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 задачи:

« КЛ_3_3 Сигналы и обработчики »

С тудент группы	ИКБО-13-21	Черномуров С.А.
Руководитель практики	Ассистент	Асадова Ю.С.
Работа представлена	«___»_____2021 г.	_____
		(подпись студента)
Оценка		_____
		(подпись руководителя)

Москва 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....

Постановка задачи.....

Метод решения.....

Описание алгоритма.....

Блок-схема алгоритма.....

Код программы.....

Тестирование.....

ЗАКЛЮЧЕНИЕ.....

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ (ИСТОЧНИКОВ).....

ВВЕДЕНИЕ

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.

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
```

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)

Метод решения

Для решения задачи используются:

- Всё, что использовалось в предыдущей версии приложения КЛ_3_2.
- Параметризированное макроопределение препроцессора. Используется для получения указателей на методы сигнала и обработчика объекта.
- **Классы Application:**
    - Методы:
        - Метод BuildTree доработан на основе предыдущей версии. Теперь в нём дополнительно реализована установка связей между объектами.
        - Метод StartApp доработан на основе предыдущей версии. Теперь в нём дополнительно реализована обработка пользовательских команд.
        - Метод Signal:
            - Функционал - параметризированный метод сигнала.
        - Метод Handler:
            - Функционал - параметризированный метод обработчика.
- **Класс Base:**
    - Свойства/поля:
        - Поле:
            - Наименование - class_number;
            - Тип - целочисленный;
            - Модификатор доступа - закрытый.
        - Поле:
            - Наименование - o_sh;
            - Тип - структура данных (содержит указатель на

сигнал, указатель на объект, с которым устанавливается связь, указатель на обработчик);
- Модификатор доступа - открытый.

- Поле:
  - Наименование - connects;
  - Тип - контейнер класса vector, содержащий указатели на объекты типа o_sh)
  - Модификатор доступа - открытый.

- Методы:
  - Метод Base (конструктор) доработан на основе предыдущей версии приложения. В него дополнительно передается номер класса, который присваивается полю class_number.
  - Метод SetConnect:
    - Функционал - параметризированный метод установки связи между сигналом текущего объекта и обработчиком целевого объекта.
  - Метод DeleteConnect:
    - Функционал - параметризированный метод удаления (разрыва) свзяи между сигналом текущего объекта и обработчиком целевого объекта.
  - Метод EmitSignal:
    - Функционал - параметризированный метод выдачи сигнала от текущего объекта с передачей строковой переменной.
  - Метод GetCoordinates:
    - Функционал - метод получения абсолютного пути текущего объекта.

        - Метод GetClassNumber:
            - Функционал - метод получения номера класса объекта.
- **Классы Child2, Child3, Child4, Child5, Child6:**
    - Свойства/поля:
        - Унаследованы из класса Base.
    - Методы:
        - Унаследованы из класса Base.
        - Метод Signal:
            - Функционал - параметризированный метод сигнала.
        - Метод Handler:
            - Функционал - параметризированный метод обработчика.

Иерархия наследования классов:

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер	Комментарий
1	Base			Базовый класс в иерархии классов. Содержит основные поля и методы		
		Application	public		2	
		Child2	public		3	
		Child3	public		4	
		Child4	public		5	
		Child5	public		6	
		Child6	public		7	

2	Application			Класс корневого объекта (приложения)		
3	Child2			Класс объектов, подчиненных корневому объекта класса Application		
