Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Настоящая курсовая работа выполнена в соответствии с требованиями ГОСТ Единой системы программной документации (ЕСПД) [1]. Все этапы решения задач курсовой работы фиксированы, соответствуют требованиям, приведенным в методическом пособии для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [2-3] и методике разработки объектно-ориентированных программ [4-6].

Объектно-ориентированное программирование (ООП) — это способ писать программы, где мы думаем о программе как о наборе объектов, которые взаимодействуют друг с другом. Каждый объект имеет свои свойства и умения, которые он может делать. Мы можем создавать новые объекты, основанные на уже существующих, что делает код более организованным и легче понимаемым.

При использовании объектно-ориентированного программирования (ООП) основная цель программиста заключается в создании структурированного, модульного и легко поддерживаемого кода, разбитого на небольшие классы для удобства понимания, повторного использования и обеспечения безопасности, а также в получении практических навыков, которые помогут решать сложные задачи.

Целью данной курсовой работы является разработка системы имитирующей работу банковского сейфа, каждая ячейка которой открывается при наличии двух ключей, с применением обьектно-ориентированной парадигмы.

1 ПОСТАНОВКА ЗАДАЧИ

Надо моделировать работу системы, которая реализует работу банковского сейфа, каждая ячейка которой открывается при наличии двух ключей. Система состоит из следующих элементов:

- банковский сейф, состоящий из множества ячеек;
- ячейка банковского сейфа;
- пульт управления из кнопок и экрана отображения сообщений;
- ключ для открытия ячейки;
- сервер с программным обеспечением.

Сейф имеет прямоугольную форму и разбит на прямоугольники. В каждом прямоугольнике располагается банковская ячейка, которая отрывается двумя ключами. Этому разбиению по ячейкам можно сопоставить матрицу (n x m). Где, n – количество рядов, а m – количество столбцов. Значения n и m больше 2. Нумерация строк и столбцов начинается с 1. Ячейки перенумерованы по радам сверху вниз от 1 до n x m.

Ключ – это ПИН-код. Для клиента ключ состоит из 6 цифр, а для банковского работника из 7.

У сейфа есть кнопочный пульт управления. С пульта можно инициировать процедуру открытия ячейки и последовательно выполнить требуемые операции. Первоначально набирается номер ячейки, далее клиент вводит свой ключ. Если ключ относится к данной ячейке, то далее банковский работник может ввести свой ключ. Если все данные введены корректно, то ячейка открывается.

Нажатие на клавиши пульта управления моделируется посредством клавиатурного ввода. Ввод делится на команды:

- BOX «номер ячейки» - команда инициирует начало открытия банковской ячейки с заданным номером.

- CLIENT_KEY «ПИН-код клиента» - ввод ключа клиентом;
- BANK_KEY «ПИН-код банка» - ввод ключа банка сотрудником банка;
- CLOSE_BOX «номер ячейки» - команда закрытия клиентом банковской ячейки.
- CANCEL - отказ от операции открытия ячейки сейфа;
- Turn off the safe – выключение сейфа (завершение работы системы).

Система работает по тактам. В рамках одного такта вводится одна команда. В рамках каждого такта элементы системы выполняют предопределенные действия. Инициирование действий моделируется посредством запросов (сигналов).

Элементы системы моделируются посредством следующих объектов:

1. Объект «система». Обеспечивает отработку тактов.
2. Объект для чтения данных и команд. Объект считывает данные для подготовки и настройки системы. После чтения очередной порции данных для настройки или данных команды, объект выдает сигнал с текстом полученных данных. Все данные настройки и данные команд синтаксический корректны. Каждая строка данных или команд соответствует одному такту.
3. Объект пульта управления. Моделирует работу пульта управления сейфа. Отрабатывает поступающие данные и команды. Отслеживает допустимую последовательность поступающих команд. Объект вырабатывает решение и выдает соответствующий сигнал или сигналы. Выдает команду сейфу, для определения состояния сейфа или состояния ячейки. Готовит данные исходя из состояния системы.
4. Объект, моделирующий сейф. При построении дерева иерархии объектов, все объекты ячеек являются подчиненными сейфу. Содержит информацию относительно количества ячеек. Количества рядов и столбцов расположения

ячеек. По запросу от пульта управления опрашивает ячейки и определяет их состояние. Имеет состояния: готов к работе; ожидает ввода ПИН-кода клиента; ожидает ПИН-код банка. Другие состояния может определить разработчик.

5. Объект, моделирующий ячейку сейфа. Содержит информацию о номере ячейки, о состоянии ячейки. Имеет состояния: ячейка закрыта; ячейка открыта.

6. Объект, моделирующий сервер и программное обеспечение. Сервер хранит информацию о ПИН-кодах для каждой ячейки банковского сейфа. Получает команды от пульта управления. Выполняет проверку соответствия введенных данных с информацией на сервере. Возвращает результат проверки.

7. Объект моделирует экран пульта управления. Отображает состояния или результаты выполнения команд системы на консоль. Текст для вывода объект получает по сигналу от других объектов системы. Каждое присланное сообщение выводиться с новой строки.

Написать программу, реализующую следующий алгоритм:

1. Вызов метода объекта «система» build_tree_objects ( ).
    1.1. Построение дерева иерархии объектов. Характеристики объектов вводятся.
        1.1.1. Установка связей (сигнал-обработчик) и приведение части объектов в состоянии готовности, для обеспечения ввода данных и настройки системы.
        1.1.2. Выдача сигнала объекту чтения для ввода данных.
        1.1.3. Отработка операции загрузки данных.
    1.2. Установка связей сигналов и обработчиков между объектами.
2. Вызов метода объекта «система» exec_app ( ).

2.1. Приведение всех объектов в состояние готовности, исходя из допустимых состояний.

2.2. Цикл для обработки такта функционирования системы.

2.2.1. Выдача необходимых сигналов.

2.2.2. Отработка взаимодействия объектов и необходимых алгоритмов.

2.3. После ввода команды «Turn off the safe» завершить работу.

Все приведенные сигналы и соответствующие обработчики должны быть реализованы. Запрос от объекта означает выдачу сигнала. Все сообщения на консоль выводятся с новой строки.

В набор поддерживаемых команд добавить команду «SHOWTREE» и по этой команде вывести дерево иерархии объектов системы с отметкой о готовности и завершить работу системы (программы).

1.1 Описание входных данных

Первая строка, ввод информации о сейфе.

```
«количество рядов» «количество столбцов»
```

Со второй строки ввод информации о ключах ячеек сейфа.

```
«номер ячейки» «ключ клиента» «ключ банка»
. . . . .
```

Завершение ввода ключей:

```
Completing key entry
```

Далее следуют команды.

Последняя команда присутствует всегда:

```
Turn off the safe
```

Пример ввода:

```
3 3
1 100001 1001001
2 200001 2002001
3 300001 3003001
4 400001 4004001
5 500001 5005001
6 600001 6006001
7 700001 7007001
8 800001 8008001
9 900001 9009001
Completing key entry
BOX 3
CLIENT_KEY 300001
BANK_KEY 3003001
BOX 3
BOX 1
CLIENT_KEY 100001
CLIENT_KEY 200001
BOX 1
CLIENT_KEY 100017
CANCEL
BOX 1
CLIENT_KEY 100001
BANK_KEY 1003001
Turn off the safe
```

1.2 Описание выходных данных

После завершения построения дерева иерархии объектов и загрузки исходных данных отображается:

```
Ready to work
```

Это сообщение выводится после обнаружения ошибки в данных команды, нарушения последовательности вводимых команд, недопустимости выполнения команды или ввода команды «CANCEL».

