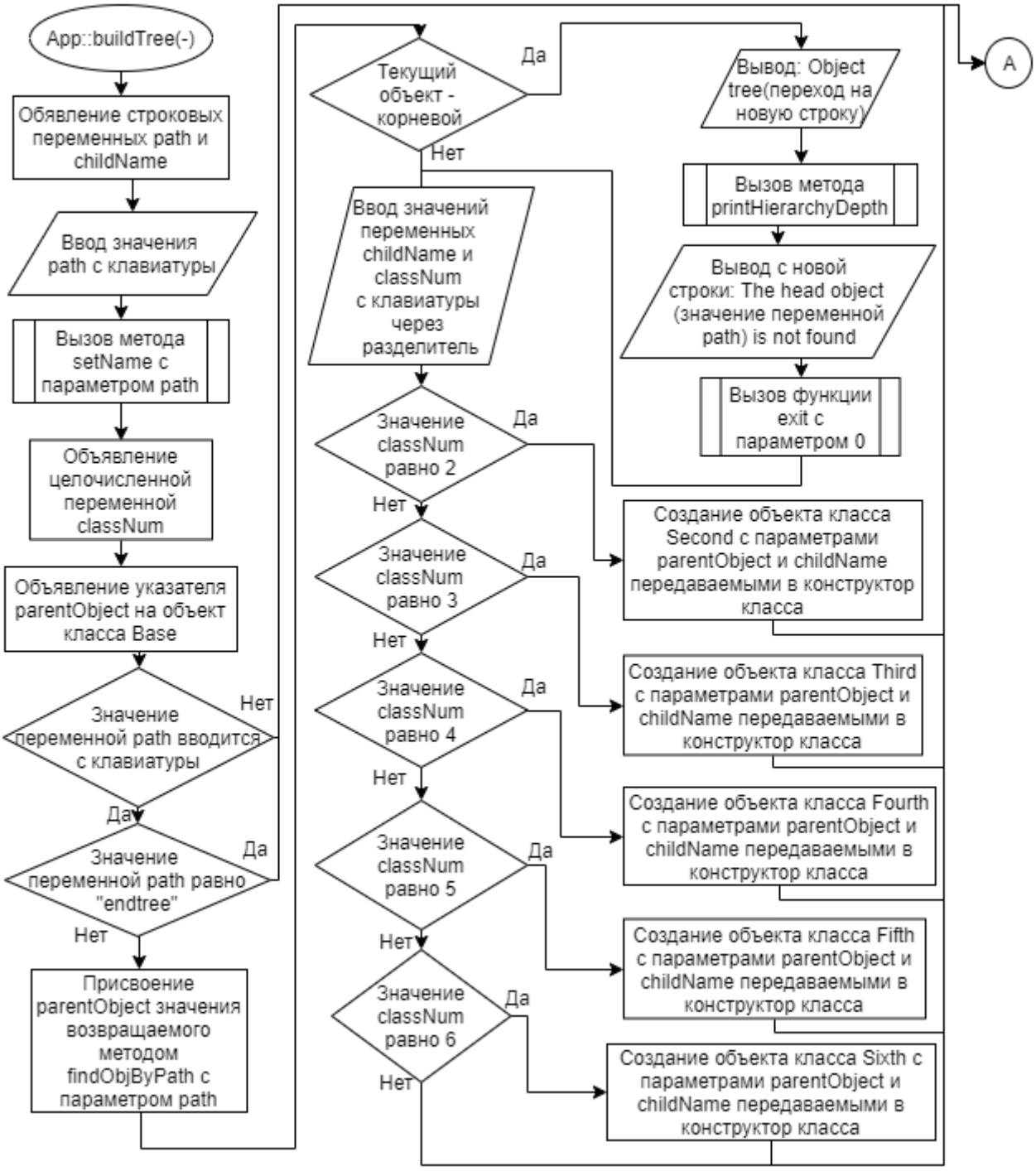

Рисунок 17 – Блок-схема алгоритма

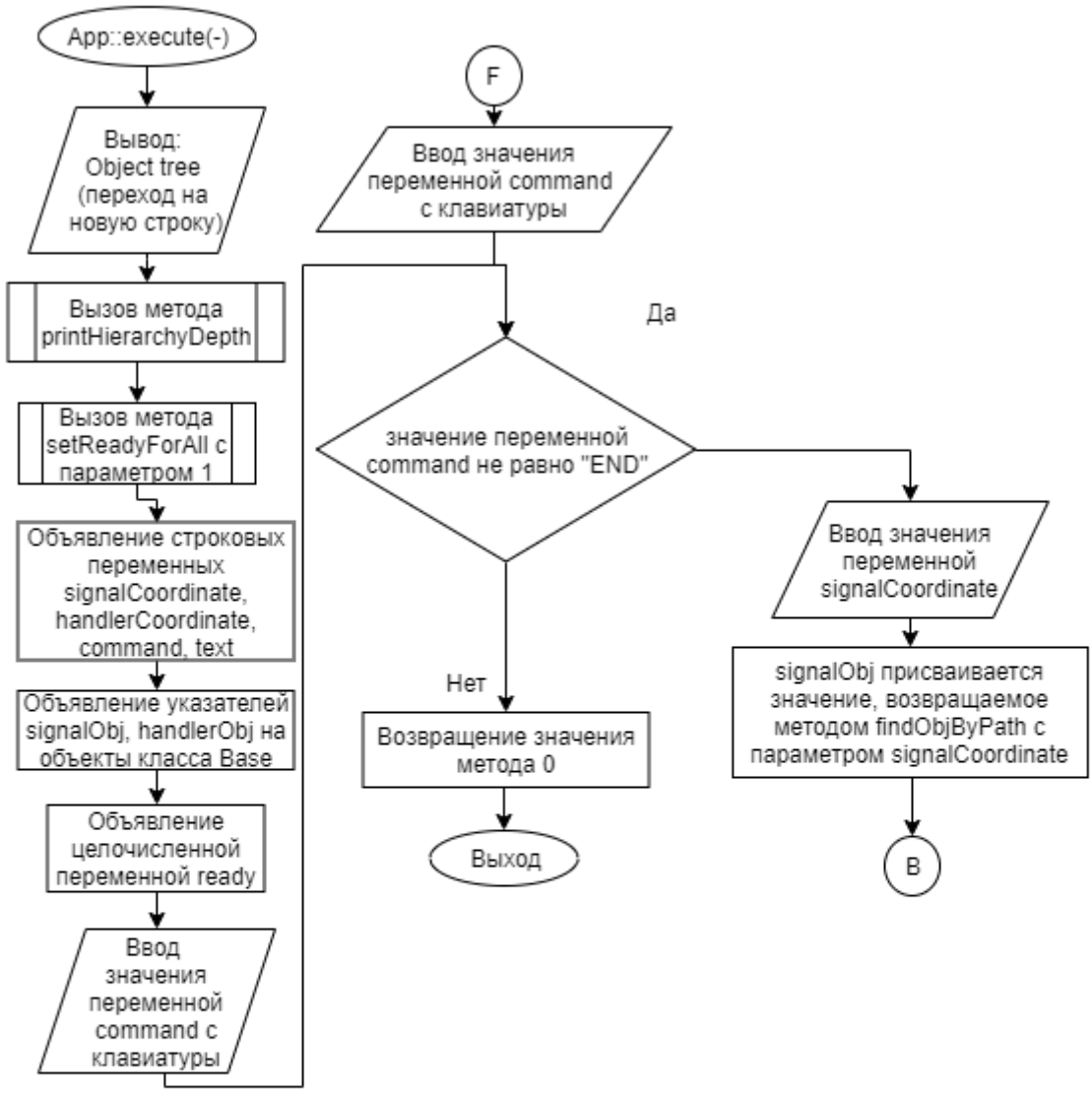

Рисунок 18 – Блок-схема алгоритма

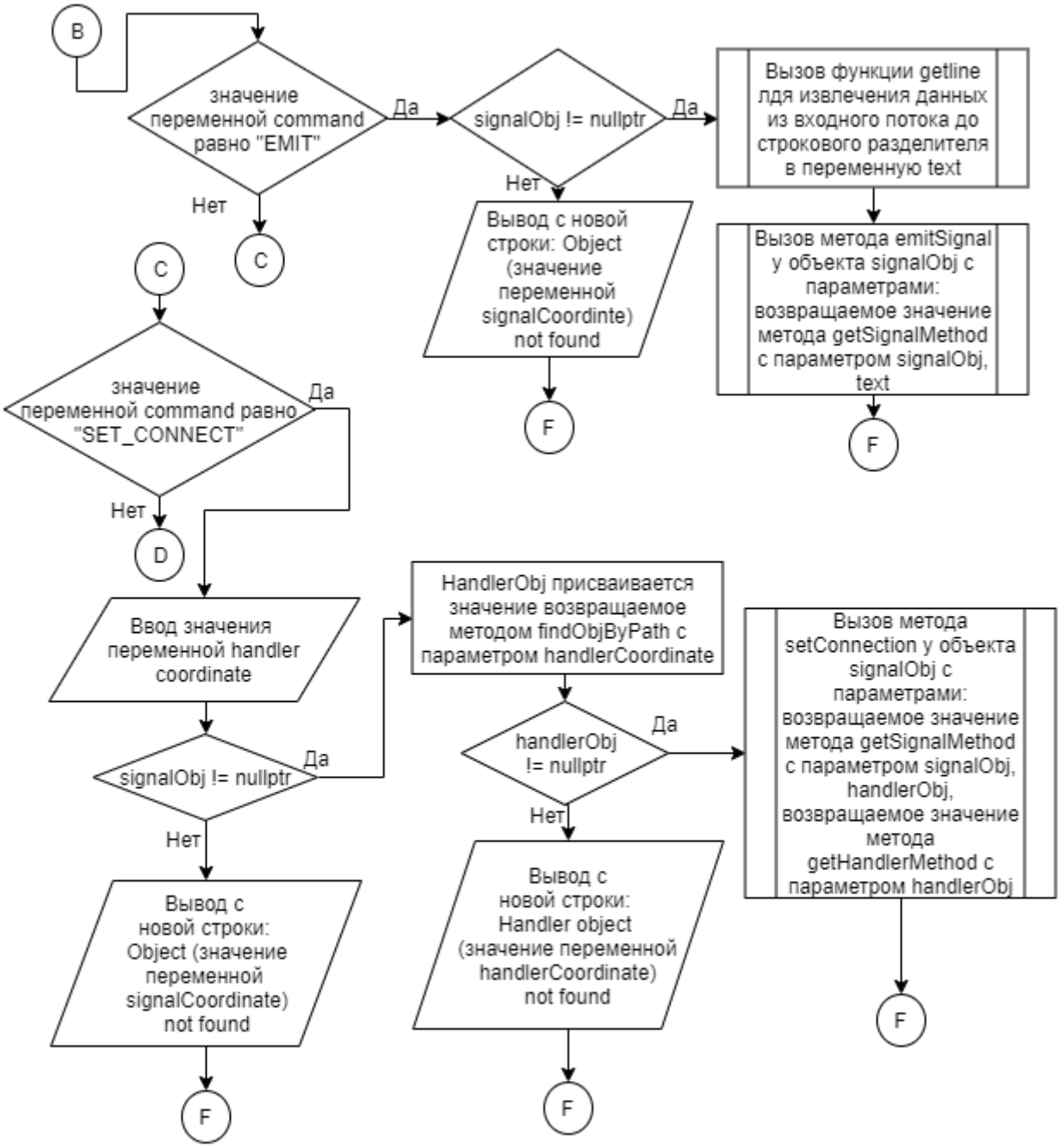

Рисунок 19 – Блок-схема алгоритма

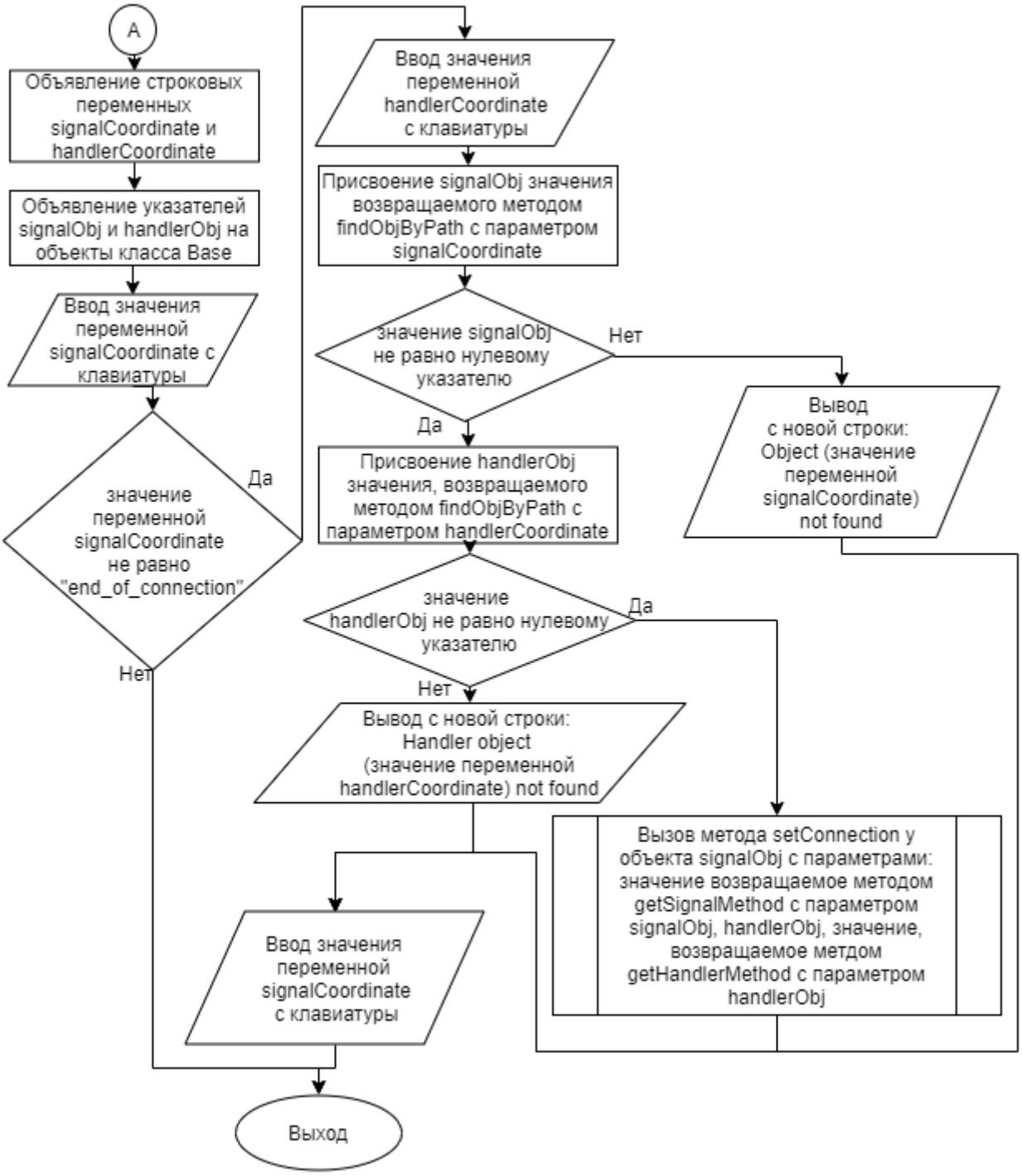

Рисунок 20 – Блок-схема алгоритма

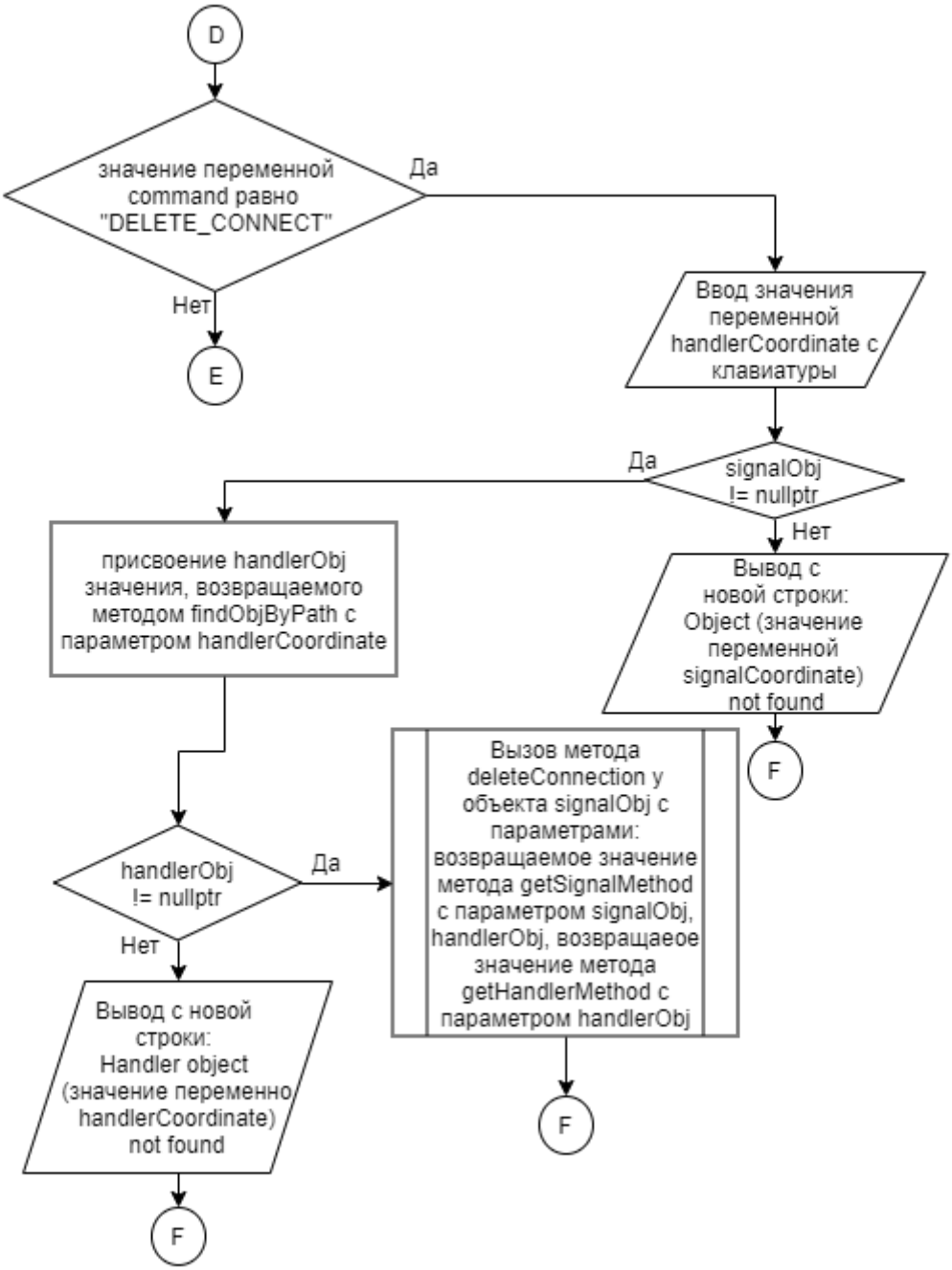

Рисунок 21 – Блок-схема алгоритма

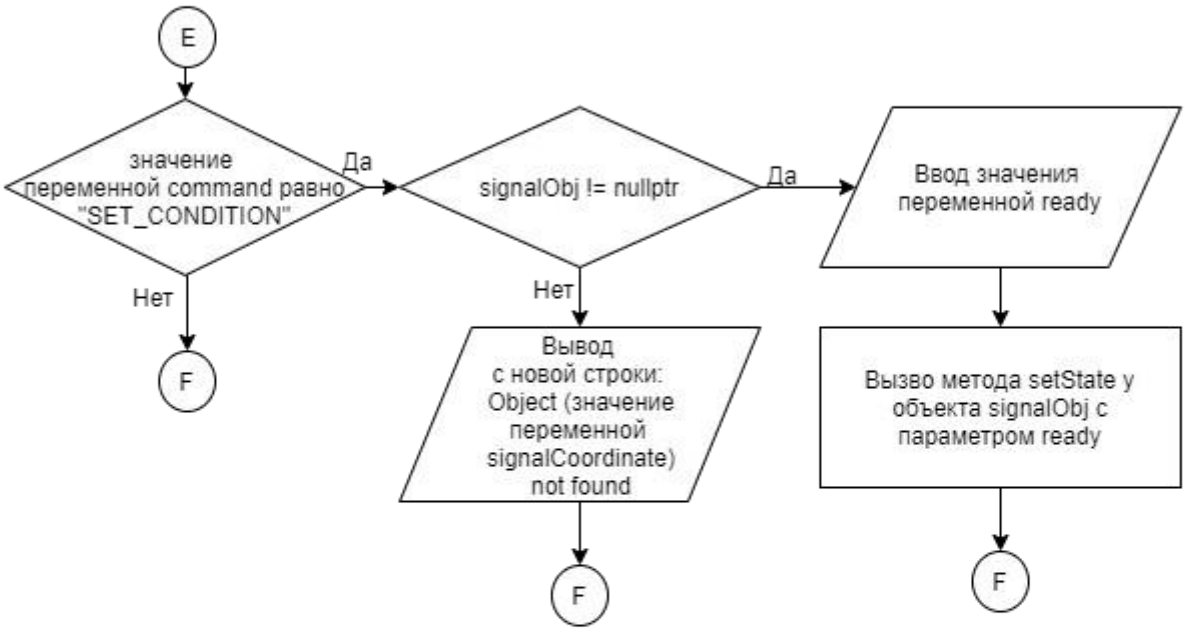

Рисунок 22 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл App.cpp

Листинг 1 – App.cpp

```
#include <iostream>
#include "App.h"

using namespace std;

App::App(Base* parent): Base(parent){
	classNumber = 1;
}

void App::buildTree(){
	string path, childName;
	cin >> path;
	setName(path);
	int classNum;
	Base* parentObject;
	while (cin >> path) {
		if (path == "endtree") break;
		parentObject = findObjByPath(path);
		if (!parentObject) {
			cout << "Object tree" << endl;
			printHierarchyDepth();
			cout << endl << "The head object " << path << " is not found";
			exit(0);
		}
		cin >> childName >> classNum;
		switch (classNum) {
			case 2:
				new Second(parentObject, childName);
				break;
			case 3:
				new Third(parentObject, childName);
				break;
			case 4:
				new Fourth(parentObject, childName);
				break;
			case 5:
				new Fifth(parentObject, childName);