4	Child3			Класс объектов, подчиненных корневому объекта класса Application		
5	Child4			Класс объектов, подчиненных корневому объекта класса Application		
6	Child5			Класс объектов, подчиненных корневому объекта класса Application		
7	Child6			Класс объектов, подчиненных корневому объекта класса		

				Application		

Описание алгоритма

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

Функция: main

Функционал: Основной алгоритм программы

Параметры: Отсутствуют

Возвращаемое значение: Целочисленный тип данных - код возврата

Алгоритм функции представлен в таблице 2.

Таблица 2. Алгоритм функции main

№	Предикат	Действия	№ перехода	Комментарий
1		Создание объекта app класса Application путем вызова параметризированного конструктора с параметрами nullptr, "", 1	2	
2		Вызов метода BuildTree объекта app	3	
3		Возврат функцией значения возвращенного методом StartApp объекта app	∅	

Класс объекта: Child2

Модификатор доступа: public

Метод: Signal

Функционал: Параметризированный метод сигнала

Параметры: Ссылка на строковую переменную s

Возвращаемое значение: Отсутствует

Алгоритм метода представлен в таблице 3.

Таблица 3. Алгоритм метода Signal класса Child2

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод на экран "\nSignal from ", Значение, возвращенное методом GetCoordinates	2	
2		Присвоение s значения s+" (class: 2)"	∅	

Класс объекта: Child3

Модификатор доступа: public

Метод: Signal

Функционал: Параметризированный метод сигнала

Параметры: Ссылка на строковую переменную s

Возвращаемое значение: Отсутствует

Алгоритм метода представлен в таблице 4.

Таблица 4. Алгоритм метода Signal класса Child3

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод на экран "\nSignal from ", Значение, возвращенное методом GetCoordinates	2	
2		Присвоение s значения s+" (class: 3)"	∅	

Класс объекта: Child4

Модификатор доступа: public

Метод: Signal

Функционал: Параметризированный метод сигнала

Параметры: Ссылка на строковую переменную s

Возвращаемое значение: Отсутствует

Алгоритм метода представлен в таблице 5.

Таблица 5. Алгоритм метода Signal класса Child4

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод на экран "\nSignal from ", Значение, возвращенное методом GetCoordinates	2	
2		Присвоение s значения s+" (class: 4)"	∅	

Класс объекта: Child5

Модификатор доступа: public

Метод: Signal

Функционал: Параметризированный метод сигнала

Параметры: Ссылка на строковую переменную s

Возвращаемое значение: Отсутствует

Алгоритм метода представлен в таблице 6.

Таблица 6. Алгоритм метода Signal класса Child5

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод на экран "\nSignal from ", Значение, возвращенное методом GetCoordinates	2	
2		Присвоение s значения s+" (class: 5)"	∅	

Класс объекта: Child6

Модификатор доступа: public

Метод: Signal

Функционал: Параметризированный метод сигнала

Параметры: Ссылка на строковую переменную s

Возвращаемое значение: Отсутствует

Алгоритм метода представлен в таблице 7.

Таблица 7. Алгоритм метода Signal класса Child6

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод на экран "\nSignal from ", Значение, возвращенное методом GetCoordinates	2	
2		Присвоение s значения s+" (class: 6)"	∅	

Класс объекта: Application

Модификатор доступа: public

Метод: Signal

Функционал: Параметризированный метод сигнала

Параметры: Ссылка на строковую переменную s

Возвращаемое значение: Отсутствует

Алгоритм метода представлен в таблице 8.

Таблица 8. Алгоритм метода Signal класса Application

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод на экран "\nSignal from ", Значение, возвращенное методом GetCoordinates	2	
2		Присвоение s значения s+" (class: 1)"	∅	