Вывод информации приглашения на ввод:

```
Ready to work
```

Только после этого сообщения можно ввести команду номера ячейки. Если ячейка уже открыта, то выдается сообщение:

```
The safe deposit box is open
```

После команды BOX выдается приглашение ввода ПИН-кода клиента:

```
Enter the code
```

Если код введен корректно, выдается приглашение ввода ПИН-кода банка:

```
Enter the bank code
```

После подтверждения ключей ячейка открывается и выдается сообщение:

```
The safe deposit box «номер ячейки» is open
```

Если ключ клиента введен некорректно, то выдается сообщение:

```
The client is key is incorrect
```

Если ключ банка введен некорректно, то выдается сообщение:

```
The bank is key is incorrect
```

Если нарушена последовательность ввода команд, то выдается сообщение:

```
Error in the command sequence
```

Сообщение завершения работы системы:

```
Turn off the safe
```

Пример вывода:

```
Ready to work
Enter the code
Enter the bank code
The safe deposit box 3 is open
Ready to work
The safe deposit box is open
Ready to work
Enter the code
Enter the bank code
Error in the command sequence
Ready to work
Enter the code
The client is key is incorrect
Ready to work
Ready to work
Enter the code
Enter the bank code
The bank is key is incorrect
Ready to work
Turn off the safe
```

2 МЕТОД РЕШЕНИЯ

Для решения задачи используется:

- объект класса ClassBase предназначен для ;
- объект класса System предназначен для ;
- объект класса InfoNCommandReader предназначен для ;
- объект класса Controller предназначен для ;
- объект класса Vault предназначен для ;
- объект класса VaultCell предназначен для ;
- объект класса Server предназначен для ;
- объект класса ControllerDisplay предназначен для ;
- cin - объект стандартного потока ввода с клавиатуры;
- cout - объект стандартного потока вывода на экран;
- if, else - условный оператор (оператор ветвления);
- new - оператор создания нового объекта;
- for - оператор цикла;
- while - оператор цикла;
- switch - оператор выбора;
- vector - динамический массив;
- <vector> - библиотека для работы с векторами;
- <string> - библиотека для работы со строками;
- <algorithm> - библиотека для удобного удаления элементов из вектора .

Класс ClassBase:

- свойства/поля:
    - поле хранение наименования объекта:
        - наименование — objName;
        - тип — string;

        - модификатор доступа — private;
    - поле хранение указателя на родительский объект:
        - наименование — objHead;
        - тип — указатель на объект класса ClassBase;
        - модификатор доступа — private;
    - поле хранение массива указателей на подчиненные объекты:
        - наименование — objSub;
        - тип — vector>ClassBase*>;
        - модификатор доступа — private;
    - поле хранение состояния объекта:
        - наименование — objState;
        - тип — int;
        - модификатор доступа — private;
    - поле хранение структуры сигнала:
        - наименование — connectionStats;
        - тип — struct;
        - модификатор доступа — private;
    - поле хранение массива соединения сигналов:
        - наименование — connections;
        - тип — vector<connectionStats*>;
        - модификатор доступа — private;
- функционал:
    - метод ClassBase — параметризированный конструктор с параметром указателя наголовной объект в дереве иерархии и наименованием объекта;
    - метод getObjName — метод, возвращающий наименование объекта;
    - метод getObjHead — метод, возвращающий указатель на головной

объект текущего объекта;

- метод getObjState — метод, возвращающий состояние объекта;
- метод setObjName — метод, устанавливающий текущему объекту новое наименование;
- метод setObjHead — метод, устанавливающий текущему объекту новый родительский объект;
- метод setObjState — метод, устанавливающий текущему объекту новое состояние;
- метод printTreeAndState — метод, выводящий дерево объектов с их состояниями;
- метод findObjByCoordinate — метод, возвращающий указатель на найденный по координате объект;
- метод setConnection — метод, устанавливающий новое соединение;
- метод delConnection — метод, удаляющий существующее соединение;
- метод emitSignal — метод, запускающий сигнал.

Класс System:

- функционал:
  - метод System — параметризированный конструктор, вызывающий конструктор родительского класса ClassBase;
  - метод buildTree — метод, инициализирующий дерево объектов и связи объектов;
  - метод commandHadler — метод, запускающий сигнал приема команд;
  - метод startApp — метод, запускающий систему;
  - метод getSignal — метод, возвращающий сигнал объекта;
  - метод getHandler — метод, возвращающий обработчик объекта.

Класс InfoNCommandReader:

- функционал:
    - метод InfoNCommandReader — параметризированный конструктор, вызывающий конструктор родительского класса ClassBase;
    - метод getCinSignal — метод сигнала, считывающий строку;
    - метод getCinHandler — метод обработчика, возвращающий входящие данные.

Класс Controller:

- функционал:
    - метод Controller — параметризированный конструктор, вызывающий конструктор родительского класса ClassBase;
    - метод commandSignal — метод, принимающий строку с командой, обрабатывающий командуи вырабатывающий сигнал для вывода;
    - метод vaultInfoHandler — метод обработчика, возвращающий входящие данные.

Класс Vault:

- свойства/поля:
    - поле хранение ячейки сейфа:
        - наименование — cellMas;
        - тип — ClassBase**;
        - модификатор доступа — private;
    - поле хранение количества ячеек:
        - наименование — cellAmount;
        - тип — int;
        - модификатор доступа — private;
    - поле хранение количества рядов:
        - наименование — rowAmount;

        - тип — int;
        - модификатор доступа — private;
    - поле хранение открытия выбранной ячейки:
        - наименование — currentCellNum;
        - тип — string;
        - модификатор доступа — private;
- функционал:
    - метод Vault — параметризированный конструктор, вызывающий конструктор родительского класса ClassBase;
    - метод vaultSizeHandler — метод обработчика, записывающий количество ячеек, рядов и столбцов;
    - метод vaultCellCreateHandler — метод обработчика, создающий ячейки сейфа;
    - метод getCellAmountSignal — метод сигнала, записывающий в адрес передаваемой строки количество ячеек сейфа;
    - метод getVaultCellStateSignal — метод сигнала, записывающий в адрес передаваемой строки состояние ячейки;
    - метод getVaultStateSignal — метод сигнала, записывающий в адрес передаваемой строки состояние сейфа;
    - метод getCurrentCellNumSignal — метод сигнала, записывающий в адрес передаваемой строки номер открываемой ячейки;
    - метод changeVaultCellStateSignal — метод сигнала, изменяющий состояние ячейки;
    - метод setVaultStateSignal — метод сигнала, изменяющий состояние сейфа;
    - метод setCurrentCellNumSignal — метод сигнала, изменяющий номер открываемой ячейки.

Класс VaultCell:

- свойства/поля:
  - поле хранение номера ячейки:
    - наименование — cellNumber;
    - тип — int;
    - модификатор доступа — private;
- функционал:
  - метод VaultCell — параметризированный конструктор, вызывающий конструктор родительского класса ClassBase и устанавливающий номер ячейки.

Класс Server:

- свойства/поля:
  - поле хранение ПИН-кодов клиентов:
    - наименование — clientPinMas;
    - тип — string*;
    - модификатор доступа — private;
  - поле хранение ПИН-кодов банка:
    - наименование — bankPinMas;
    - тип — string*;
    - модификатор доступа — private;
- функционал:
  - метод Server — параметризированный конструктор, вызывающий конструктор родительского класса ClassBase;
  - метод createMassivesHandler — метод обработчика, создающий массивы ПИН-кодов клиента и банка;
  - метод setPinsHandler — метод обработчика, заполняющий массивы ПИН-кодов клиента и банка;

- метод clientPinCompareHandler — метод обработчика, сравнивающий входящий ПИН-код клиента и ПИН-код клиента на сервере и возвращающий результат;
- метод bankPinCompareHandler — метод обработчика, сравнивающий входящий ПИН-код банка и ПИН-код банка на сервере и возвращающий результат.