				break;
			case 6:
				new Sixth(parentObject, childName);
				break;
```

```
                  default:
                        break;
            }
      }
      string signalCoordinate, handlerCoordinate;
      Base* signalObj, * handlerObj;
      cin >> signalCoordinate;
      while (signalCoordinate != "end_of_connections") {
            cin >> handlerCoordinate;
            signalObj = findObjByPath(signalCoordinate);
            if (signalObj != nullptr) {
                  handlerObj = findObjByPath(handlerCoordinate);
                  if        (handlerObj        !=        nullptr)        signalObj-
>setConnection(getSignalMethod(signalObj),                                handlerObj,
getHandlerMethod(handlerObj));
                  else cout << endl << "Handler object " << handlerCoordinate << "
not found";
            }
            else cout << endl << "Object " << signalCoordinate << " not found";
            cin >> signalCoordinate;
      }
}

int App::execute() {
      cout << "Object tree" << endl;
      printHierarchyDepth();
      setReadyForAll(1);
      string signalCoordinate, handlerCoordinate, command, text;
      Base* signalObj, *handlerObj;
      int ready;
      cin >> command;
      while (command!= "END") {
            if (command == "EMIT") {
                  cin >> signalCoordinate;
                  signalObj = findObjByPath(signalCoordinate);
                  if (signalObj != nullptr) {
                        getline(cin, text);
                        signalObj->emitSignal(getSignalMethod(signalObj), text);
                  }
                  else  cout  <<  endl  <<  "Object  "  <<  signalCoordinate  <<  "  not
found";
            }
            else if (command == "SET_CONNECT") {
                  cin >> signalCoordinate;
                  signalObj = findObjByPath(signalCoordinate);
                  cin >> handlerCoordinate;
                  if (signalObj != nullptr) {
                        handlerObj = findObjByPath(handlerCoordinate);
                        if      (handlerObj      !=      nullptr)      signalObj-
>setConnection(getSignalMethod(signalObj),                                handlerObj,
getHandlerMethod(handlerObj));
                        else   cout   <<   endl   <<   "Handler   object   "   <<
handlerCoordinate << " not found";
                  }
                  else  cout  <<  endl  <<  "Object  "  <<  signalCoordinate  <<  "  not
found";
            }
```

```
            else if (command == "DELETE_CONNECT") {
                  cin >> signalCoordinate;