Класс объекта: Child2

Модификатор доступа: public

Метод: Handler

Функционал: Параметризированный метод обработчика

Параметры: Строковый параметр s

Возвращаемое значение: Отсутствует

Алгоритм метода представлен в таблице 9.

Таблица 9. Алгоритм метода Handler класса Child2

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод на экран "\nSignal to ", значение, возвращенное методом GetCoordinates, " Text: ", s	∅	

Класс объекта: Child3

Модификатор доступа: public

Метод: Handler

Функционал: Параметризированный метод обработчика

Параметры: Строковый параметр s

Возвращаемое значение: Отсутствует

Алгоритм метода представлен в таблице 10.

Таблица 10. Алгоритм метода Handler класса Child3

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод на экран "\nSignal to ", значение, возвращенное методом GetCoordinates, " Text: ", s	∅	

Класс объекта: Child4

Модификатор доступа: public

Метод: Handler

Функционал: Параметризированный метод обработчика

Параметры: Строковый параметр s

Возвращаемое значение: Отсутствует

Алгоритм метода представлен в таблице 11.

Таблица 11. Алгоритм метода Handler класса Child4

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод на экран "\nSignal to ",	∅	

		значение, возвращенное методом GetCoordinates, " Text: ", s		

Класс объекта: Child5

Модификатор доступа: public

Метод: Handler

Функционал: Параметризированный метод обработчика

Параметры: Строковый параметр s

Возвращаемое значение: Отсутствует

Алгоритм метода представлен в таблице 12.

Таблица 12. Алгоритм метода Handler класса Child5

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод на экран "\nSignal to ", значение, возвращенное методом GetCoordinates, " Text: ", s	∅	

Класс объекта: Child6

Модификатор доступа: public

Метод: Handler

Функционал: Параметризированный метод обработчика

Параметры: Строковый параметр s

Возвращаемое значение: Отсутствует

Алгоритм метода представлен в таблице 13.

Таблица 13. Алгоритм метода Handler класса Child6

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод на экран "\nSignal to ", значение, возвращенное методом GetCoordinates, " Text: ", s	∅	

Класс объекта: Application

Модификатор доступа: public

Метод: Handler

Функционал: Параметризированный метод обработчика

Параметры: Строковый параметр s

Возвращаемое значение: Отсутствует

Алгоритм метода представлен в таблице 14.

Таблица 14. Алгоритм метода Handler класса Application

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод на экран "\nSignal to ", значение, возвращенное методом GetCoordinates, " Text: ", s	∅	

Конструктор класса: Base

Модификатор доступа: public

Функционал: Параметризированный конструктор, содержащий указатель на головной объект в дереве иерархии, наименование объекта, номер класса объекта

Параметры: Указатель на объект класса Base - parent, Строковый параметр name, целочисленный параметр class_number

Алгоритм конструктора представлен в таблице 15.

Таблица 15. Алгоритм конструктора класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Вызов метода SetName с параметром name	2	
2		Вызов метода SetParent с параметром parent	3	
3	Значение parent не равно nullptr	Вызов метода push_back с параметром: указатель на текущий объект; свойства children объекта parent	4	
			4	
4		Присвоение свойству class_number текущего объекта значения class_number	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: SetConnect

Функционал: Параметризированный метод установки связи между сигналом текущего объекта и обработчиком целевого объекта

Параметры: Указатель на метод сигнала signal, Указатель connected_obj на объект класса Base, Указатель на метод обработчика handler

Возвращаемое значение: Отсутствует