Класс ControllerDisplay:

- функционал:
  - метод ControllerDisplay — параметризированный конструктор, вызывающий конструктор родительского класса ClassBase;
  - метод controllerDisplayHandler — метод обработчика, выводящий передаваемую строку.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	ClassBase			Базовый класс в иерархии классов. Содержит основные поля и методы	
		System	public		2
		InfoNCommandReader	public		3
		Controller	public		4
		Vault	public		5
		VaultCell	public		6
		Server	public		7
		ControllerDisplay	public		8
2	System			Класс системы. Обеспечивает создание дерева объектов, вызов сигналов ввода первоначальных данных, вызов	

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
				сигналов ввода команд	
3	InfoNCommandReader			Класс, обеспечивающий ввод данных	
4	Controller			Класс пульта управления. Моделирует работу пульта управления сейфа. Отрабатывает поступающие данные и команды	
5	Vault			Класс сейфа. Содержит информацию относительно количества ячеек. Количества рядов и столбцов расположения ячеек	
6	VaultCell			Класс ячейки сейфа. Содержит информацию о номере ячейки, о состоянии ячейки	
7	Server			Класс сервера. Сервер хранит информацию о ПИН-кодах для каждой ячейки банковского сейфа. Проверяет вводимые ПИН-коды	
8	ControllerDisplay			Класс дисплея пульта управления. Отображает состояния или результаты выполнения команд системы на консоль	

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 ClassBase

Функционал: параметризированный конструктор с параметром указателя наголовной объект в дереве иерархии и наименованием объекта.

Параметры: ClassBase* objHead - головной объект, string objName - имя объекта.

Алгоритм конструктора представлен в таблице 2.

Таблица 2 – Алгоритм конструктора класса ClassBase

№	Предикат	Действия	№ перехода
1		Вызов метода setObjName с параметром objName	2
2	objHead не равен nullptr	Вызов метода setObjHead с параметром objHead	3
			∅
3		Добавление в вектор objSub объекта objHead текущего объекта	∅

3.2 Алгоритм метода getObjName класса ClassBase

Функционал: метод, возвращающий наименование объекта.

Параметры: нет.

Возвращаемое значение: string - наименование объекта.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода getObjName класса ClassBase

№	Предикат	Действия	№ перехода
1		Возврат objName	∅

3.3 Алгоритм метода getObjHead класса ClassBase

Функционал: метод, возвращающий указатель на головной объект текущего объекта.

Параметры: нет.

Возвращаемое значение: ClassBase* - указатель на головное объект.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода getObjHead класса ClassBase

№	Предикат	Действия	№ перехода
1		Возврат objHead	∅

3.4 Алгоритм метода getObjState класса ClassBase

Функционал: метод, возвращающий состояние объекта.

Параметры: нет.

Возвращаемое значение: int - состояние объекта.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода getObjState класса ClassBase

№	Предикат	Действия	№ перехода
1		Возврат objState	∅

3.5 Алгоритм метода setObjName класса ClassBase

Функционал: метод, устанавливающий текущему объекту новое

наименование.

Параметры: string objName - новое наименование объекта.

Возвращаемое значение: Без возвращаемого значения.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода setObjName класса ClassBase

№	Предикат	Действия	№ перехода
1		Присваивание переменной objName текущего объекта значения objName	∅

3.6 Алгоритм метода setObjHead класса ClassBase

Функционал: метод, устанавливающий текущему объекту новый родительский объект.

Параметры: ClassBase* objHead - новый указатель на головной объект.

Возвращаемое значение: Без возвращаемого значения.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода setObjHead класса ClassBase

№	Предикат	Действия	№ перехода
1		Присваивание переменной objHead текущего объекта значения objHead	∅

3.7 Алгоритм метода setObjState класса ClassBase

Функционал: метод, устанавливающий текущему объекту новое состояние.

Параметры: int objState - новое состояние объекта.

Возвращаемое значение: Без возвращаемого значения.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода setObjState класса ClassBase

№	Предикат	Действия	№ перехода
1	objState равен 0	Присваивание переменной objState текущего объекта значения objState	2
			5
2		Инициализация счетчика i с начальным значением равным 0	3
3	i < размер массива objSub	Вызов метода setObjState объекта objSub[i] с параметром objState	4
			5
4		Увеличение i на единицу	3
5	objHead не равен nullptr и objHead->objState не равен 0	присваивание переменной objState текущего объекта значения objState	∅
			∅

3.8 Алгоритм метода printTreeAndState класса ClassBase

Функционал: метод, выводящий дерево объектов с их состояниями.

Параметры: int step - глубина шага.

Возвращаемое значение: Без возвращаемого значения.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода printTreeAndState класса ClassBase

№	Предикат	Действия	№ перехода
1		Вывод << endl	2
2		Инициализация счетчика i с начальным значением равным 0	3
3	i меньше step	Вывод << " "	4
			5
4		Увеличение счетчика i на единицу	3

№	Предикат	Действия	№ перехода
5		Вывод возврата вызова метода getObjName текущего объекта	6
6	Возврат вызова getObjName текущего объекта равен "Vault"		7
			10
7	objState текущего объекта равен 1	Вывод << " is ready to work"	8
			8
8	objState текущего объекта равен 2	Вывод << " is waiting client PIN"	9
			9
9	objState текущего объекта равен 3	Вывод << " is waiting bank PIN"	14
			14
10	Поиск "VaultCell" в результате вызова метода getObjName текущего объекта не равен string::npos		11
			13
11	objState текущего объекта равен 1	Вывод << " is closed"	12
			12
12	objState текущего объекта равен 2	Вывод << " is opened"	14
			14
13	objState текущего объекта равен 0	Вывод << " is ready"	14
		Вывод << " is not ready"	14

№	Предикат	Действия	№ перехода
14	Размер вектора objSub текущего объекта больше 0	Инициализация счетчика i с начальным значением равным 0	15
			∅
15	i меньше размера вектора objSub текущего объекта	Вызов метода printTreeAndState объекта objSub[i] с параметром step + 1	16
			∅
16		Увеличение счетчика i на единицу	15

3.9 Алгоритм метода findObjByCoordinate класса ClassBase

Функционал: метод, возвращающий указатель на найденный по координате объект.

Параметры: ClassBase* CurrentObj - текущий объект, string Coordinate - координата.

Возвращаемое значение: ClassBase* - указатель на найденный объект.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода findObjByCoordinate класса ClassBase

№	Предикат	Действия	№ перехода
1		Инициализация переменной типа ClassBase* с наименованием object и значением nullptr	2
2	Длина Coordinate больше 0 и первый символ Coordinate равен '/'	Удаление из Coordinate первого символа	3
			4
3	Coordinate равно ""	Возврат текущего объекта	∅
		Вызов метода findObjByCoordinate текущего объекта с параметрами CurrentObj и Coordinate и	∅

№	Предикат	Действия	№ перехода
		возврат результата	
4	Длина Coordinate больше 0	Инициализация переменной типа size_t с наименованием position и значением результата поиска "/" в Coordinate	5
		Возврат object	∅
5		Инициализация переменной типа string с наименованием ObjName и значением переменной Coordinate	6
6	position не равен string::npos	Удаление из ObjName символов начиная с position и до конца; Удаление из Coordinate символов начиная с 0 и до position	7
		Присваивание Coordinate значения ""	7
7		Инициализация счетчика i с начальным значением 0	8
8	i меньше длины вектора objSub объекта CurrentObj		9
			12
9	Возврат метода getObjName объекта objSub[i] объекта CurrentObj равен ObjName	Присваивание object значения objSub[i] CurrentObj'a	10
			11
10	Coordinate не равно ""	Вызов метода findObjByCoordinate объекта object с возвратом результата	∅
		Возврат object	∅
11		Увеличение счетчика i на единицу	8
12		Возврат object	∅

3.10 Алгоритм метода setConnection класса ClassBase

Функционал: метод, устанавливающий новое соединение.