                  signalObj = findObjByPath(signalCoordinate);
                  cin >> handlerCoordinate;
                  if (signalObj != nullptr) {
                        handlerObj = findObjByPath(handlerCoordinate);
                        if     (handlerObj     !=     nullptr)     signalObj     ->
deleteConnection(getSignalMethod(signalObj),                              handlerObj,
getHandlerMethod(handlerObj));
                        else   cout   <<   endl   <<   "Handler   object   "    <<
handlerCoordinate << " not found";
                  }
                  else cout << endl << "Object " << signalCoordinate << " not
found";
            }
            else if (command == "SET_CONDITION") {
                  cin >> signalCoordinate;
                  signalObj = findObjByPath(signalCoordinate);
                  if (signalObj != nullptr) {
                        cin >> ready;
                        signalObj -> setState(ready);
                  }
                  else cout << endl << "Object " << signalCoordinate << " not
found";
            }
            cin >> command;
      }
      return 0;
}

void App::signalMethod(string& text) {
      cout << endl << "Signal from " << getAbsoluteCoordinate();
      text += " (class: 1)";
}

void App::handlerMethod(string text) {
      cout << endl << "Signal to " << getAbsoluteCoordinate() << " Text: " <<
text;
}

Base::TYPE_SIGNAL App::getSignalMethod(Base* pointer) {
      if (pointer->getClassNumber() == 1) return SIGNAL_D(App::signalMethod);
      else       if       (pointer->getClassNumber()       ==       2)       return
SIGNAL_D(Second::signalMethod);
      else       if       (pointer->getClassNumber()       ==       3)       return
SIGNAL_D(Third::signalMethod);
      else       if       (pointer->getClassNumber()       ==       4)       return
SIGNAL_D(Fourth::signalMethod);
      else       if       (pointer->getClassNumber()       ==       5)       return
SIGNAL_D(Fifth::signalMethod);
      else return SIGNAL_D(Sixth::signalMethod);
}

Base::TYPE_HANDLER App::getHandlerMethod(Base* pointer) {
      if (pointer->getClassNumber() == 1) return HANDLER_D(App::handlerMethod);
      else       if       (pointer->getClassNumber()       ==       2)       return
HANDLER_D(Second::handlerMethod);
```

```
	else	if	(pointer->getClassNumber()	==	3)	return
HANDLER_D(Third::handlerMethod);
	else	if	(pointer->getClassNumber()	==	4)	return
HANDLER_D(Fourth::handlerMethod);
	else	if	(pointer->getClassNumber()	==	5)	return
HANDLER_D(Fifth::handlerMethod);
	else return HANDLER_D(Sixth::handlerMethod);
}
```