Алгоритм метода представлен в таблице 16.

Таблица 16. Алгоритм метода SetConnect класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление указателя на объект структуры o_sh - p_value	2	
2		Объявление целочисленной переменной с инициализацией i=0	3	Использование i в качестве
3	Значение i меньше значения, возвращенного методом size объекта connects		4	
			6	Выход из цикла
4	Поля объекта connects[i] равны signal, connected_obj и handler		∅	
			5	
5		Инкрементирование i	3	
6		Создание динамической структуры p_value типа o_sh	7	
7		Присвоение полям p_value значений signal, connected_obj, handler	8	
8		Вызов метода push_back объекта connects с параметром p_value	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: DeleteConnect

Функционал: Параметризированный метод удаления (разрыва) связяи между сигналом текущего объекта и обработчиком целевого объекта

Параметры: Указатель на метод сигнала signal, Указатель connected_obj на объект класса Base, Указатель на метод обработчика handler

Возвращаемое значение: Отсутствует

Алгоритм метода представлен в таблице 17.

Таблица 17. Алгоритм метода DeleteConnect класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление целочисленной переменной с инициализацией i=0	2	Использование i в качестве счетчика
2	Значение i меньше значения, возвращенного методом size объекта connects		3	
			∅	Выход из цикла
3	Поля объекта connects[i] равны signal, connected_obj и handler	Вызов метода erase объекта connects с параметром connects.begin()+i	∅	
			4	
4		Инкрементирование i	2	

Класс объекта: Base

Модификатор доступа: public

Метод: EmitSignal

Функционал: Параметризированный метод выдачи сигнала от текущего объекта с передачей строковой переменной

Параметры: Указатель на метод сигнала signal, ссылка на строковую переменную message

Возвращаемое значение: Отсутствует

Алгоритм метода представлен в таблице 18.

Таблица 18. Алгоритм метода EmitSignal класса Base

№	Предикат	Действия	№ перехода	Комментарий
1	Значение status не равно нулю	Вызов метода signal текущего объекта по указателю с параметром message	2	
			∅	
2		Объявление целочисленной переменной с инициализацией i=0	3	Использование i в качестве счетчика
3	Значение i меньше значения возвращенного методом size объекта connects		4	
			∅	Выход из цикла
4	Значения поля signal объекта connects[i] равно signal и значения поля status поля connected_obj объекта connects[i] не равно нулю	Вызов метода handler поля connects[i] объекта connected_obj по указателю с параметром message	5	

			5	
5		Инкрементирование i	3	

Класс объекта: Base

Модификатор доступа: public

Метод: GetCoordinates

Функционал: Метод получения абсолютного пути текущего объекта

Параметры: Отсутствуют

Возвращаемое значение: Строковый тип данных - значение path

Алгоритм метода представлен в таблице 19.

Таблица 19. Алгоритм метода GetCoordinates класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление указателя на объект класса Base с инициализацией obj=указатель на текущий объект	2	
2		Объявление строковой переменной с инициализацией path=""	3	
3	Значение возвращенное методом GetHead объекта obj не равно nullptr	Присвоение path значения "/"+значение, возвращенное методом GetName объекта obj+path	4	
			5	Выход из цикла
4		Присвоение obj значения	3	

		возвращенного методом GetHead объекта obj		
5	Значение, возвращенное методом size объекта path равно нулю	Возврат методом "/"	∅	
		Возврат методом path	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: GetClassNumber

Функционал: Метод получения номера класса объекта

Параметры: Отсутствуют

Возвращаемое значение: Целочисленный тип данных - значения поля class_number