Параметры: string signalName - имя создаваемого сигнала, TYPE_SIGNAL signal - указатель на метод сигнала текущего объекта, ClassBase* object - указатель на целевой объект, TYPE_HANDLER handler - указатель на метод обработчика целевого объекта.

Возвращаемое значение: Без возвращаемого значения.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода setConnection класса ClassBase

№	Предикат	Действия	№ перехода
1		Инициализация счетчика i с начальным значением 0	2
2	i меньше размера вектора connections		3
			5
3	Элемент signalName элемента i вектора connections равен signalName	Возврат	∅
			4
4		Увеличение счетчика i на единицу	2
5		Инициализация переменной типа connectionStats* с наименованием newConnection	6
6		Присваивание newConnection ячейке signalName значения signalName Присваивание newConnection ячейке signal значения signal Присваивание newConnection ячейке baseClass	7

№	Предикат	Действия	№ перехода
		значения object Присваивание newConnection ячейке handler значения handler	
7		Добавление в конец вектора connections текущего объекта значения newConnection	∅

3.11 Алгоритм метода delConnection класса ClassBase

Функционал: метод, удаляющий существующее соединение.

Параметры: string signalName - имя удаляемого соединения.

Возвращаемое значение: Без возвращаемого значения.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода delConnection класса ClassBase

№	Предикат	Действия	№ перехода
1		Инициализация счетчика i с начальным значением 0	2
2	i меньше размера вектора connections		3
			∅
3	Элемент signalName элемента i вектора connections равен signalName	Удаление из массива connections текущего объекта элемента i	4
			5
4		Выход из цикла	∅
5		Увеличение i на единицу	2

3.12 Алгоритм метода emitSignal класса ClassBase

Функционал: метод, запускающий сигнал.

Параметры: string signalName - имя сигнала, string info - передаваемая информация.

Возвращаемое значение: string - результат выполнения обработчика.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода emitSignal класса ClassBase

№	Предикат	Действия	№ перехода
1	objState равен 0	Возврат "null"	∅
			2
2		Инициализация переменной типа string с наименованием infoToReturn и со значением "null"	3
3		Инициализация счетчика i с начальным значением 0	4
4	i меньше размера вектора connections		5
			10
5	Элемент signalName элемента i вектора connections равен signalName		6
			9
6	Элемент signal элемента i вектора connections не равен nullptr	Вызов метода сигнала текущего объекта с параметром info	7
			7
7	Элемент handler элемента i вектора connections не равен nullptr	Вызов метода обработчика целевого объекта с параметром info и присваивание возврата переменной infoToReturn	8

№	Предикат	Действия	№ перехода
			8
8		Выход из цикла	10
9		Увеличение счетчика i на единицу	4
10		Возврат infoToReturn	∅

3.13 Алгоритм конструктора класса System

Функционал: параметризированный конструктор, вызывающий конструктор родительского класса ClassBase.

Параметры: ClassBase* objHead - указатель на головной объект.

Алгоритм конструктора представлен в таблице 14.

Таблица 14 – Алгоритм конструктора класса System

№	Предикат	Действия	№ перехода
1		Вызов конструктора родительского класса ClassBase	∅

3.14 Алгоритм метода buildTree класса System

Функционал: метод, инициализирующий дерево объектов и связи объектов.

Параметры: нет.

Возвращаемое значение: Без возвращаемого значения.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода buildTree класса System

№	Предикат	Действия	№ перехода
1		Инициализация переменной типа ClassBase* с наименованием RootObj и значением текущего объекта	2
2		Инициализация переменной типа ClassBase* с	3

№	Предикат	Действия	№ перехода
		наименованием Obj и значением nullptr	
3		Инициализация переменной типа ClassBase* с наименованием InputReaderObj и значением нового объекта класса InfoNCommandReader	4
4		Инициализация переменной типа ClassBase* с наименованием ControllerObj и значением нового объекта класса Controller	5
5		Инициализация переменной типа ClassBase* с наименованием VaultObj и значением нового объекта класса Vault	6
6		Вызов метода setConnection для объекта InputReaderObj с параметрами: "GetInputNSetVaultSize", результат вызова метода getSignal("getCinSignal"), VaultObj, результат вызова метода getHandler("vaultSizeHandler")	7
7		Вызов метода emitSignal для объекта InputReaderObj с параметром "GetInputNSetVaultSize"	8
8		Вызов метода setConnection для объекта VaultObj с параметрами: "CellCreation", результат вызова метода getSignal("getCellAmountSignal"), VaultObj, результат вызова метода getHandler("vaultCellCreateHandler")	9
9		Вызов метода emitSignal для объекта VaultObj с параметром "CellCreation"	10
10		Инициализация переменной типа ClassBase* с наименованием ServerObj и значением нового объекта класса Server	11

№	Предикат	Действия	№ перехода
11		Вызов метода setConnection для объекта VaultObj с параметрами: "CreateServerMassives", результат вызова метода getSignal("getCellAmountSignal"), ServerObj, результат вызова метода getHandler("createMassives")	12
12		Вызов метода emitSignal для объекта VaultObj с параметром "CreateServerMassives"	13
13		Инициализация переменной типа ClassBase* с наименованием ControllerDisplayObj и значением нового объекта класса ControllerDisplay	14
14		Вызов метода setConnection для объекта InputReaderObj с параметрами: "SetPins", результат вызова метода getSignal("getCinSignal"), ServerObj, результат вызова метода getHandler("setPinsHandler")	15
15		Вызов метода setConnection для объекта InputReaderObj с параметрами: "GetLine", результат вызова метода getSignal("getCinSignal"), InputReaderObj, результат вызова метода getHandler("getCinHandler")	16
16		Вызов метода setConnection для объекта ControllerObj с параметрами: "ProcessCommandNPrintResult", результат вызова метода getSignal("commandSignal"), ControllerDisplayObj, результат вызова метода getHandler("controllerDisplayHandler")	17
17		Вызов метода setConnection для объекта VaultObj с параметрами: "GetVaultCellState", результат	18

№	Предикат	Действия	№ перехода
		вызова метода getSignal("getVaultCellStateSignal"), ControllerObj, результат вызова метода getHandler("vaultInfoHandler")	
18		Вызов метода setConnection для объекта VaultObj с параметрами: "GetVaultState", результат вызова метода getSignal("getVaultStateSignal"),ControllerObj, результат вызова метода getHandler("vaultInfoHandler")	19
19		Вызов метода setConnection для объекта VaultObj с параметрами: "GetCurrentCellNum", результат вызова метода getSignal("getCurrentCellNumSignal"), ControllerObj, результат вызова метода getHandler("vaultInfoHandler")	20
20		Вызов метода setConnection для объекта VaultObj с параметрами: "ChangeVaultCellState", результат вызова метода getSignal("changeVaultCellStateSignal")	21
21		Вызов метода setConnection для объекта VaultObj с параметрами: "SetVaultState", результат вызова метода getSignal("setVaultStateSignal")	22
22		Вызов метода setConnection для объекта VaultObj с параметрами: "SetCurrentCellNum", результат вызова метода getSignal("setCurrentCellNumSignal")	23
23		Вызов метода setConnection для объекта ControllerObj с параметрами: "CompareClientPin", nullptr, ServerObj, результат вызова метода	24

№	Предикат	Действия	№ перехода
		getHandler("clientPinCompareHandler")	
24		Вызов метода setConnection для объекта ControllerObj с параметрами: "CompareBankPin", nullptr, ServerObj, результат вызова метода getHandler("bankPinCompareHandler")	25
25		Вызов метода emitSignal с параметром "SetPins" и присваивание результата вызова инициализированной переменной типа string и с наименованием inputReaderAnswer	26
26	inputReaderAnswer не равен "Completing key entry"	Вызов метода emitSignal с параметром "SetPins" и присваивание результата вызова переменной inputReaderAnswer	26
			∅

3.15 Алгоритм метода commandHadler класса System

Функционал: метод, запускающий сигнал приема команд.