5.2 Файл App.h

Листинг 2 – App.h

```
#ifndef___APP_H
#define___APP_H
#include "Base.h"
#include "Second.h"
#include "Third.h"
#include "Fourth.h"
#include "Fifth.h"
#include "Sixth.h"

class App : public Base{
public:
	App(Base* parent);
	void buildTree();
	int execute();
	void signalMethod(string& text);
	void handlerMethod(string text);
	Base::TYPE_SIGNAL getSignalMethod(Base* pointer);
	Base::TYPE_HANDLER getHandlerMethod(Base* pointer);
};
#endif
```

5.3 Файл Base.cpp

Листинг 3 – Base.cpp

```
#include <iostream>
#include "Base.h"

using namespace std;

Base::Base(Base * parent, string name) {
	this->name = name;
	parentObj = parent;
	if (parentObj) parentObj->children.push_back(this);
}
```

```
void Base::setName(string name) {
	this->name = name;
}

string Base::getName() {
	return name;
}

void Base::printHierarchy() {
	if (!parentObj) cout << name;
	if (children.size()) {
		cout << endl << name;
		for (int i = 0; i < children.size(); i++) {
			cout << "  " << children[i]->getName();
		}
		children[children.size()-1]->printHierarchy();
	}
}

void Base::printHierarchyDepth(int depth) {
	cout << name;
	if (children.size()) {
		depth++;
		for (int i = 0; i < children.size(); i++) {
			cout << endl;
			for (int j = 0; j < depth; j++) cout << "    ";
			children[i]->printHierarchyDepth(depth);
		}
	}
}

void Base::printHierarchyDepthState(int depth) {
	cout << name;
	cout << ((state == 0) ? " is not ready" : " is ready");
	if (children.size()) {
		depth++;
		for (int i = 0; i < children.size(); i++) {
			cout << endl;
			for (int j = 0; j < depth; j++) cout << "    ";
			children[i]->printHierarchyDepthState(depth);
		}
	}
}

void Base::setParent(Base* newParent) {
	if (parentObj && newParent) {
		for(int i = 0;  i < parentObj->children.size(); i++) {
			if (parentObj->children[i] == this) {
				parentObj->children.erase(parentObj->children.begin()+i);
				break;
			}
		}
		parentObj = newParent;
		parentObj->children.push_back(this);
	}
}
```

```
Base* Base::getParent() {
	return parentObj;
}

Base::~Base() {
	for (int i = 0; i < children.size(); i++) delete children[i];
}

int Base::getState() {
	return state;
}

void Base::setState(int state) {
	if (parentObj && parentObj->getState() == 0) return;
	else {
		if (state == 0) {
			for (int i = 0; i < children.size(); i++) children[i]-
>setState(0);
		}
		this->state = state;
	}
}

Base* Base::findObjByName(string name) {
	Base* currObj;
	if (this->name == name) return this;
	for (int i = 0; i < children.size(); i++) {
		////////////////////////////////////////////////////////////////////
////
		currObj = children[i]->findObjByName(name);
		if (currObj) return currObj;
	}
	return nullptr;
}

Base* Base::findObjByPath(string path) {
	if (path == ".") return this;
	Base* currObj = this, *root = this;
	while (root->getParent()) root = root-> getParent();
	if (path == "/") {
		return root;
	}
	if (path[0] == '/' && path[1] == '/') {
		return root->findObjByName(path.substr(2));
	}
	if (path[0] == '/')
		path = path.substr(1);

	Base* foundObj;
	string name = "";
	for (int i = 0; i <= path.length(); i++) {
		if (path[i] == '/' || i == path.length()) {
			foundObj = nullptr;
			for (int j = 0; j < currObj->children.size(); j++) {
				if (currObj->children[j]->getName() == name) {
					foundObj = currObj->children[j];
					break;
```

```
                        }
                    }
                    currObj = foundObj;
                    if (!currObj) return nullptr;
                    name = "";
                }
                else name += path[i];
        }
        return currObj;
}

void Base::setConnection(TYPE_SIGNAL signalMethod, Base* handler, TYPE_HANDLER
handlerMethod) {
        for (int i = 0; i < connections.size(); i++) {
                if (connections[i].signalMethod == signalMethod &&
connections[i].handlerMethod == handlerMethod) return;
        }
        connectionStruct connection;
        connection.signalMethod = signalMethod;
        connection.handler = handler;
        connection.handlerMethod = handlerMethod;
        connections.push_back(connection);
}

void Base::deleteConnection(TYPE_SIGNAL signalMethod, Base* handler, TYPE_HANDLER
handlerMethod) {
        for (int i = 0; i < connections.size(); i++) {
                if (connections[i].signalMethod == signalMethod &&
connections[i].handlerMethod == handlerMethod) {
                    connections.erase(connections.begin()+i);
                    return;
                }
        }
}

void Base::emitSignal(TYPE_SIGNAL signalMethod, string& text) {
        if (getState() == 0) return;
        if (connections.empty()) return;
        (this->*signalMethod)(text);
        for (int i = 0; i < connections.size(); i++) {
                if (connections[i].signalMethod == signalMethod &&
connections[i].handler->getState() != 0) {
                    (connections[i].handler->*(connections[i].handlerMethod))(text);
                }
        }
}

string Base::getAbsoluteCoordinate(string absoluteCoordinate) {
        if (parentObj == nullptr) {
                return "/";
        }
        return parentObj->getAbsoluteCoordinate("/") + name + absoluteCoordinate;
}

int Base::getClassNumber() {
        return classNumber;
}
```

```
void Base::setReadyForAll(int state) {
	setState(state);
	for (int i = 0; i < children.size(); i++) {
		children[i]->setReadyForAll(state);
	}
}
```