Алгоритм метода представлен в таблице 20.

Таблица 20. Алгоритм метода GetClassNumber класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Возврат методом значения class_number	∅	

Класс объекта: Application

Модификатор доступа: public

Метод: BuildTree

Функционал: Метод постройки дерева иерархии объектов и установки связей между ними

Параметры: Отсутствуют

Возвращаемое значение: Отсутствует

Алгоритм метода представлен в таблице 21.

Таблица 21. Алгоритм метода BuildTree класса Application

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление строковой переменной name	2	
2		Считывание с клавиатуры значения name	3	
3		Вызов метода SetName текущего объекта	4	
4		Объявление указателя на объект класса Base с инициализацией ob_base=указатель на текущий объект	5	
5		Объявление указателей на динамические объекты классов Child2-Child6 с инициализацией child2-child6=nullptr	6	
6		Вызов метода push_back объекта ukaz с параметром ob_base	7	
7		Объявление строковых переменных parentname, childname	8	
8		Вывод на экран "Object tree\n", значение, возвращенное методом GetName текущего объекта	9	
9		Считывание с клавиатуры значения parentname	10	

10	Значение parentname не равно "endtree"	Считывание с клавиатуры значений childname, child_number	11	
			11	
11	Значение parentname не равно "endtree"	Присвоение ob_base значения возвращенного методом ObjectByName с параметрами parentname,ukaz, указатель на текущий объект	12	
			18	Выход из цикла
12	Значение ob_base равно nullptr	Вызов метода PrintTree	13	
			14	
13		Вывод на экран "\nThe head object", parentname, " is not found"	∅	
14	Значение child_number равно (2,3,4,5,6)	Создание динамического объекта child2-child6 путем вызова параметризированного конструктора с параметрами ob_base, childname, (2,3,4,5,6)	15	
			16	
15		Вызов метода push_back объекта ukaz с параметром child2-child6	16	
16		Считывание с клавиатуры значения parentname	17	
17	Значение parentname не равно "endtree"	Считывание с клавиатуры значений childname, child_number	11	
			11	
18		Объявление массива типа данных TYPE_SIGNAL с	19	

		инициализацией signals[]={указатели на методы signal классов Application, Child2-Child6}		
19		Объявление массива типа данных TYPE_HANDLER с инициализацией handlers[]={указатели на методы handler классов Application, Child2-Child6}	20	
20	Логическое значение "Истина"	Объявление строковых переменных sender, receiver, message	21	
			∅	Выход из цикла
21		Считывание с клавиатуры значения sender	22	
22	Значение sender равно "end_of_connections"		∅	Принудительное завершение цикла
			23	
23		Считывание с клавиатуры значения receiver	24	
24		Объявление указателя на объект класса Base с инициализацией obj_sender=значение возвращенное методом ObjectByName с параметрами sender, ukaz, указатель на текущий объект	25	
25		Объявление указателя на объект класса Base с инициализацией obj_receiver=значение возвращенное методом ObjectByName с	26	

		параметрами receiver, ukaz, указатель на текущий объект		
26		Вызов метода SetConnect объекта obj_sender с параметрами signals[декрементированное значение, возвращенное методом GetClassNumber объекта obj_sender], obj_receiver, handlers[декрементированное значение, возвращенное методом GetClassNumber объекта obj_receiver]	20	

Класс объекта: Application

Модификатор доступа: public

Метод: StartApp

Функционал: Метод запуска приложения и обработки пользовательских команд

Параметры: Отсутствуют

Возвращаемое значение: Целочисленный тип данных - код возврата

Алгоритм метода представлен в таблице 22.

Таблица 22. Алгоритм метода StartApp класса Application

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление массива типа данных TYPE_SIGNAL с инициализацией signals[]={указатели на методы signal классов Application, Child2-Child6}	2	