Параметры: нет.

Возвращаемое значение: Без возвращаемого значения.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода commandHadler класса System

№	Предикат	Действия	№ перехода
1		Инициализация переменной типа ClassBase* с наименованием RootObj и значением текущего объекта	2
2		Вызов метода findObjByCoordinate с параметрами RootObj и "/InfoNCommandReader" и запись результата вызова в инициализированную	3

№	Предикат	Действия	№ перехода
		переменную типа ClassBase* и наименованием InputReaderObj	
3		Вызов метода findObjByCoordinate с параметрами RootObj и "/Controller" и запись результата вызова в инициализированную переменную типа ClassBase* и наименованием ControllerObj	4
4		Вывод << "Ready to work"	5
5		Вызов метода emitSignal объекта ControllerObj с параметрами: "ProcessCommandNPrintResult", результат вызова метода emitSignal("GetLine") объекта InputReaderObj, и запись результата вызова в инициализированную переменную типа string и с наименованием controllerAnswer	6
6	controllerAnswer не равен "SHOWTREE" и controllerAnswer не равен "Turn off the safe"	Вызов метода emitSignal объекта ControllerObj с параметрами: "ProcessCommandNPrintResult", результат вызова метода emitSignal("GetLine") объекта InputReaderObj, и запись результата вызова в controllerAnswer	6
			∅

3.16 Алгоритм метода startApp класса System

Функционал: метод, запускающий систему.

Параметры: нет.

Возвращаемое значение: int-индикатор успешного выполнения программы.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода startApp класса System

№	Предикат	Действия	№ перехода
1		Вызов метода commandHandler текущего объекта	∅

3.17 Алгоритм метода getSignal класса System

Функционал: метод, возвращающий сигнал объекта.

Параметры: string methodName - наименование метода.

Возвращаемое значение: TYPE_SIGNAL - указатель на метод сигнала текущего объекта.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода getSignal класса System

№	Предикат	Действия	№ перехода
1	methodName равен "getCinSignal"	Возврат указателя на метод getCinSignal класса InfoNCommandReader	∅
			2
2	methodName равен "getCellAmountSignal"	Возврат указателя на метод getCellAmountSignal класса Vault	∅
			3
3	methodName равен "commandSignal"	Возврат указателя на метод commandSignal класса Controller	∅
			4
4	methodName равен "getVaultCellStateSignal"	Возврат указателя на метод getVaultCellStateSignal класса Vault	∅
			5
5	methodName равен "getVaultStateSignal"	Возврат указателя на метод getVaultStateSignal класса Vault	∅
			6
6	methodName равен	Возврат указателя на метод	∅

№	Предикат	Действия	№ перехода
	"getCurrentCellNumSignal"	getCurrentCellNumSignal класса Vault	
			7
7	methodName равен "changeVaultCellStateSignal"	Возврат указателя на метод changeVaultCellStateSignal класса Vault	∅
			8
8	methodName равен "setVaultStateSignal"	Возврат указателя на метод setVaultStateSignal класса Vault	∅
			9
9	methodName равен "setCurrentCellNumSignal"	Возврат указателя на метод setCurrentCellNumSignal класса Vault	∅
		Возврат nullptr	∅

3.18 Алгоритм метода getHandler класса System

Функционал: метод, возвращающий обработчик объекта.

Параметры: string methodName - наименование метода.

Возвращаемое значение: TYPE_HANDLER - указатель на метод обработчика целевого объекта.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода getHandler класса System

№	Предикат	Действия	№ перехода
1	methodName равен "getCinHandler"	Возврат указателя на метод getCinHandler класса InfoNCommandReader	∅
			2
2	methodName равен "vaultSizeHandler"	Возврат указателя на метод vaultSizeHandler класса Vault	∅
			3
3	methodName равен	Возврат указателя на метод vaultCellCreateHandler	∅

№	Предикат	Действия	№ перехода
	"vaultCellCreateHandler"	класса Vault	
			4
4	methodName равен "createMassives"	Возврат указателя на метод createMassivesHandler класса Server	∅
			5
5	methodName равен "setPinsHandler"	Возврат указателя на метод setPinsHandler класса Server	∅
			6
6	methodName равен "controllerDisplayHandler"	Возврат указателя на метод controllerDisplayHandler класса ControllerDisplay	∅
			7
7	methodName равен "vaultInfoHandler"	Возврат указателя на метод vaultInfoHandler класса Controller	∅
			8
8	methodName равен "clientPinCompareHandler"	Возврат указателя на метод clientPinCompareHandler класса Server	∅
			9
9	methodName равен "bankPinCompareHandler"	Возврат указателя на метод bankPinCompareHandler класса Server	∅
		Возврат nullptr	∅

3.19 Алгоритм конструктора класса InfoNCommandReader

Функционал: параметризированный конструктор, вызывающий конструктор родительского класса ClassBase.

Параметры: ClassBase* objHead - головной объект, string objName - имя объекта.

Алгоритм конструктора представлен в таблице 20.

Таблица 20 – Алгоритм конструктора класса InfoNCommandReader

№	Предикат	Действия	№ перехода
1		Вызов конструктора родительского класса ClassBase	∅

3.20 Алгоритм метода getCinSignal класса InfoNCommandReader

Функционал: метод сигнала, считывающий строку.

Параметры: string& info - адрес на строку с поступающей информацией.

Возвращаемое значение: Без возвращаемого значения.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода getCinSignal класса InfoNCommandReader

№	Предикат	Действия	№ перехода
1		Объявление переменной типа string с наименованием line	2
2		Считывание строки и её запись в line	3
3		Присваивание info значения переменной line	∅

3.21 Алгоритм метода getCinHandler класса InfoNCommandReader

Функционал: метод обработчика, возвращающий входящие данные.

Параметры: string info - строка с поступающей информацией.

Возвращаемое значение: string - обработанное значение.

Алгоритм метода представлен в таблице 22.

Таблица 22 – Алгоритм метода getCinHandler класса InfoNCommandReader

№	Предикат	Действия	№ перехода
1		Возврат info	∅

3.22 Алгоритм конструктора класса Controller

Функционал: параметризированный конструктор, вызывающий конструктор родительского класса ClassBase.

Параметры: ClassBase* objHead - головной объект, string objName - имя объекта.

Алгоритм конструктора представлен в таблице 23.

Таблица 23 – Алгоритм конструктора класса Controller

№	Предикат	Действия	№ перехода
1		Вызов конструктора родительского класса ClassBase	∅

3.23 Алгоритм метода commandSignal класса Controller

Функционал: метод, принимающий строку с командой, обрабатывающий командуи вырабатывающий сигнал для вывода.

Параметры: string& commandLine - адрес на строку с поступающей информацией.

Возвращаемое значение: Без возвращаемого значения.

Алгоритм метода представлен в таблице 24.