5.4 Файл Base.h

Листинг 4 – Base.h

```
#ifndef___BASE_H
#define___BASE_H

#define SIGNAL_D(signal_f) (TYPE_SIGNAL) (&signal_f)
#define HANDLER_D(handler_f) (TYPE_HANDLER) (&handler_f)

#include <iostream>
#include <string>
#include <vector>

using namespace std;

class Base {
public:
	//////////
	typedef void (Base::* TYPE_SIGNAL) (string&);
	typedef void (Base::* TYPE_HANDLER) (string);
	//////////
private:
	string name;
	Base * parentObj = nullptr;
	vector <Base*> children;
	int state = 0;
	struct connectionStruct {
		TYPE_SIGNAL signalMethod;
		Base* handler;
		TYPE_HANDLER handlerMethod;
	};
	vector <connectionStruct> connections;
protected:
	int classNumber;
public:
	Base(Base * parent, string name = "");
	void setName(string name);
	string getName();
	void setParent(Base* newParent);
	void printHierarchy();
	Base* getParent();
	~Base();
	int getState();
	void setState(int state);
```

```
	Base* findObjByPath(string path);
	void printHierarchyDepth(int depth = 0);
	void printHierarchyDepthState(int depth = 0);
	Base* findObjByName(string name);
	void setConnection(TYPE_SIGNAL signalMethod, Base* handler, TYPE_HANDLER
handlerMethod);
	void deleteConnection(TYPE_SIGNAL signalMethod, Base* handler, TYPE_HANDLER
handlerMethod);
	void emitSignal(TYPE_SIGNAL signalMethod, string& text);
	int getClassNumber();
	void setReadyForAll(int state);
	string getAbsoluteCoordinate(string absoluteCoordinate = "");
};

#endif
```

5.5 Файл Fifth.cpp

Листинг 5 – Fifth.cpp

```
#include "Fifth.h"

Fifth::Fifth(Base* parent, string name) : Base(parent, name) {
	classNumber = 5;
}

void Fifth::signalMethod(string& text) {
	cout << endl << "Signal from " << getAbsoluteCoordinate();
	text += " (class: 5)";
}

void Fifth::handlerMethod(string text) {
	cout << endl << "Signal to " << getAbsoluteCoordinate() << " Text: " <<
text;
}
```

5.6 Файл Fifth.h

Листинг 6 – Fifth.h

```
#ifndef___FIFTH_H
#define___FIFTH_H
#include "Base.h"

class Fifth : public Base{
public:
	Fifth(Base* parent, string name = "");
	void signalMethod(string& text);
	void handlerMethod(string text);
```

```
};
#endif
```

5.7 Файл Fourth.cpp

Листинг 7 – Fourth.cpp

```
#include "Fourth.h"

Fourth::Fourth(Base* parent, string name) : Base(parent, name) {
	classNumber = 4;
}

void Fourth::signalMethod(string& text) {
	cout << endl << "Signal from " << getAbsoluteCoordinate();
	text += " (class: 4)";
}

void Fourth::handlerMethod(string text) {
	cout << endl << "Signal to " << getAbsoluteCoordinate() << " Text: " <<
text;
}
```

5.8 Файл Fourth.h

Листинг 8 – Fourth.h

```
#ifndef___FOURTH_H
#define___FOURTH_H
#include "Base.h"

class Fourth : public Base{
public:
	Fourth(Base* parent, string name = "");
	void signalMethod(string& text);
	void handlerMethod(string text);
};
#endif
```

5.9 Файл main.cpp

Листинг 9 – main.cpp

```
#include "App.h"
```

```
int main() {
	App AppObj(nullptr);
	AppObj.buildTree();
	return AppObj.execute();
}
```

5.10 Файл Second.cpp

Листинг 10 – Second.cpp

```
#include "Second.h"

Second::Second(Base* parent, string name) : Base(parent, name) {
	classNumber = 2;
}

void Second::signalMethod(string& text) {
	cout << endl << "Signal from " << getAbsoluteCoordinate();
	text += " (class: 2)";
}

void Second::handlerMethod(string text) {
	cout << endl << "Signal to " << getAbsoluteCoordinate() << " Text: " <<
text;
}
```

5.11 Файл Second.h

Листинг 11 – Second.h

```
#ifndef___SECOND_H
#define___SECOND_H
#include "Base.h"

class Second : public Base{
public:
	Second(Base* parent, string name = "");
	void signalMethod(string& text);
	void handlerMethod(string text);
};
#endif
```