2		Объявление массива типа данных TYPE_HANDLER с инициализацией handlers[]={указатели на методы handler классов Application, Child2-Child6}	3	
3		Вызов метода PrintTree текущего объекта	4	
4		Приведение всех объектов в состояние готовности	5	
5		Объявление строковых переменных command, objectname, message	6	
6	Значение command считано с клавиатуры		7	
			∅	Выход из цикла
7	Значение command равно "END"		∅	
			8	
8		Считывание с клавиатуры значения objectname	9	
9		Объявления указателя на объект класса Base с инициализацией obj=значение, возвращенное методом ObjectByName с параметрами objectname, ukaz, указатель на текущий объект	10	
10	Значение command равно "EMIT"	Считывание с клавиатуры значения message	11	
			12	
11	Значение obj	Вызов метода EmitSignal	6	

	не равно nullptr	объекта obj с параметрами signals[декрементированное значение возвращенное методом GetClassNumber объекта obj], message		
		Вывод на экран "\nObject ",objectname," not found"	6	
12	Значение command равно "SET_CONNECT"		13	
			16	
13	Значение obj равно nullptr	Вывод на экран "\nObject ", objectname," not found"	6	
			14	
14	Значение connected_obj равно nullptr	Вывод на экран "\nHandler object ", object2name," not found"	6	
			15	
15		Вызов метода SetConnect объекта obj с параметрами signals[декрементированное значение, возвращенное методом GetClassNumber объекта obj], connected_obj, handlers[декрементированное значение возвращенное методом GetClassNumber объекта connected_obj]	6	
16	Значение command равно DELETE_CONNECT		17	
			6	
17	Значение obj равно nullptr	Вывод на экран "\nObject ", objectname," not found"	6	
			18	

18	Значение connected_obj равно nullptr	Вывод на экран "\nHandler object ",object2name," not found"	6	
			19	
19		Вызов метода DeleteConnect объекта obj с параметрами signals[декрементированное значение возвращенное методом GetClassNumber объекта obj], connected_obj, handlers[декрементированное значение возвращенное методом GetClassNumber объекта connected_obj]	6	

Блок-схема алгоритма

Представим описание алгоритмов в графическом виде на рисунках ниже.

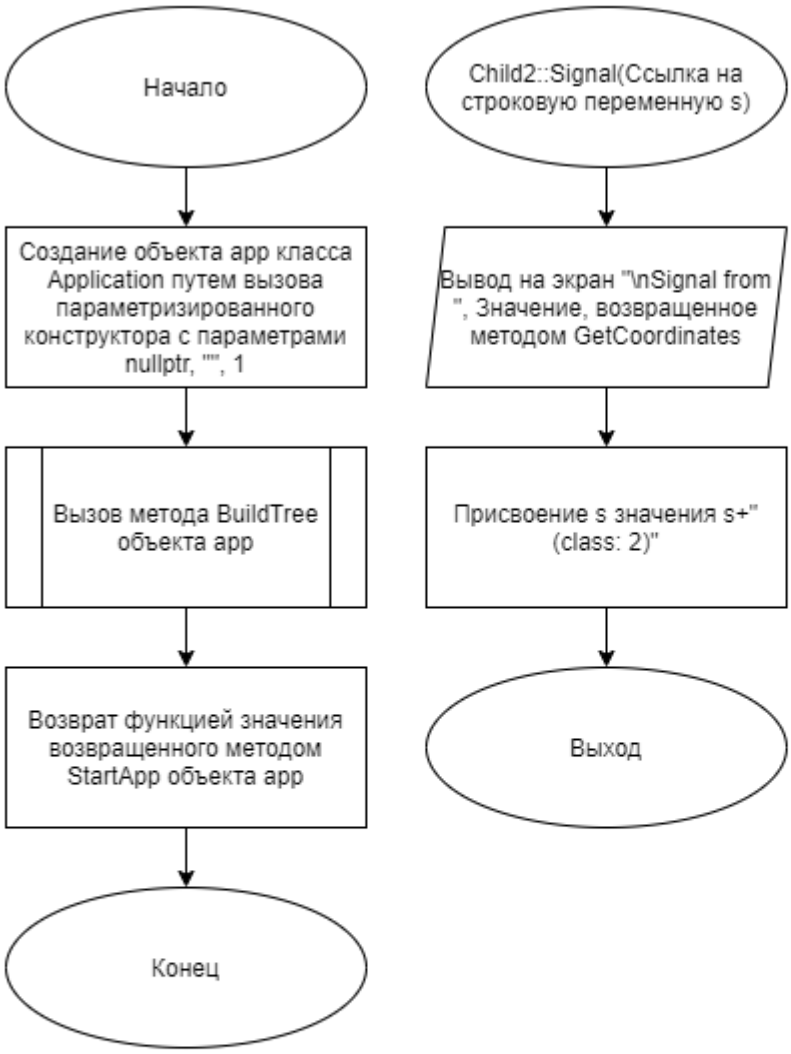

Рис. 1. Блок-схема алгоритма.

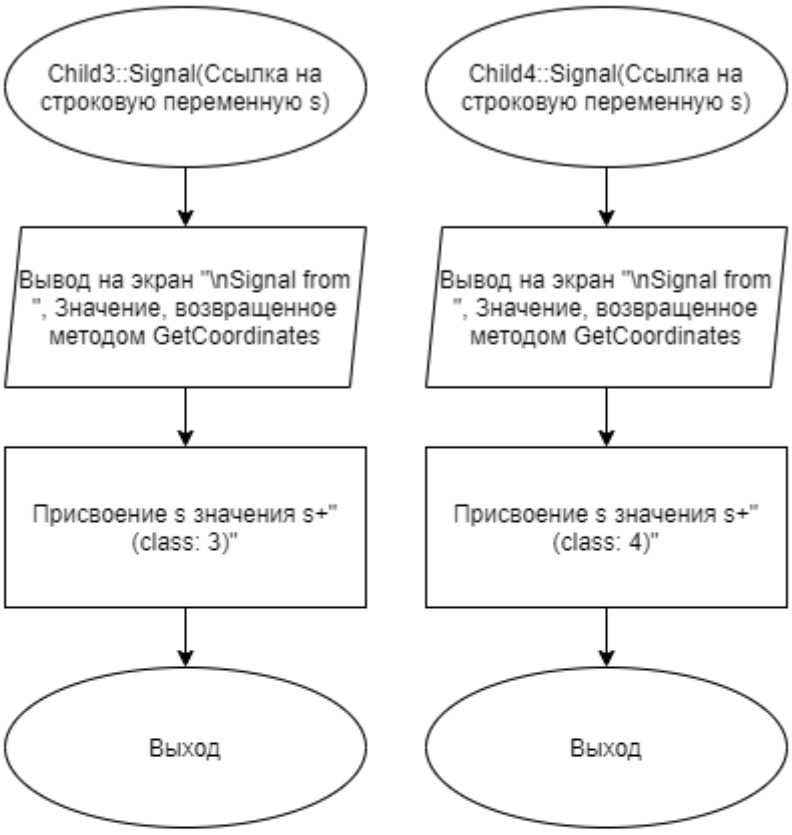

Рис. 2. Блок-схема алгоритма.

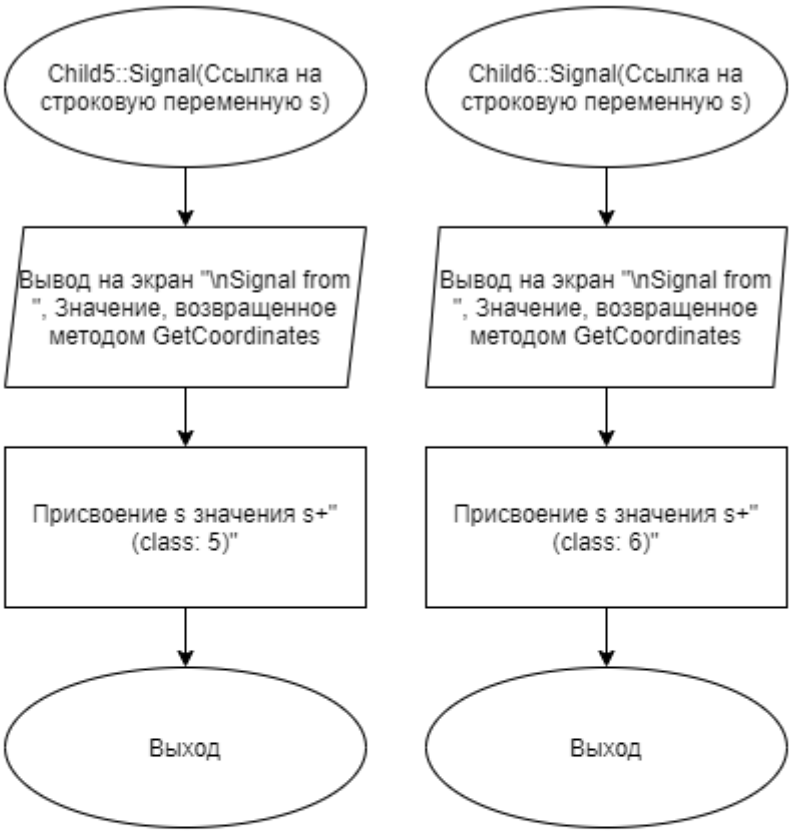

Рис. 3. Блок-схема алгоритма.

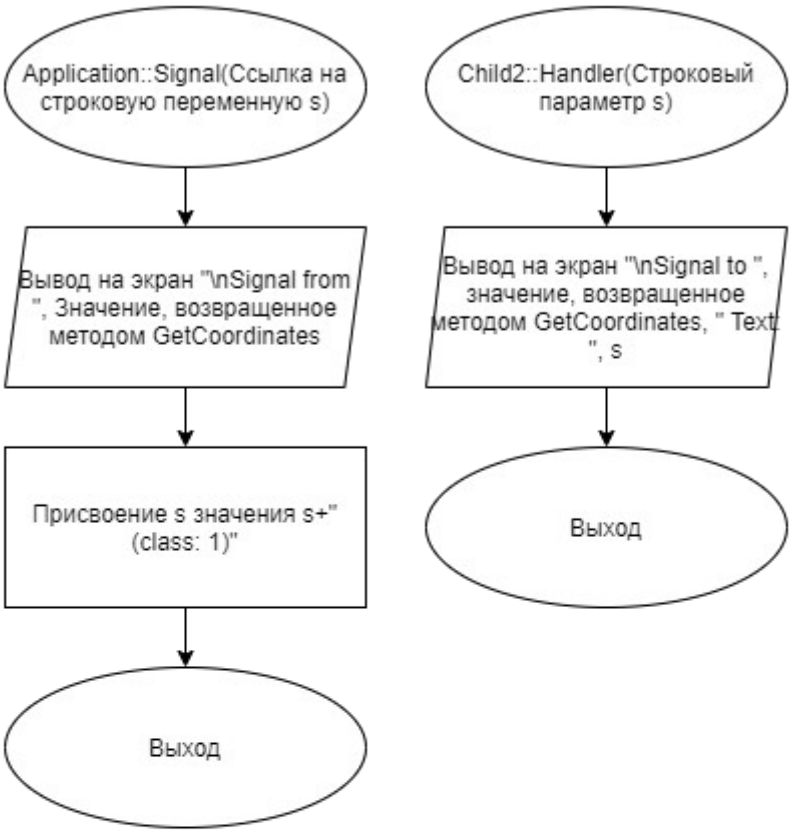

Рис. 4. Блок-схема алгоритма.

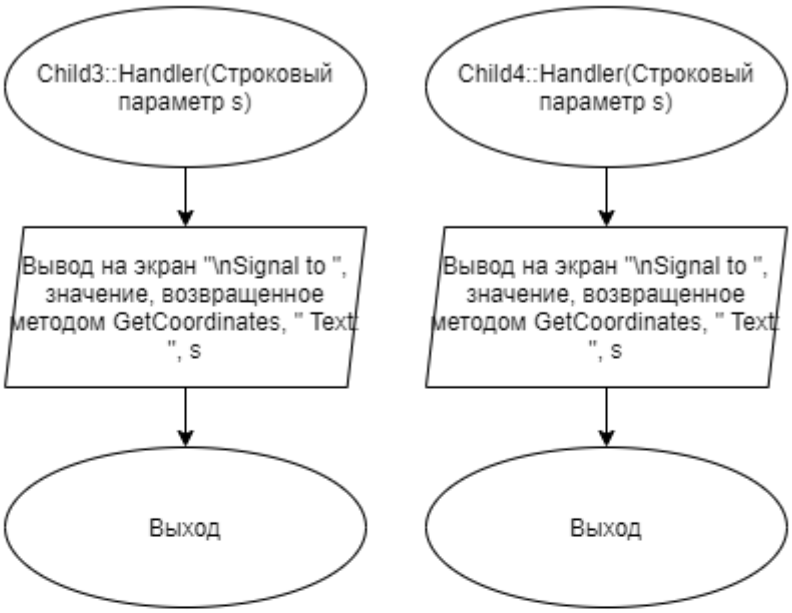

Рис. 5. Блок-схема алгоритма.

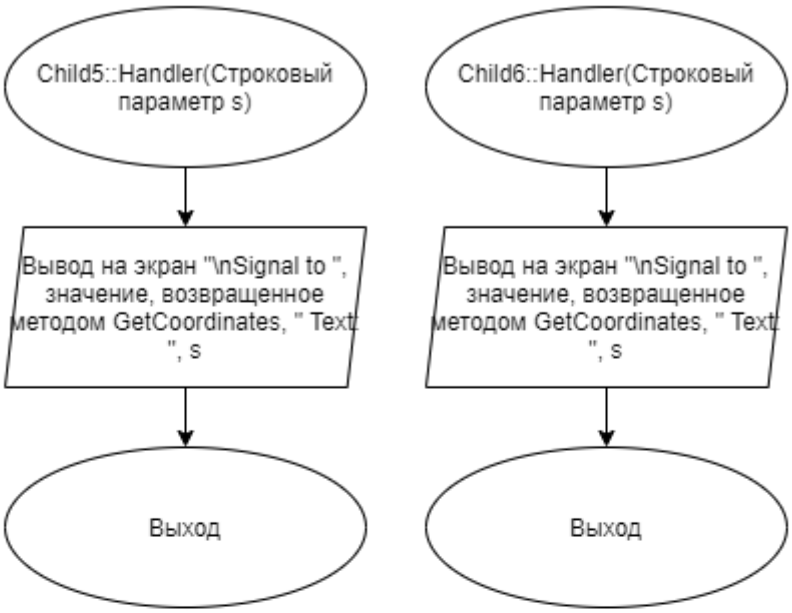

Рис. 6. Блок-схема алгоритма.

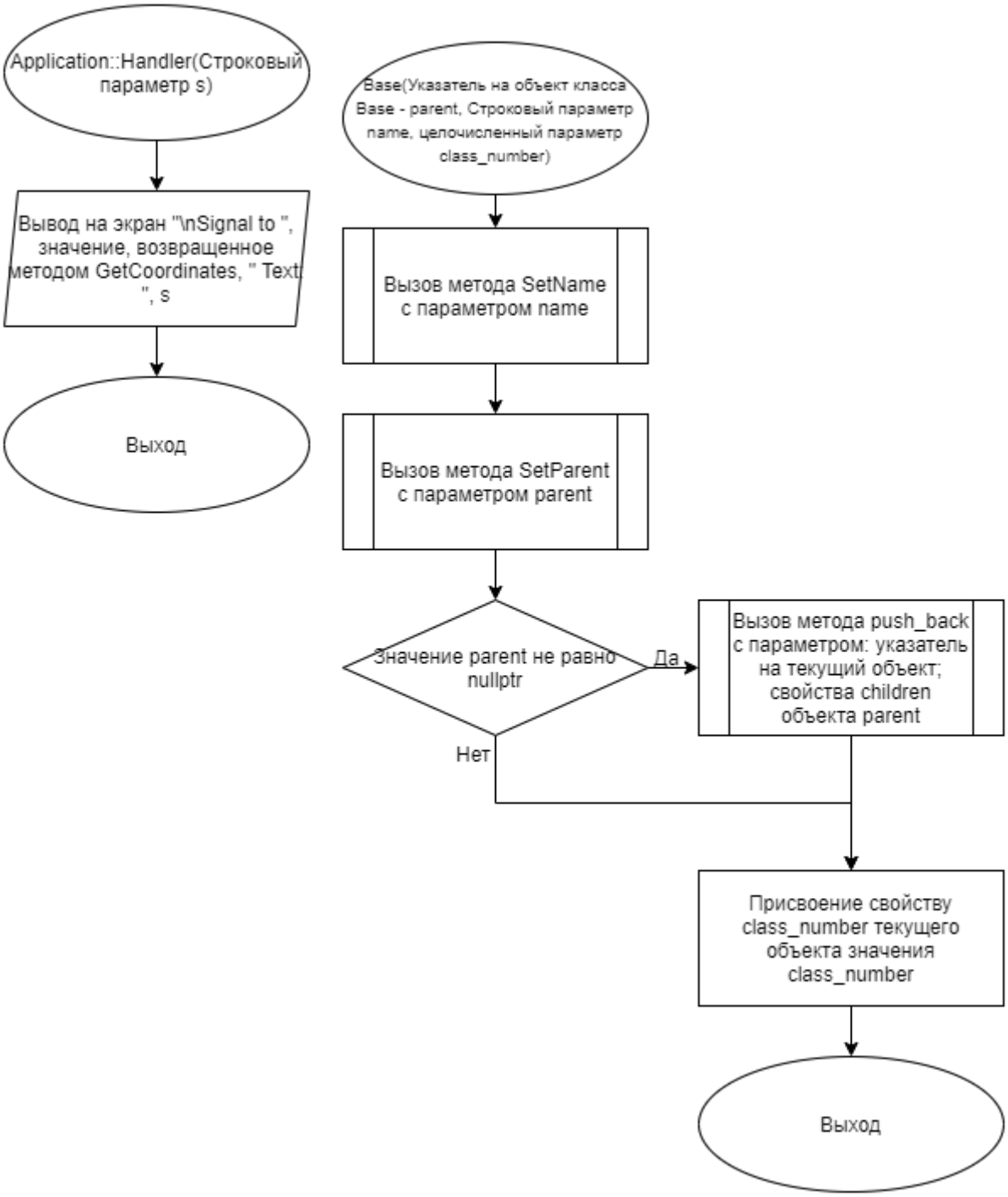

Рис. 7. Блок-схема алгоритма.

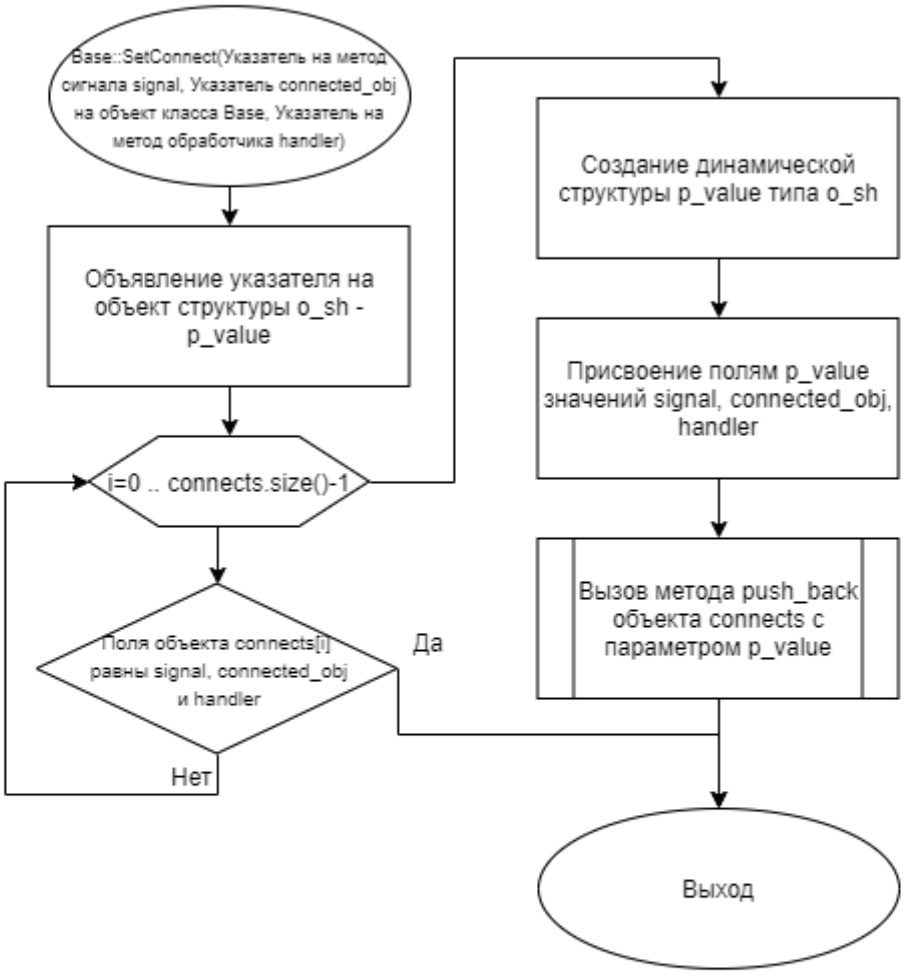

Рис. 8. Блок-схема алгоритма.

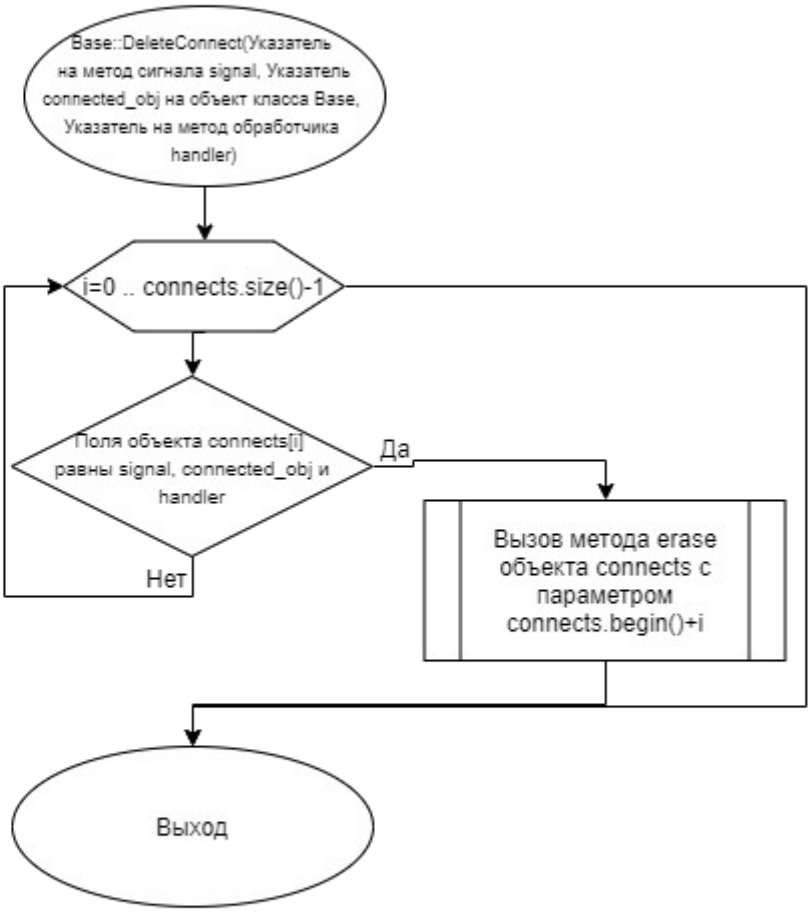

Рис. 9. Блок-схема алгоритма.

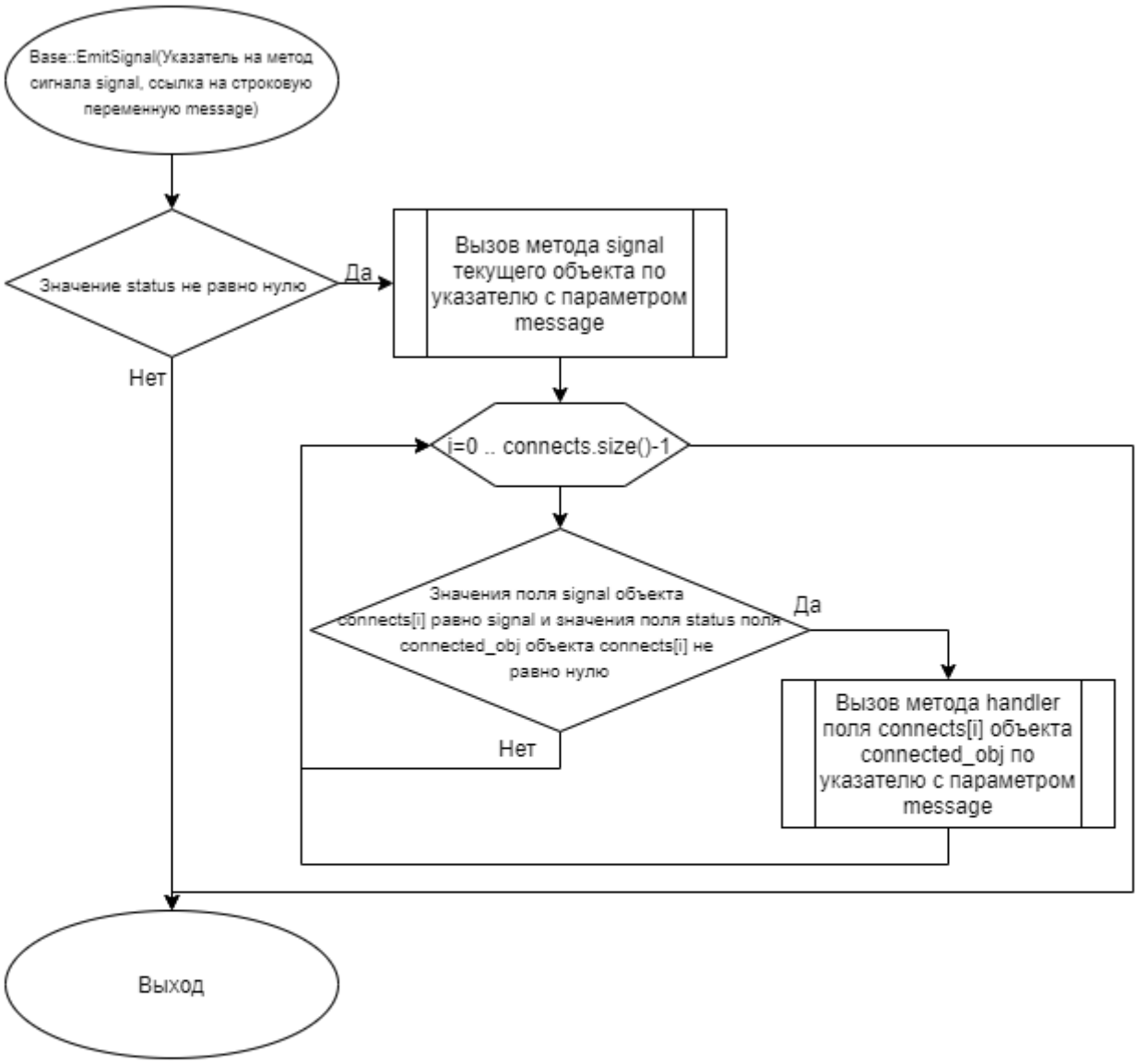

Рис. 10. Блок-схема алгоритма.

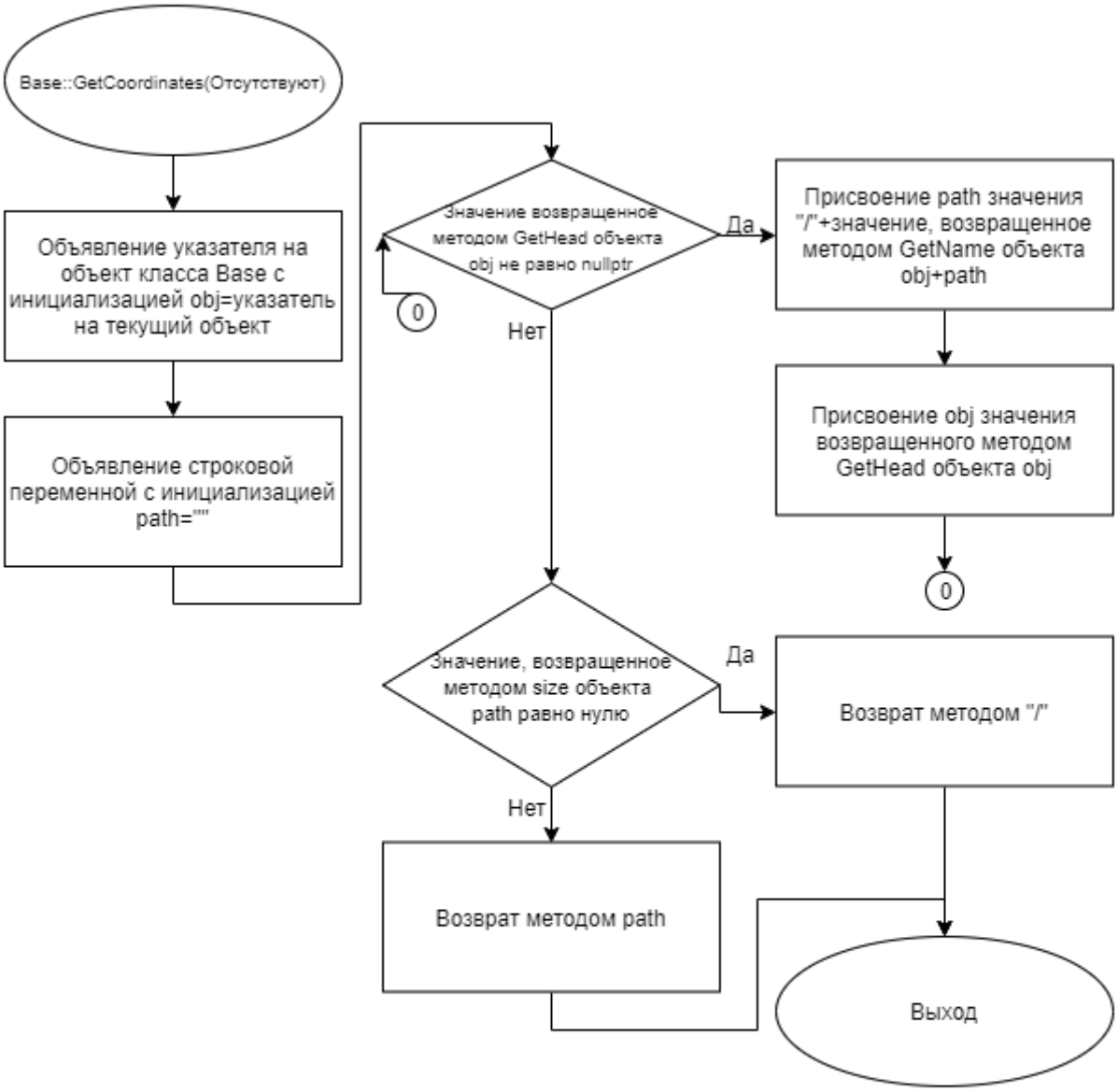

Рис. 11. Блок-схема алгоритма.

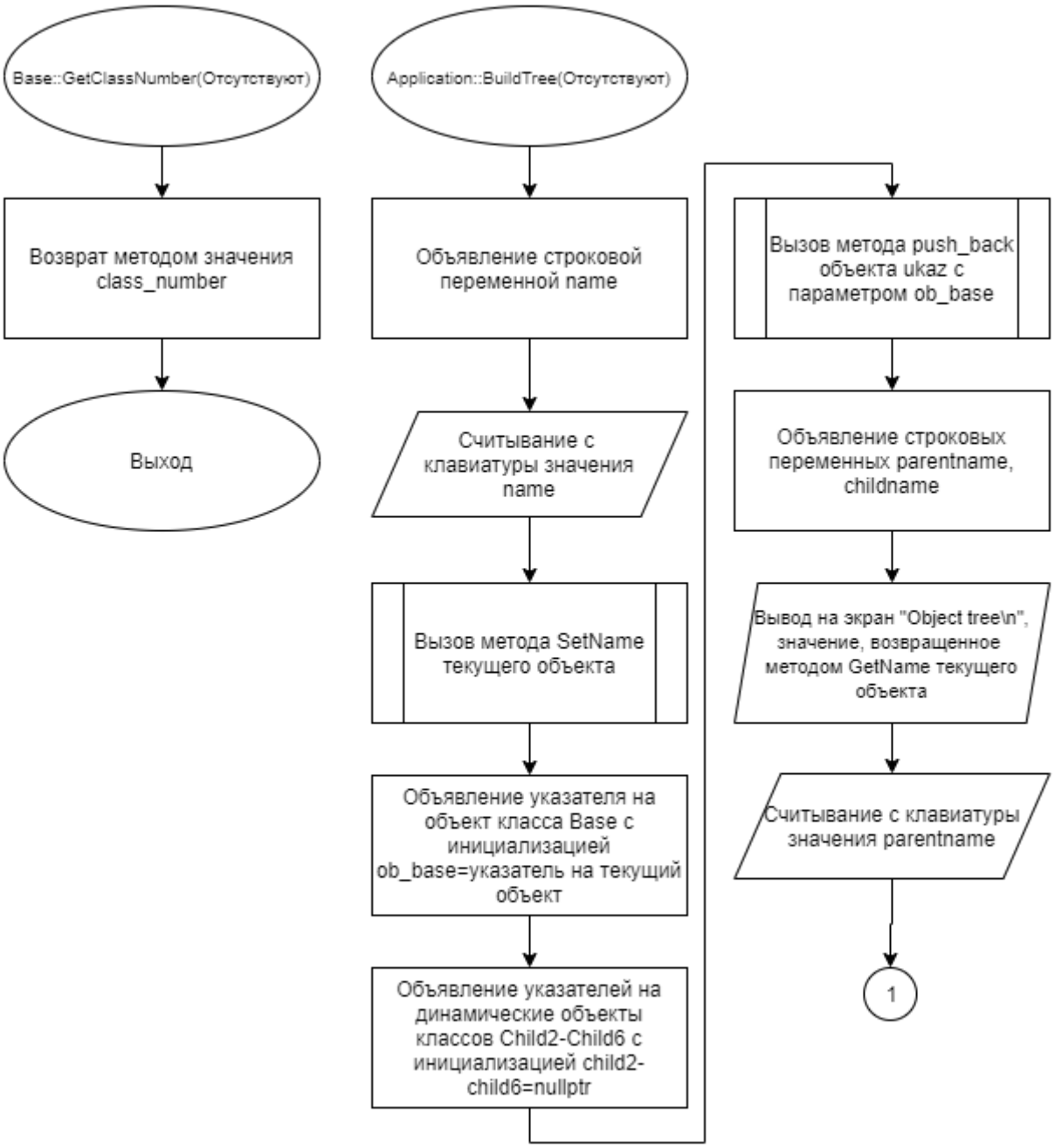

Рис. 12. Блок-схема алгоритма.

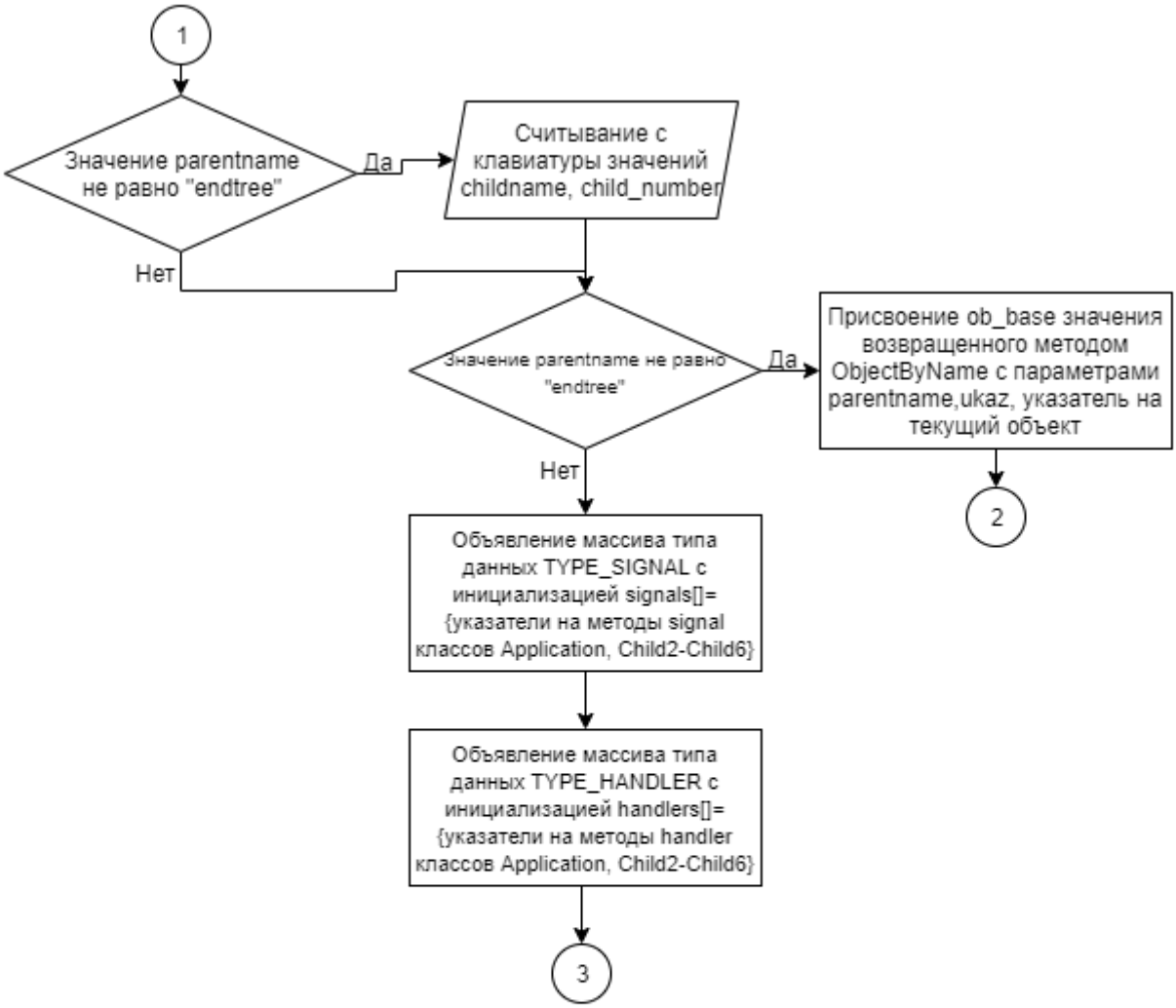

Рис. 13. Блок-схема алгоритма.

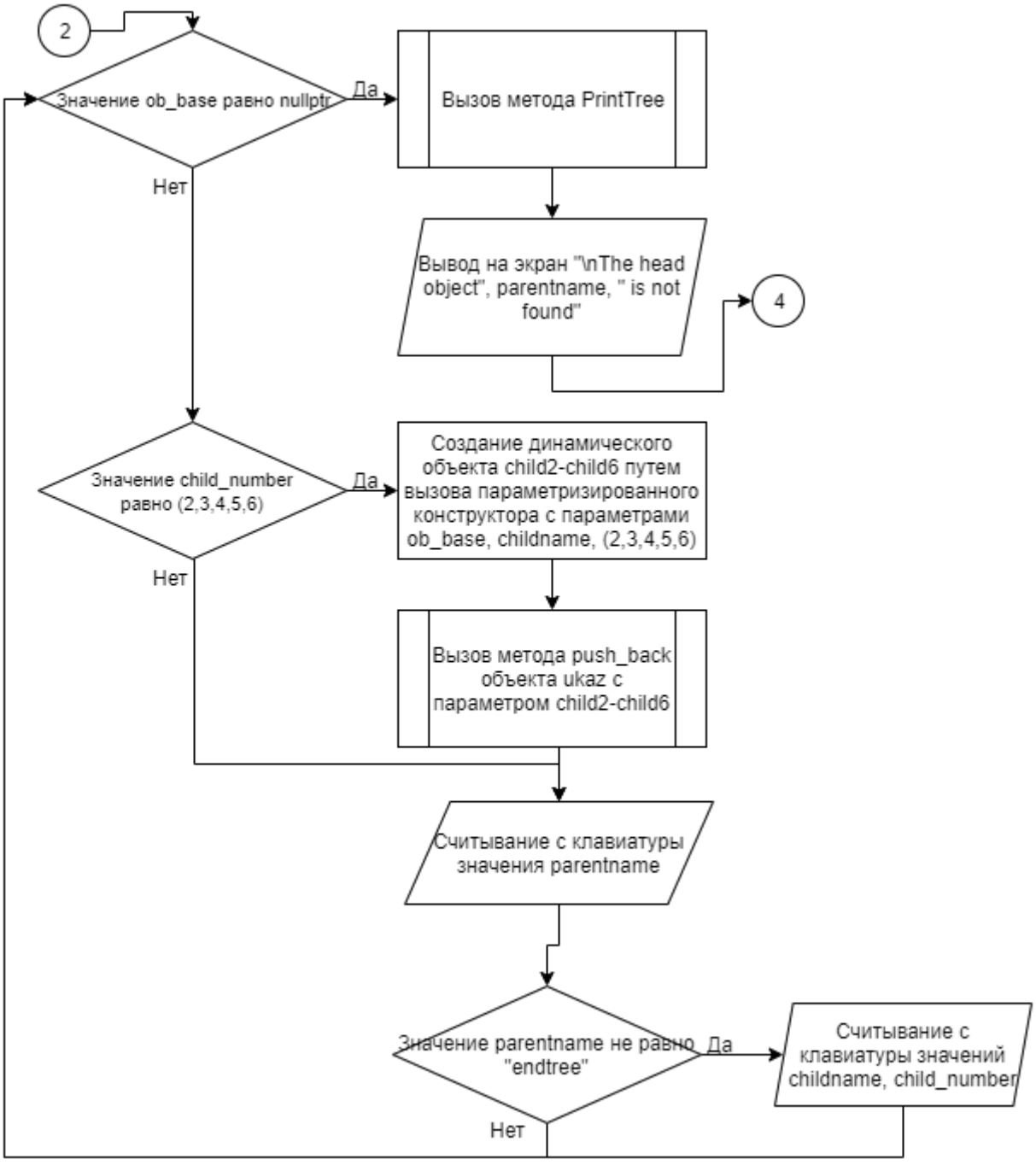

Рис. 14. Блок-схема алгоритма.

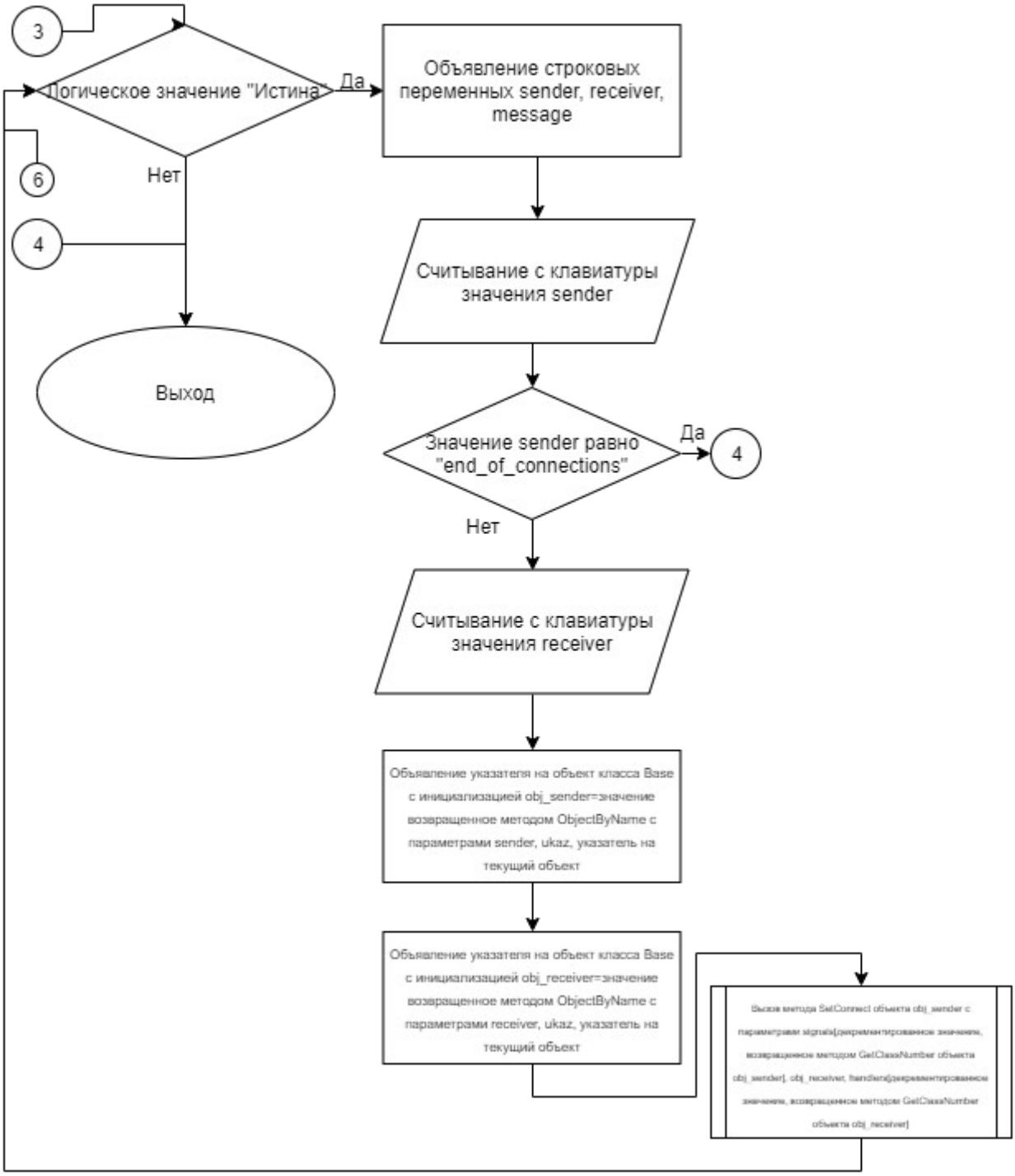

Рис. 15. Блок-схема алгоритма.

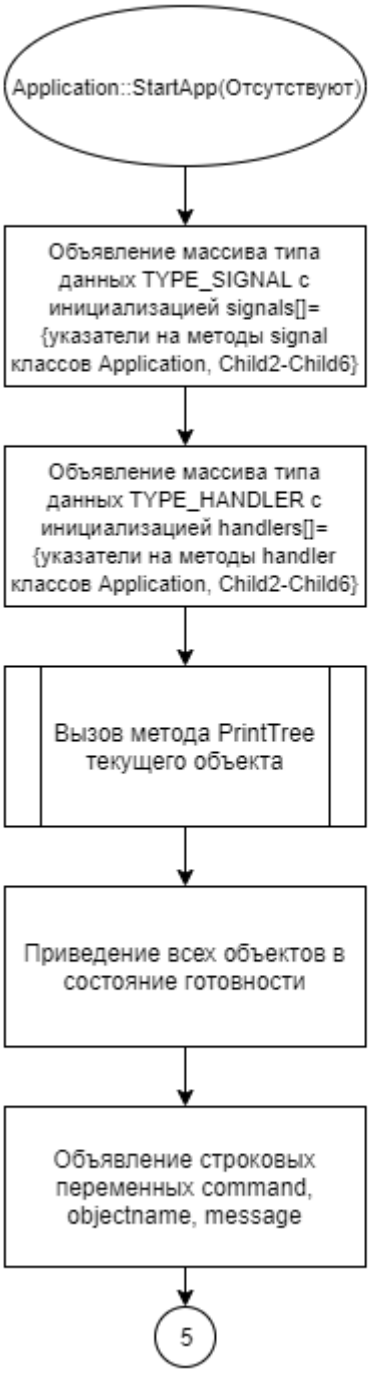

Рис. 16. Блок-схема алгоритма.

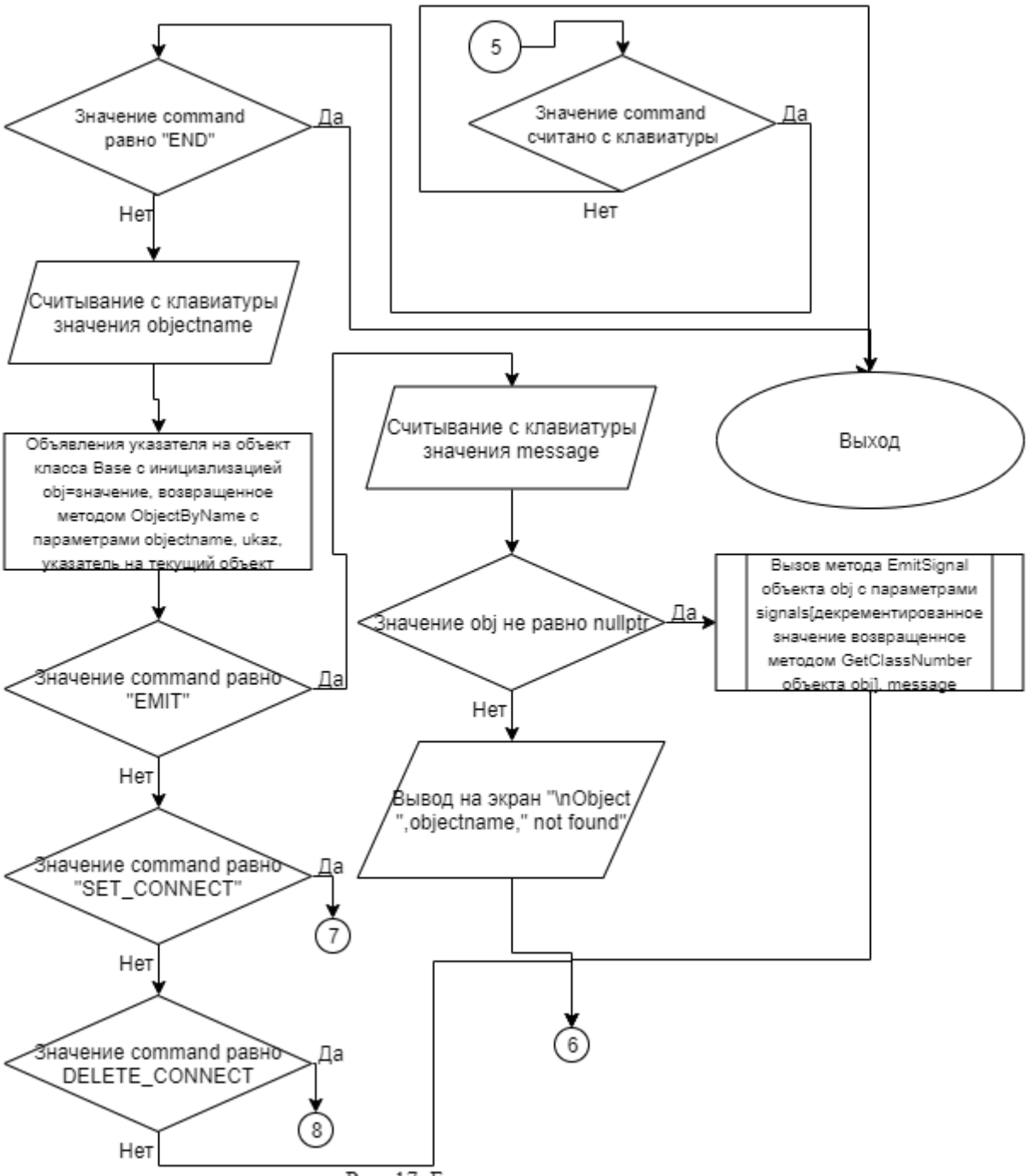

Рис. 17. Блок-схема алгоритма.

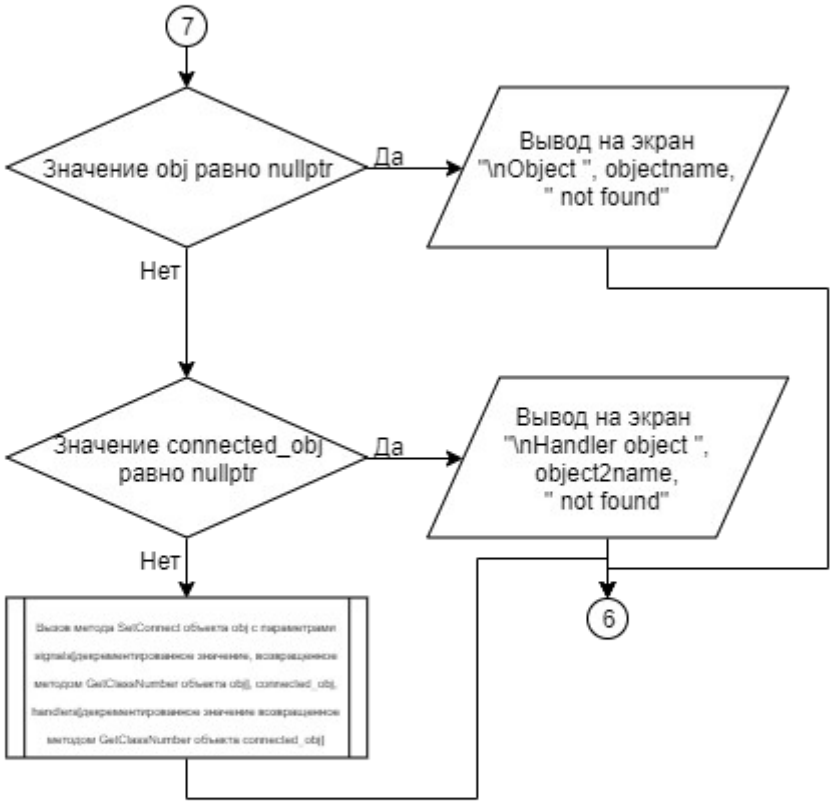

Рис. 18. Блок-схема алгоритма.

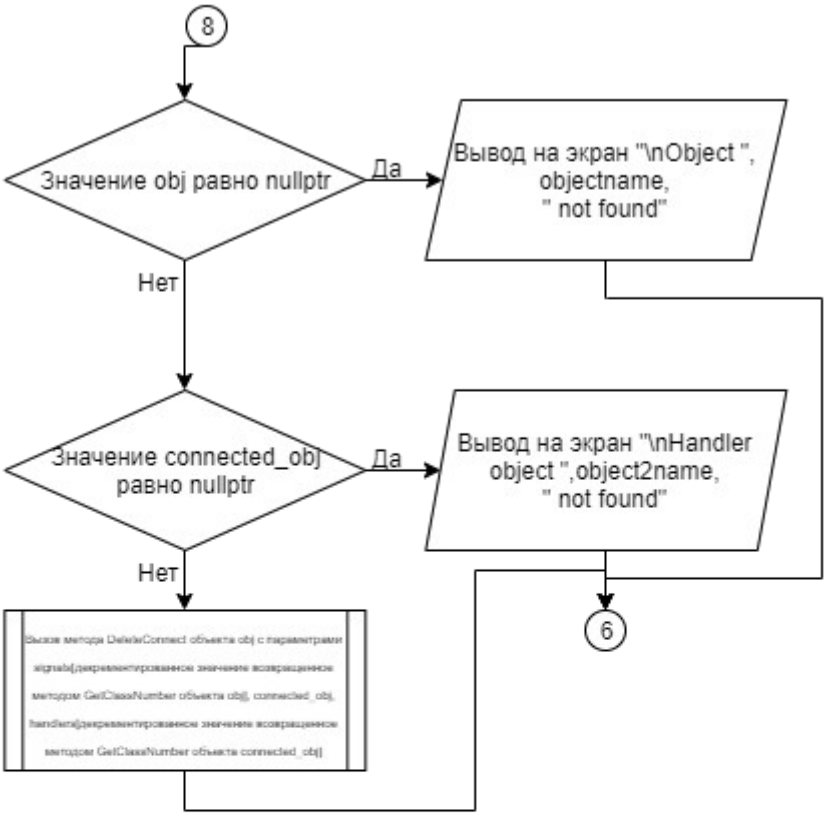

Рис. 19. Блок-схема алгоритма.

Код программы

Программная реализация алгоритмов для решения задачи представлена ниже.

Файл Application.cpp