Таблица 24 – Алгоритм метода commandSignal класса Controller

№	Предикат	Действия	№ перехода
1		Инициализация переменной типа ClassBase* с наименованием ControllerObj и значением текущего объекта	2
2		Вызов метода findObjByCoordinate с параметрами: результат вызова функции getObjHead, "/Vault" и запись результата в инициализированную переменную типа ClassBase* с наименованием	3

№	Предикат	Действия	№ перехода
		VaultObj	
3	commandLine равен "Turn off the safe"	Возврат	∅
			4
4	commandLine равен "CANCEL"	Вызов метода emitSignal объекта VaultObj с параметрами: "setVaultState", "is ready to work"	5
			7
5		Присваивание commandLine значения "Ready to work"	6
6		Возврат	∅
7	commandLine равен "SHOWTREE"	Вызов метода printTreeAndState с параметром 0 объекта, что определяется результатом вызова метода getObjHead текущего объекта	8
			9
8		Возврат	∅
9		Объявление переменной типа string с наименованием command Объявление переменной типа string с наименованием value	10
10		Инициализация переменной типа size_t с наименованием pos и значением результата поиска " " в commandLine	11
11		Инициализация переменной типа string с наименованием commandLineDup и значением commandLine	12
12		Присваивание переменной command результата удаления из commandLine символов начиная с pos и до конца	13
13		Присваивание переменной value результата	14

№	Предикат	Действия	№ перехода
		удаления из commandLineDup символов начиная с 0 и до pos + 1	
14	command равен "BOX"		15
			23
15	Возврат вызова метода emitSignal("GetVaultState") объекта VaultObj не равен "is ready to work"	Присваивание commandLine значения "Error in the command sequence\nReady to work"	16
			17
16		Возврат	∅
17	Возврат вызова метода emitSignal("GetVaultCellState", value) объекта VaultObj равен "is opened"	Присваивание commandLine значения "The safe deposit box is open\nReady to work"	18
			19
18		Возврат	∅
19		Вызов метода emitSignal объекта VaultObj с параметрами: "setVaultState", "is waiting client PIN"	20
20		Вызов метода emitSignal объекта VaultObj с параметрами: "SetCurrentCellNum", value	21
21		Присваивание commandLine значения "Enter the code"	22
22		Возврат	∅
23	command равно "CLIENT_KEY"		24
			33
24	Возврат вызова метода emitSignal("GetVaultState") объекта VaultObj не равен "is	Присваивание commandLine значения "Error in the command sequence\nReady to work"	25

№	Предикат	Действия	№ перехода
	waiting client PIN"		
			27
25		Вызов метода emitSignal объекта VaultObj с параметрами: "setVaultState", "is ready to work"	26
26		Возврат	∅
27	Возврат вызова метода emitSignal("CompareClientPin", VaultObj->emitSignal("GetCurrentCellNum") + " " + value) объекта ControllerObj равен "clientPinInCorrect"	Присваивание commandLine значения "The client is key is incorrect\nReady to work"	28
			30
28		Вызов метода emitSignal объекта VaultObj с параметрами: "setVaultState", "is ready to work"	29
29		Возврат	∅
30		Вызов метода emitSignal объекта VaultObj с параметрами: "setVaultState", "is waiting bank PIN"	31
31		Присваивание commandLine значения "Enter the bank code"	32
32		Возврат	∅
33	command равно "BANK_KEY"		34
			44
34	Возврат вызова метода emitSignal("GetVaultState") объекта VaultObj не равен "is waiting bank PIN"	Присваивание commandLine значения "Error in the command sequence\nReady to work"	35
			37

№	Предикат	Действия	№ перехода
35		Вызов метода emitSignal объекта VaultObj с параметрами: "setVaultState", "is ready to work"	36
36		Возврат	∅
37	Возврат вызова метода emitSignal("CompareBankPin", VaultObj->emitSignal("GetCurrentCellNum") + " " + value) объекта ControllerObj равен "bankPinInCorrect"	Присваивание commandLine значения "The bank is key is incorrect\nReady to work"	38
			40
38		Вызов метода emitSignal объекта VaultObj с параметрами: "setVaultState", "is ready to work"	39
39		Возврат	∅
40		Вызов метода emitSignal объекта VaultObj с параметрами: "ChangeVaultCellState", результат вызова метода emitSignal объекта VaultObj с параметром "GetCurrentCellNum"	41
41		Вызов метода emitSignal объекта VaultObj с параметрами: "setVaultState", "is ready to work"	42
42		Вызов метода emitSignal объекта VaultObj с параметром "GetCurrentCellNum" и участие результата вызовы метода в формировании значения commandLine "The safe deposit box " + результат работы метода + " is open\nReady to work"	43
43		Возврат	∅
44	command равно "CLOSE_BOX"		45

№	Предикат	Действия	№ перехода
			∅
45	Возврат вызова метода emitSignal("GetVaultState") объекта VaultObj не равен "is ready to work"	Присваивание commandLine значения "Error in the command sequence\nReady to work"	46
			48
46		Вызов метода emitSignal объекта VaultObj с параметрами: "setVaultState", "is ready to work"	47
47		Возврат	∅
48		Вызов метода emitSignal объекта VaultObj с параметрами: "ChangeVaultCellState", результат вызова метода emitSignal объекта VaultObj с параметром "GetCurrentCellNum"	49
49		Вызов метода emitSignal объекта VaultObj с параметрами: "setVaultState", "is ready to work"	50
50		Присваивание commandLine значения "Ready to work"	51
51		Возврат	∅

3.24 Алгоритм метода vaultInfoHandler класса Controller

Функционал: метод обработчика, возвращающий входящие данные.

Параметры: string info - строка с поступающей информацией.

Возвращаемое значение: string - обработанное значение.

Алгоритм метода представлен в таблице 25.

Таблица 25 – Алгоритм метода vaultInfoHandler класса Controller

№	Предикат	Действия	№ перехода
1		Возврат info	∅

3.25 Алгоритм конструктора класса Vault

Функционал: параметризированный конструктор, вызывающий конструктор родительского класса ClassBase.

Параметры: ClassBase* objHead - головной объект, string objName - имя объекта.

Алгоритм конструктора представлен в таблице 26.

Таблица 26 – Алгоритм конструктора класса Vault

№	Предикат	Действия	№ перехода
1		Вызов конструктора родительского класса ClassBase	∅

3.26 Алгоритм метода vaultSizeHandler класса Vault

Функционал: метод обработчика, записывающий количество ячеек, рядов и столбцов.

Параметры: string vaultSize - строка с поступающей информацией.

Возвращаемое значение: string - обработанное значение.

Алгоритм метода представлен в таблице 27.

Таблица 27 – Алгоритм метода vaultSizeHandler класса Vault

№	Предикат	Действия	№ перехода
1		Инициализация переменной типа size_t с наименованием pos и значением результат поиска " " в vaultSize	2
2		Инициалищация переменной типа string с наименованием vaultSizeDup и значением vaultSize	3
3		Инициализация переменной типа int с наименованием rowAmount и значением равным переводу из формата string в формат int значения после удаления из vaultSize символов начиная с 0 и до pos	4
4		Инициализация переменной типа int с наименованием columnAmount	5

№	Предикат	Действия	№ перехода
		и значением равным переводу из формата string в формат int значения после удаления из vaultSizeDup символов начиная с pos и до конца	
5		Присваивание cellAmount текущего объекта значения произведения rowAmount и columnAmount Присваивание rowAmount текущего объекта значения rowAmount Присваивание columnAmount текущего объекта значения columnAmount	6
6		Возврат "null"	∅

3.27 Алгоритм метода vaultCellCreateHandler класса Vault

Функционал: метод обработчика, создающий ячейки сейфа.

Параметры: string vaultSize - строка с поступающей информацией.

Возвращаемое значение: string - обработанное значение.

Алгоритм метода представлен в таблице 28.

Таблица 28 – Алгоритм метода vaultCellCreateHandler класса Vault

№	Предикат	Действия	№ перехода
1		Объявление переменной типа ClassBase* с наименованием Obj	2
2		Присваивание cellMas нового массива типа CLassBase* и размером cellAmount	3
3		Инициализация счетчика i с начальным значением 0	4
4	i меньше stoi(vaultSize)	Присваивание Obj нового объекта класса VaultCell	5
			7
5		Присваивание ячейке i массива cellMas значения Obj	6
6		Увеличение счетчика i на единицу	4

№	Предикат	Действия	№ перехода
7		Возврат "null"	∅

3.28 Алгоритм метода getCellAmountSignal класса Vault

Функционал: метод сигнала, записывающий в адрес передаваемой строки количество ячеек сейфа.