5.12 Файл Sixth.cpp

Листинг 12 – Sixth.cpp

```
#include "Sixth.h"

Sixth::Sixth(Base* parent, string name) : Base(parent, name) {
	classNumber = 6;
}

void Sixth::signalMethod(string& text) {
	cout << endl << "Signal from " << getAbsoluteCoordinate();
	text += " (class: 6)";
}

void Sixth::handlerMethod(string text) {
	cout << endl << "Signal to " << getAbsoluteCoordinate() << " Text: " <<
text;
}
```

5.13 Файл Sixth.h

Листинг 13 – Sixth.h

```
#ifndef___SIXTH_H
#define___SIXTH_H
#include "Base.h"

class Sixth : public Base{
public:
	Sixth(Base* parent, string name = "");
	void signalMethod(string& text);
	void handlerMethod(string text);
};
#endif
```

5.14 Файл Third.cpp

Листинг 14 – Third.cpp

```
#include "Third.h"

Third::Third(Base* parent, string name) : Base(parent, name) {
	classNumber = 3;
}

void Third::signalMethod(string& text) {
```

```
	cout << endl << "Signal from " << getAbsoluteCoordinate();
	text += " (class: 3)";
}

void Third::handlerMethod(string text) {
	cout << endl << "Signal to " << getAbsoluteCoordinate() << " Text: " <<
text;
}
```

5.15 Файл Third.h

Листинг 15 – Third.h

```
#ifndef___THIRD_H
#define___THIRD_H
#include "Base.h"

class Third : public Base{
public:
	Third(Base* parent, string name = "");
	void signalMethod(string& text);
	void handlerMethod(string text);
};
#endif
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 31.

Таблица 31 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
appls_root / object_s1 3 / object_s2 2 /object_s2 object_s4 4 / object_s13 5 /object_s2 object_s6 6 /object_s1 object_s7 2 endtree /object_s2/object_s4 /object_s2/object_s6 /object_s2 /object_s1/object_s7 / /object_s2/object_s4 /object_s2/object_s4 / end_of_connections EMIT /object_s2/object_s4 Send message 1 DELETE_CONNECT /object_s2/object_s4 / EMIT /object_s2/object_s4 Send message 2 SET_CONDITION /object_s2/object_s4 0 EMIT /object_s2/object_s4 Send message 3 SET_CONNECT /object_s1 /object_s2/object_s6 EMIT /object_s1 Send message 4 END	Object tree appls_root object_s1 object_s7 object_s2 object_s4 object_s6 object_s13 Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 1 (class: 4) Signal to / Text: Send message 1 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 2 (class: 4) Signal from /object_s1 Signal to /object_s2/object_s6 Text: Send message 4 (class: 3)	Object tree appls_root object_s1 object_s7 object_s2 object_s4 object_s6 object_s13 Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 1 (class: 4) Signal to / Text: Send message 1 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 2 (class: 4) Signal from /object_s1 Signal to /object_s2/object_s6 Text: Send message 4 (class: 3)

ЗАКЛЮЧЕНИЕ

При помощи полученных за курс Объектно ориентированного программирования знаний реализовать поставленную задачу. Провести необходимые тестирования кода и убедиться в его работоспособности.

За пройденный курс Объектно ориентированного программирования приобрел такие знания, как:

понимание понятия "класс" и "объект", обрел базовые умения работы с объектами и классами [2];

узнал про перегрузку операторов, получил необходимые навыки работы с перегрузкой операторов [2];

узнал основные парадигмы Объектно Ориентированного программирования [2];

приобрел умение работать с дружественными функциями [2].

Также в ходе выполнения курсовой работы был получен незаменимый опыт проектирования системы и работы с документацией, а именно с техническим заданием.

Поставленная задача была успешно решена. Был описан метод решения, написан алгоритм всех необходимых функций и методов, отрисованы блок-схемы по всем описанным в алгоритме методам и функциям.

Разработка программного продукта проходила в учебно-технологической среде "АСО Avrora". Следует отметить, что данная среда разработки сильно упрощает процесс оформления работ, а именно:

3. "АСО Avrora" имеет удобный функционал для построения алгоритмов решаемой задачи;
4. "АСО Avrora" предоставляет возможность генерации блок-схем по алгоритму;
5. "АСО Avrora" позволяет проводить тестирование с автоматическим

сравнением выводимых программой данных с ожидаемыми;

6. "АСО Avrora" работает круглосуточно;
7. "АСО Avrora" имеет возможность автоматической генерации отчета;
8. В "АСО Avrora" реализована система котроля версий (избавляет от необходимости тратить время на перенесение программы на другие носители для работы на разных устройствах).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Васильев А.Н. Объектно-ориентированное программирование на C++. Издательство: Наука и Техника. Санкт-Петербург, 2016г. 543 стр.

2. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2017. — 624 с.

3. Методическое пособие для проведения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratorny h_rabot_3.pdf (дата обращения 05.05.2021).

4. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» [Электронный ресурс]. URL: https://mirea.aco-avrora.ru/student/files/Prilozheniye_k_methodichke.pdf (дата обращения 05.05.2021).

5. Видео лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».

6. Антик М.И. Дискретная математика [Электронный ресурс]: Учебное пособие /Антик М.И., Казанцева Л.В. — М.: МИРЭА — Российский технологический университет, 2018 — 1 электрон. опт. диск (CD-ROM).