```
#include "Application.h"

#include "Child2.h"
#include "Child3.h"
#include "Child4.h"
#include "Child5.h"
#include "Child6.h"
#include <iostream>
using namespace std;

#define SIGNAL_D(signal_f) (TYPE_SIGNAL) (&signal_f)
#define HENDLER_D(hendler_f) (TYPE_HANDLER) (&hendler_f)
typedef void (Base :: * TYPE_SIGNAL) (string&);
typedef void (Base :: * TYPE_HANDLER) (string);

void Application :: Signal(string& s){
        if (status)cout<<"\n"<<"Signal from "<<GetCoordinates();
        s+=" (class: 1)";
}

void Application :: Handler(string s){
        if (status)cout<<"\n"<<"Signal to "<<GetCoordinates()<<" Text: "<<s;
}

void Application::BuildTree() {
        string name;
        cin >> name;

        this->SetName(name);

        Base* ob_base=this;
        Child2* child2=nullptr;
        Child3* child3=nullptr;
        Child4* child4=nullptr;
        Child5* child5=nullptr;
        Child6* child6=nullptr;



        ukaz.push_back(ob_base);

        string parentname;
        string childname;
        int child_number;
```

```
        cout<<"Object tree\n";
        cout << this->GetName();

        cin>>parentname;
                if (parentname!="endtree")
                        cin>>childname>>child_number;

        while (parentname != "endtree") {

                ob_base = ObjectByName(parentname, ukaz, this);

                if (ob_base==nullptr) {PrintTree(); cout<<"\nThe head object
"<< parentname << " is not found"; exit(0);}

                if (child_number == 2) {child2=new
Child2(ob_base,childname,2); ukaz.push_back(child2);}
                else
                if (child_number == 3) {child3=new
Child3(ob_base,childname,3); ukaz.push_back(child3);}
                else
                if (child_number == 4) {child4=new
Child4(ob_base,childname,4); ukaz.push_back(child4);}
                else
                if (child_number == 5) {child5=new
Child5(ob_base,childname,5); ukaz.push_back(child5);}
                else
                if (child_number == 6) {child6=new
Child6(ob_base,childname,6); ukaz.push_back(child6);}

                cin>>parentname;
                if (parentname!="endtree")
                        cin>>childname>>child_number;

        }

        TYPE_SIGNAL signals[]={SIGNAL_D(Application::Signal),
SIGNAL_D(Child2::Signal),SIGNAL_D(Child3::Signal),

SIGNAL_D(Child4::Signal), SIGNAL_D(Child5::Signal), SIGNAL_D(Child6::Signal)};

        TYPE_HANDLER handlers[]={HENDLER_D(Application::Handler),
HENDLER_D(Child2::Handler),HENDLER_D(Child3::Handler),

HENDLER_D(Child4::Handler), HENDLER_D(Child5::Handler),
HENDLER_D(Child6::Handler)};

        while (true){
                string sender, receiver, message;
                cin>>sender;
                if (sender=="end_of_connections") break;

                cin>>receiver;

                Base* obj_sender=ObjectByName(sender,ukaz,this);
                Base* obj_receiver=ObjectByName(receiver,ukaz,this);

                obj_sender->SetConnect(signals[obj_sender->GetClassNumber()-
1],
                                                                obj_receiver,
```

```
handlers[obj_receiver->GetClassNumber()-1]);
        }
}

int Application::StartApp() {
        TYPE_SIGNAL signals[]={SIGNAL_D(Application::Signal),
SIGNAL_D(Child2::Signal),SIGNAL_D(Child3::Signal),

SIGNAL_D(Child4::Signal), SIGNAL_D(Child5::Signal), SIGNAL_D(Child6::Signal)};

        TYPE_HANDLER handlers[]={HENDLER_D(Application::Handler),
HENDLER_D(Child2::Handler),HENDLER_D(Child3::Handler),

HENDLER_D(Child4::Handler), HENDLER_D(Child5::Handler),
HENDLER_D(Child6::Handler)};

        this->PrintTree();

        for (int i=0;i<ukaz.size();i++)
                ukaz[i]->SetStatus(1);

        string command, objectname, message;
        while (cin>>command){
                if (command=="END") return 0;

                cin>>objectname;

                if (command=="EMIT"){
                        getline(cin,message);
                        Base* obj=ObjectByName(objectname,ukaz,this);
                        if (obj!=nullptr) obj->EmitSignal(signals[obj-
>GetClassNumber()-1], message);
                        else cout<<"\n"<<"Object "<<objectname<<" not found";
                }
                else
                if (command=="SET_CONDITION"){
                        int ready;
                        cin>>ready;
                        Base* obj=ObjectByName(objectname,ukaz,this);

                        //int status=atoi(objectname.c_str());
                        if (obj!=nullptr) obj->SetStatus(ready);
                        else cout<<"\n"<<"Object "<<objectname<<" not found";

                }
                else{
                        Base* obj=ObjectByName(objectname,ukaz,this);
                        string object2name;
                        cin>>object2name;
                if (command=="SET_CONNECT"){

                        Base*
connected_obj=ObjectByName(object2name,ukaz,this);
                                if (obj==nullptr) cout<<"\n"<<"Object
"<<objectname<<" not found";
                                else
                                if (connected_obj==nullptr) cout<<"\
```

```
n"<<"Handler object "<<object2name<<" not found";
                                    else
                                    obj->SetConnect(signals[obj->GetClassNumber()-
1], connected_obj,

handlers[connected_obj->GetClassNumber()-1]);
                            }
                            else
                            if (command=="DELETE_CONNECT"){

                            Base*
connected_obj=ObjectByName(object2name,ukaz,this);
                                    if (obj==nullptr) cout<<"\n"<<"Object
"<<objectname<<" not found";
                                    else
                                    if (connected_obj==nullptr) cout<<"\
n"<<"Handler object "<<object2name<<" not found";
                                    else
                                    obj->DeleteConnect(signals[obj-
>GetClassNumber()-1], connected_obj,

handlers[connected_obj->GetClassNumber()-1]);
                            }
                    }
                    }
        return 0;
}
```

Файл Application.h

```
#ifndef APP_H
#define APP_H

#define SIGNAL_D(signal_f) (TYPE_SIGNAL) (&signal_f)
#define HENDLER_D(hendler_f) (TYPE_HANDLER) (&hendler_f)

#include "Base.h"

class Application : public Base {
        using Base::Base;
private:
        vector <Base*> ukaz;

public:
        void BuildTree(); // доработан
        int StartApp(); // доработан

        void Signal(string& s); //новое
        void Handler(string s); //новое


};

#endif
```

Файл Base.cpp

```
#include "Base.h"
#include <iostream>
using namespace std;

#define SIGNAL_D(signal_f) (TYPE_SIGNAL) (&signal_f)
#define HENDLER_D(hendler_f) (TYPE_HANDLER) (&hendler_f)

typedef void (Base :: * TYPE_SIGNAL) (string&);
typedef void (Base :: * TYPE_HANDLER) (string);



Base::Base(Base* parent, string name, int class_number) {
        SetName(name);
        SetParent(parent);
        if (parent) { // !=nullptr
                parent->children.push_back(this);
        }
        this->class_number=class_number;
}

void Base::SetName(string name) {
        this->name = name; // наименование объекта
}

string Base::GetName() {
        return name; // возврат имени объекта
}

void Base::PrintTree() {


        for (int i = 0; i < children.size(); i++) {
                int flag=0;
                Base* ob=this;
                string s="    ";

                while (ob->GetHead()!=nullptr){
                        flag++;
                        ob=ob->GetHead();
                }

                for (int j=0;j<flag;j++)
                        s+="    ";

                cout<<"\n"<<s;

                cout<<children[i]->GetName();

                children[i]->PrintTree();
        }
        }
```

```
Base* Base::GetHead() {
	return  parent; // возврат указателя на головной объект
}

void Base::SetParent(Base* parent) {
	this->parent = parent;
}

Base* Base::ObjectByName(string name, vector <Base*> &vec, Base* current){
//получение объекта по имении в дереве иерархи
	if (name=="/"){
		Base* ob=this;
		while (ob->GetHead()!=nullptr){
			ob=ob->GetHead();
		}
		return ob;
	}
	else
	if (name=="."){
		return current;
	}
	else
	if (name.size()>1 && name[0]=='/' && name[1]!='/'){
		for (int i=0;i<vec.size();i++){
			string copy=name;
			Base* object;
			int last_slash_pos=copy.rfind("/");

			string ob="";
			for (int k=last_slash_pos+1;k<copy.size();k++)
				ob+=copy[k];

			if (vec[i]->GetName()==ob){
				object=vec[i];

				while (object->GetHead() != nullptr){
					last_slash_pos=copy.rfind("/");
					string ob="", ob2="";
					if (copy.rfind("/")==-1) return
nullptr;
					for (int
f=last_slash_pos+1;f<copy.size();f++)
						ob+=copy[f];

					copy.erase(last_slash_pos, ob.size()
+1);

					last_slash_pos=copy.rfind("/");
					if (last_slash_pos!=-1)
					for (int
p=last_slash_pos+1;p<copy.size();p++)
						ob2+=copy[p];

					if (ob2!=""){
						for (int j=0;j<vec.size();j++)
{
							if (vec[j]-
>GetName()==ob2) break;
```