Параметры: string& info - адрес на строку с поступающей информацией.

Возвращаемое значение: Без возвращаемого значения.

Алгоритм метода представлен в таблице 29.

Таблица 29 – Алгоритм метода getCellAmountSignal класса Vault

№	Предикат	Действия	№ перехода
1		Присваивание info значения cellAmount переведенного в формат string	∅

3.29 Алгоритм метода getVaultCellStateSignal класса Vault

Функционал: метод сигнала, записывающий в адрес передаваемой строки состояние ячейки.

Параметры: string& info - адрес на строку с поступающей информацией.

Возвращаемое значение: Без возвращаемого значения.

Алгоритм метода представлен в таблице 30.

Таблица 30 – Алгоритм метода getVaultCellStateSignal класса Vault

№	Предикат	Действия	№ перехода
1		Инициализация переменной типа int с наименованием cellNum и значением info переведенного в формат int	2
2	Возврат вызова метода	Присваивание info значения "is closed"	3

№	Предикат	Действия	№ перехода
	getObjState() объекта cellMas[cellNum - 1] равен 1		
			3
3	Возврат вызова метода getObjState() объекта cellMas[cellNum - 1] равен 2	Присваивание info значения "is opened"	∅
			∅

3.30 Алгоритм метода getVaultStateSignal класса Vault

Функционал: метод сигнала, записывающий в адрес передаваемой строки состояние сейфа.

Параметры: string& info - адрес на строку с поступающей информацией.

Возвращаемое значение: Без возвращаемого значения.

Алгоритм метода представлен в таблице 31.

Таблица 31 – Алгоритм метода getVaultStateSignal класса Vault

№	Предикат	Действия	№ перехода
1	Возврат вызова метода getObjState() текущего объекта равен 1	Присваивание info значения "is ready to work"	2
			2
2	Возврат вызова метода getObjState() текущего объекта равен 2	Присваивание info значения "is waiting client PIN"	3
			3
3	Возврат вызова метода getObjState() текущего	Присваивание info значения "is waiting bank PIN"	∅

№	Предикат	Действия	№ перехода
	объекта равен 3		
			∅

3.31 Алгоритм метода getCurrentCellNumSignal класса Vault

Функционал: метод сигнала, записывающий в адрес передаваемой строки номер открываемой ячейки.

Параметры: string& info - адрес на строку с поступающей информацией.

Возвращаемое значение: Без возвращаемого значения.

Алгоритм метода представлен в таблице 32.

Таблица 32 – Алгоритм метода getCurrentCellNumSignal класса Vault

№	Предикат	Действия	№ перехода
1		Присваивание info значения currentCellNum	∅

3.32 Алгоритм метода changeVaultCellStateSignal класса Vault

Функционал: метод сигнала, изменяющий состояние ячейки.

Параметры: string& info - адрес на строку с поступающей информацией.

Возвращаемое значение: Без возвращаемого значения.

Алгоритм метода представлен в таблице 33.

Таблица 33 – Алгоритм метода changeVaultCellStateSignal класса Vault

№	Предикат	Действия	№ перехода
1		Инициализация переменной типа int с наименованием cellNum и значением info переведенного в формат int	2
2	Возврат вызова метода	Вызов метода setObjState с параметром 2 объекта	∅

№	Предикат	Действия	№ перехода
	getObjState() объекта cellMas[cellNum - 1] равен 1	cellMas[cellNum - 1]	
			3
3	Возврат вызова метода getObjState() объекта cellMas[cellNum - 1] равен 2	Вызов метода setObjState с параметром 1 объекта cellMas[cellNum - 1]	∅
			∅

3.33 Алгоритм метода setVaultStateSignal класса Vault

Функционал: метод сигнала, изменяющий состояние сейфа.

Параметры: string& info - адрес на строку с поступающей информацией.

Возвращаемое значение: Без возвращаемого значения.

Алгоритм метода представлен в таблице 34.

Таблица 34 – Алгоритм метода setVaultStateSignal класса Vault

№	Предикат	Действия	№ перехода
1	info равно "is ready to work"	Вызов метода setObjState с параметром 1 текущего объекта	2
			2
2	info равно "is waiting client PIN"	Вызов метода setObjState с параметром 2 текущего объекта	3
			3
3	info равно "is waiting bank PIN"	Вызов метода setObjState с параметром 3 текущего объекта	∅
			∅

3.34 Алгоритм метода setCurrentCellNumSignal класса Vault

Функционал: метод сигнала, изменяющий номер открываемой ячейки.

Параметры: string& info - адрес на строку с поступающей информацией.

Возвращаемое значение: Без возвращаемого значения.

Алгоритм метода представлен в таблице 35.

Таблица 35 – Алгоритм метода setCurrentCellNumSignal класса Vault

№	Предикат	Действия	№ перехода
1		Присваивание currentCellNum значения info	∅

3.35 Алгоритм конструктора класса VaultCell

Функционал: параметризированный конструктор, вызывающий конструктор родительского класса ClassBase и устанавливающий номер ячейки.

Параметры: ClassBase* objHead - головной объект, string objName - имя объекта, int cellNumber - номер ячейки.

Алгоритм конструктора представлен в таблице 36.

Таблица 36 – Алгоритм конструктора класса VaultCell

№	Предикат	Действия	№ перехода
1		Вызов конструктора родительского класса ClassBase	2
2		Присваивание cellNumber текущего объекта значения cellNumber	∅

3.36 Алгоритм конструктора класса Server

Функционал: параметризированный конструктор, вызывающий конструктор родительского класса ClassBase.

Параметры: ClassBase* objHead - головной объект, string objName - имя объекта..

Алгоритм конструктора представлен в таблице 37.

Таблица 37 – Алгоритм конструктора класса Server

№	Предикат	Действия	№ перехода
1		Вызов конструктора родительского класса ClassBase	∅

3.37 Алгоритм метода createMassivesHandler класса Server

Функционал: метод обработчика, создающий массивы ПИН-кодов клиента и банка.

Параметры: string vaultSize - строка с поступающей информацией.

Возвращаемое значение: string - обработанное значение.

Алгоритм метода представлен в таблице 38.

Таблица 38 – Алгоритм метода createMassivesHandler класса Server

№	Предикат	Действия	№ перехода
1		Присваивание переменной clientPinMas нового массива типа string с количеством элементов vaultSize, переведенный из формата string в формат int	2
2		Присваивание переменной bankPinMas нового массива типа string с количеством элементов vaultSize, переведенный из формата string в формат int	3
3		Возврат "null"	∅

3.38 Алгоритм метода setPinsHandler класса Server

Функционал: метод обработчика, заполняющий массивы ПИН-кодов клиента и банка.

Параметры: string vaultSize - строка с поступающей информацией.

Возвращаемое значение: string - обработанное значение.

Алгоритм метода представлен в таблице 39.

Таблица 39 – Алгоритм метода setPinsHandler класса Server

№	Предикат	Действия	№ перехода
1	info равно "Completing key entry"	Возврат "Completing key entry"	∅
			2
2		Объявление переменной типа int с наименованием cellIndex Объявление переменной типа string с наименованием clientPin Объявление переменной типа string с наименованием bankPin	3
3		Инициализация переменной типа size_t с наименованием pos и значением равным результату поиска " " в info	4
4		Инициализация переменной типа string с наименованием infoDup и значением info	5
5		Присваивание переменной cellIndex значения infoDup после удаления символов начиная с pos и до конца, переведенного из формата string в формат int	6
6		Удаление символов из info начиная с 0 и до pos + 1	7
7		Присваивание переменной pos результата поиска "" в info	8
8		Присваивание переменной infoDup значения info	9
9		Присваивание переменной clientPin значения infoDup после удаления символов начиная с pos и до конца	10
10		Удаление символов из info начиная с 0 и до pos + 1	11
11		Присваивание bankPin значения info	12

№	Предикат	Действия	№ перехода
12		Присваивание элементу cellIndex - 1 массива clientPinMas значения clientPin	13
13		Присваивание элементу cellIndex - 1 массива bankPinMas значения bankPin	14
14		Возврат "null"	∅

3.39 Алгоритм метода clientPinCompareHandler класса Server

Функционал: метод обработчика, сравнивающий входящий ПИН-код клиента и ПИН-код клиента на сервере и возвращающий результат.