```
                                                        if (j==vec.size()-1 &&
vec[j]->GetName()!=ob2) return nullptr;
                                                }
                                        }
                                        if (object->GetName()==ob)
object=object->GetHead();
                                        else break;
                                }
                                if (object->GetHead()==nullptr) return vec[i];
                        }
                }
                return nullptr;
        }
        if (name[0]=='/' && name[1]=='/'){
                name.erase(0,2);
                for (int i=0;i<vec.size();i++)
                        if (vec[i]->GetName()==name) return vec[i];
                return nullptr;
        }
        else
        if (name.size()==0 || (name[0]!='/' && name.find("/")>0)){
                if (name.find("/")!=0) name="/"+name;

                for (int i=0;i<vec.size();i++){
                        string copy=name;
                        Base* object;
                        int last_slash_pos=copy.rfind("/");

                        string ob="";
                        for (int k=last_slash_pos+1;k<copy.size();k++)
                                ob+=copy[k];

                        if (vec[i]->GetName()==ob){
                                object=vec[i];

                                while (object->GetHead() != current){
                                        last_slash_pos=copy.rfind("/");
                                        string ob="", ob2="";
                                        if (copy.rfind("/")==-1) return
nullptr;
                                        for (int
f=last_slash_pos+1;f<copy.size();f++)
                                                ob+=copy[f];

                                        copy.erase(last_slash_pos, ob.size()
+1);

                                        last_slash_pos=copy.rfind("/");
                                        if (last_slash_pos!=-1)
                                        for (int
p=last_slash_pos+1;p<copy.size();p++)
                                                ob2+=copy[p];

                                        if (ob2!=""){
                                                for (int j=0;j<vec.size();j++)
{
                                                        if (vec[j]-
```

```
>GetName()==ob2) break;
                                                        if (j==vec.size()-1 &&
vec[j]->GetName()!=ob2) return nullptr;
                                                }
                                        }
                                        if (object->GetName()==ob)
object=object->GetHead();
                                        else break;
                                }
                                if (object->GetHead()==current) return vec[i];
                        }
                }
                return nullptr;
        }

        else return nullptr;
}
void Base::SetStatus(int status){
        if (status==0){
                this->status=status;
                for (int i = 0; i < children.size(); i++)
                        children[i]->SetStatus(0);
        }
        else{
                Base* ob=this;
                while (ob->GetHead()!=nullptr){
                        ob=ob->GetHead();
                        if (ob->status==0) return;
                }
                this->status=status;
        }
}

void Base::PrintTreeAndStatus(){
        for (int i = 0; i < children.size(); i++) {
                int flag=0;
                Base* ob=this;
                string s="    ";

                while (ob->GetHead()!=nullptr){
                        flag++;
                        ob=ob->GetHead();
                }

                for (int j=0;j<flag;j++)
                        s+="    ";

                cout<<"\n"<<s;

                cout<<children[i]->GetName()<<" is ";
                if (children[i]->status!=0) cout<<"ready";
                else cout<<"not ready";

                children[i]->PrintTreeAndStatus();
        }
}

void Base :: SetConnect(TYPE_SIGNAL signal, Base* connected_obj, TYPE_HANDLER
handler){
        o_sh* p_value;
```

```
        for (int i=0;i<connects.size();i++){
                if (connects[i]->signal == signal && connects[i]-
>connected_obj == connected_obj &&
                        connects[i]->handler == handler) return;
        }

        p_value=new o_sh();

        p_value->signal=signal;
        p_value->connected_obj=connected_obj;
        p_value->handler=handler;

        connects.push_back(p_value);
}

void Base :: DeleteConnect(TYPE_SIGNAL signal, Base* connected_obj,
TYPE_HANDLER handler){
        for (int i=0;i<connects.size();i++)
                if (connects[i]->signal == signal && connects[i]-
>connected_obj == connected_obj &&
                        connects[i]->handler == handler)
{connects.erase(connects.begin()+i); return;}
}

void Base :: EmitSignal(TYPE_SIGNAL signal, string& message){
        if (status!=0){
                (this->*(signal))(message);
                for (int i=0;i<connects.size();i++){
                        if (connects[i]->signal==signal && connects[i]-
>connected_obj->status!=0)
                                (connects[i]->connected_obj->*(connects[i]-
>handler))(message);
                }
        }
}

string Base::GetCoordinates(){
        Base* obj=this;
        string path="";

        while (obj->GetHead()!=nullptr){
                path="/"+obj->GetName()+path;
                obj=obj->GetHead();
        }
        if (path.size()==0) return "/";
        else return path;
}

int Base::GetClassNumber(){
        return class_number;
}
```

Файл Base.h

```
#ifndef BASE_H
#define BASE_H

#define SIGNAL_D(signal_f) (TYPE_SIGNAL) (&signal_f)
#define HENDLER_D(hendler_f) (TYPE_HANDLER) (&hendler_f)

#include <string>
#include <vector>
using namespace std;

class Base {
        protected:
                string name;
                int status=1;
                vector <Base*> children;
        private:
                Base* parent;
                int class_number; //новое

        public:

                typedef void (Base :: * TYPE_SIGNAL) (string&);
                typedef void (Base :: * TYPE_HANDLER) (string);

                Base(Base* parent, string name, int class_number);

                void SetName(string name);
                void PrintTree();
                void SetParent(Base* parent);

                Base* GetHead();
                Base* ObjectByName(string name, vector <Base*> &vec, Base*
current);

                string GetName();

                void PrintTreeAndStatus();
                void SetStatus(int status);

//////////////////////////////////////////

                struct o_sh{ //новое
                        TYPE_SIGNAL signal;
                        Base* connected_obj;
                        TYPE_HANDLER handler;
                };

                vector <o_sh*> connects; //новое
                void SetConnect(TYPE_SIGNAL p_signal, Base* p_object,
TYPE_HANDLER p_ob_hendler); //новое
                void DeleteConnect(TYPE_SIGNAL p_signal, Base* p_object,
TYPE_HANDLER p_ob_hendler); //новое
                void EmitSignal(TYPE_SIGNAL p_signal, string& s_command);
//новое

                string GetCoordinates(); //новое

                int GetClassNumber(); //новое
};
```

```
#endif
```

Файл Child2.cpp

```
#include "Child2.h"
#include <iostream>
#include <string>
using namespace std;
void Child2 :: Signal(string& s){
        if (status)cout<<"\n"<<"Signal from "<<GetCoordinates();
        s+=" (class: 2)";
}

void Child2 :: Handler(string s){
        if (status)cout<<"\n"<<"Signal to "<<GetCoordinates()<<" Text: "<<s;
}
```

Файл Child2.h

```
#ifndef CHILD2_H
#define CHILD2_H

#include "Application.h"

class Child2 : public Base {
        using Base::Base;


        public:

                void Signal(string& s); //новое
                void Handler(string s); //новое



};

#endif
```

Файл Child3.cpp

```
#include "Child3.h"
#include <iostream>
#include <string>
```

```
using namespace std;
void Child3 :: Signal(string& s){
        if (status)cout<<"\n"<<"Signal from "<<GetCoordinates();
        s+=" (class: 3)";
}

void Child3 :: Handler(string s){
        if (status)cout<<"\n"<<"Signal to "<<GetCoordinates()<<" Text: "<<s;
}
```

Файл Child3.h

```
#ifndef CHILD3_H
#define CHILD3_H

#include "Application.h"

class Child3 : public Base {
        using Base::Base;


        public:
                void Signal(string& s); //новое
                void Handler(string s); //новое


};

#endif
```

Файл Child4.cpp

```
#include "Child4.h"
#include <iostream>
#include <string>
using namespace std;
void Child4 :: Signal(string& s){
        if (status)cout<<"\n"<<"Signal from "<<GetCoordinates();
        s+=" (class: 4)";
}

void Child4 :: Handler(string s){
        if (status)cout<<"\n"<<"Signal to "<<GetCoordinates()<<" Text: "<<s;
}
```

Файл Child4.h

```
#ifndef CHILD4_H
#define CHILD4_H

#include "Application.h"

class Child4 : public Base {
        using Base::Base;


        public:
                void Signal(string& s); //новое
                void Handler(string s); //новое


};

#endif
```

Файл Child5.cpp

```
#include "Child5.h"
#include <iostream>
#include <string>
using namespace std;
void Child5 :: Signal(string& s){
        if (status)cout<<"\n"<<"Signal from "<<GetCoordinates();
        s+=" (class: 5)";
}

void Child5 :: Handler(string s){
        if (status)cout<<"\n"<<"Signal to "<<GetCoordinates()<<" Text: "<<s;
}
```

Файл Child5.h

```
#ifndef CHILD5_H
#define CHILD5_H

#include "Application.h"

class Child5 : public Base {
        using Base::Base;


        public:
                void Signal(string& s); //новое
                void Handler(string s); //новое
```

```
};

#endif
```

Файл Child6.cpp

```
#include "Child6.h"
#include <iostream>
#include <string>
using namespace std;
void Child6 :: Signal(string& s){
        if (status)cout<<"\n"<<"Signal from "<<GetCoordinates();
        s+=" (class: 6)";
}

void Child6 :: Handler(string s){
        if (status)cout<<"\n"<<"Signal to "<<GetCoordinates()<<" Text: "<<s;
}
```

Файл Child6.h

```
#ifndef CHILD6_H
#define CHILD6_H

#include "Application.h"

class Child6 : public Base {
        using Base::Base;


        public:
                void Signal(string& s); //новое
                void Handler(string s); //новое

};

#endif
```

Файл main.cpp

```
#include "Base.h"
#include "Application.h"
#include "Child2.h"
#include <iostream>
#include <string>
using namespace std;
```

```
int main()
{
        Application app(nullptr, "", 1);
        app.BuildTree();
        return app.StartApp();
}
```

Тестирование

Результат тестирования программы представлен в следующей таблице.

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)
appls_root / object_s1 3 / object_s2 2 /object_s2 object_s4 4 / object_s13	Object tree appls_root object_s1 object_s7 object_s2 object_s4	Object tree appls_root object_s1 object_s7 object_s2 object_s4

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 EMIT /object_s2/object_s4 Send message 3 SET_CONNECT /object_s1 /object_s2/object_s6 EMIT /object_s1 Send message 4 END	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_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 3 (class: 4) Signal from /object_s1 Signal to /object_s2/object_s6 Text: Send message 4 (class: 3)	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_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 3 (class: 4) Signal from /object_s1 Signal to /object_s2/object_s6 Text: Send message 4 (class: 3)
appls_root / object_s1 3 / object_s2 2 /object_s200000 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	Object tree appls_root object_s1 object_s2 The head object /object_s200000 is not found	Object tree appls_root object_s1 object_s2 The head object /object_s200000 is not found

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		
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_s45 Send message 1 DELETE_CONNECT /object_s2/object_s4 / EMIT /object_s2/object_s45 Send message 2 EMIT /object_s2/object_s45 Send message 3 SET_CONNECT /object_s1 /object_s2/object_s6 EMIT /object_s1 Send message 4	Object tree appls_root object_s1 object_s7 object_s2 object_s4 object_s6 object_s13 Object /object_s2/object_s45 not found Object /object_s2/object_s45 not found Object /object_s2/object_s45 not found 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 Object /object_s2/object_s45 not found Object /object_s2/object_s45 not found Object /object_s2/object_s45 not found Signal from /object_s1 Signal to /object_s2/object_s6 Text: Send message 4 (class: 3)

END		

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ (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_laboratornyh_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).