Параметры: string vaultSize - строка с поступающей информацией.

Возвращаемое значение: string - обработанное значение.

Алгоритм метода представлен в таблице 40.

Таблица 40 – Алгоритм метода clientPinCompareHandler класса Server

№	Предикат	Действия	№ перехода
1		Объявление переменной типа int с наименованием cellIndex Объявление переменной типа string с наименованием clientPin	2
2		Инициализация переменной типа size_t с наименованием pos и значением равным результату поиска " " в info	3
3		Инициализация переменной типа string с наименованием infoDup и значением info	4
4		Присваивание переменной cellIndex значения infoDup после удаления символов начиная с pos и до конца, переведенного из формата string в формат int	5
5		Присваивание переменной clientPin переменной	6

№	Предикат	Действия	№ перехода
		info после удаление символов начиная с 0 и до pos + 1	
6	clientPinMas[cellIndex - 1] равно clientPin	Возврат "clientPinCorrect"	∅
		Возврат "clientPinInCorrect"	∅

3.40 Алгоритм метода bankPinCompareHandler класса Server

Функционал: метод обработчика, сравнивающий входящий ПИН-код банка и ПИН-код банка на сервере и возвращающий результат.

Параметры: string vaultSize - строка с поступающей информацией.

Возвращаемое значение: string - обработанное значение.

Алгоритм метода представлен в таблице 41.

Таблица 41 – Алгоритм метода bankPinCompareHandler класса Server

№	Предикат	Действия	№ перехода
1		Объявление переменной типа int с наименованием cellIndex Объявление переменной типа string с наименованием bankPin	2
2		Инициализация переменной типа size_t с наименованием pos и значением равным результату поиска " " в info	3
3		Инициализация переменной типа string с наименованием infoDup и значением info	4
4		Присваивание переменной cellIndex значения infoDup после удаления символов начиная с pos и до конца, переведенного из формата string в	5

№	Предикат	Действия	№ перехода
		формат int	
5		Присваивание переменной bankPin переменной info после удаление символов начиная с 0 и до pos + 1	6
6	bankPinMas[cellIndex - 1] равно bankPin	Возврат "bankPinCorrect"	∅
		Возврат "bankPinInCorrect"	∅

3.41 Алгоритм конструктора класса ControllerDisplay

Функционал: параметризированный конструктор, вызывающий конструктор родительского класса ClassBase.

Параметры: ClassBase* objHead - головной объект, string objName - имя объекта.

Алгоритм конструктора представлен в таблице 42.

Таблица 42 – Алгоритм конструктора класса ControllerDisplay

№	Предикат	Действия	№ перехода
1		Вызов конструктора родительского класса ClassBase	∅

3.42 Алгоритм метода controllerDisplayHandler класса ControllerDisplay

Функционал: метод обработчика, выводящий передаваемую строку.

Параметры: string vaultSize - строка с поступающей информацией.

Возвращаемое значение: string - обработанное значение.

Алгоритм метода представлен в таблице 43.

Таблица 43 – Алгоритм метода controllerDisplayHandler класса ControllerDisplay

№	Предикат	Действия	№ перехода
1	info равно "SHOWTREE"	Вывод << endl << "Turn off the safe"	2
			3
2		Возврат "SHOWTREE"	∅
3		Вывод << endl << info	4
4	info равно "Turn off the safe"	Возврат "Turn off the safe"	∅
			5
5		Возврат "null"	∅

3.43 Алгоритм функции main

Функционал: точка входа в программу.

Параметры: нет.

Возвращаемое значение: Целочисленное - индикатор успешности выполнения программы.

Алгоритм функции представлен в таблице 44.

Таблица 44 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Инициализация объекта класса System с наименованием SystemObj	2
2		Вызов метода buildTree объекта SystemObj	3
3		Вызов метода startApp объекта SustemObj и возврат результата	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-46.

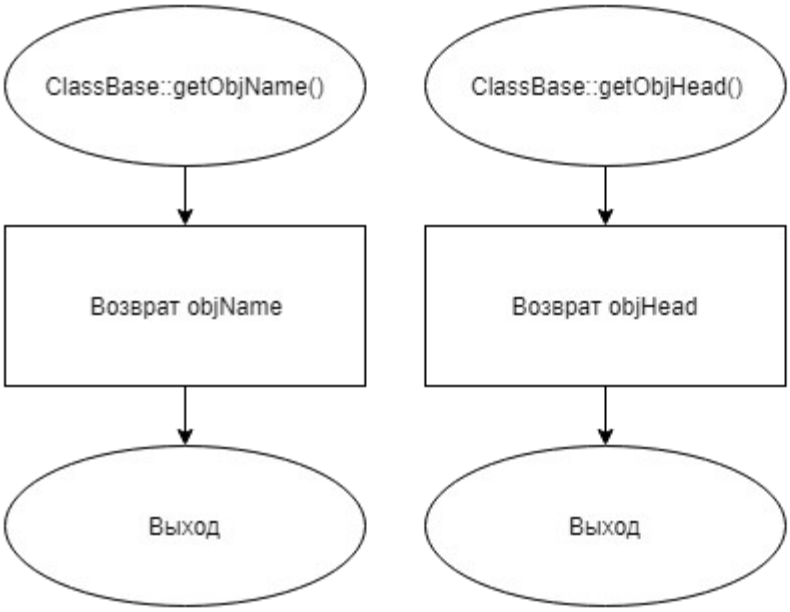

Рисунок 1 – Блок-схема алгоритма

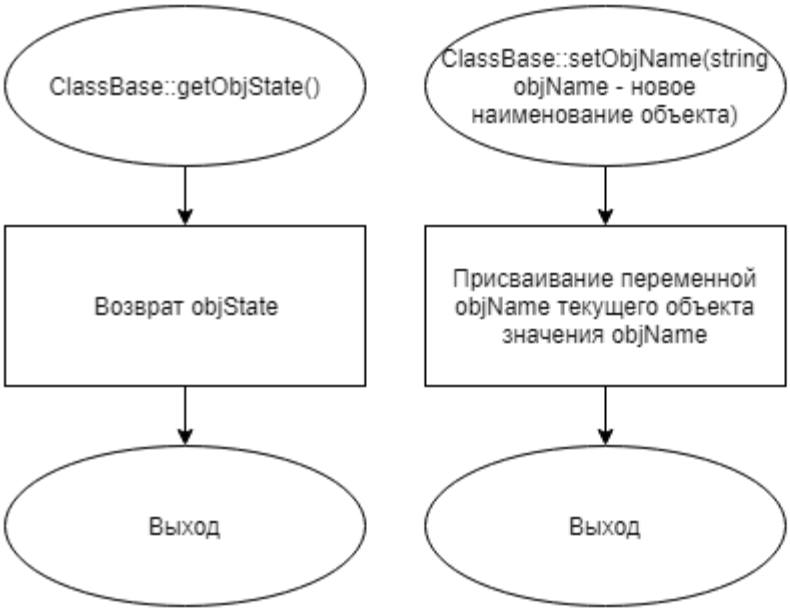

Рисунок 2 – Блок-схема алгоритма

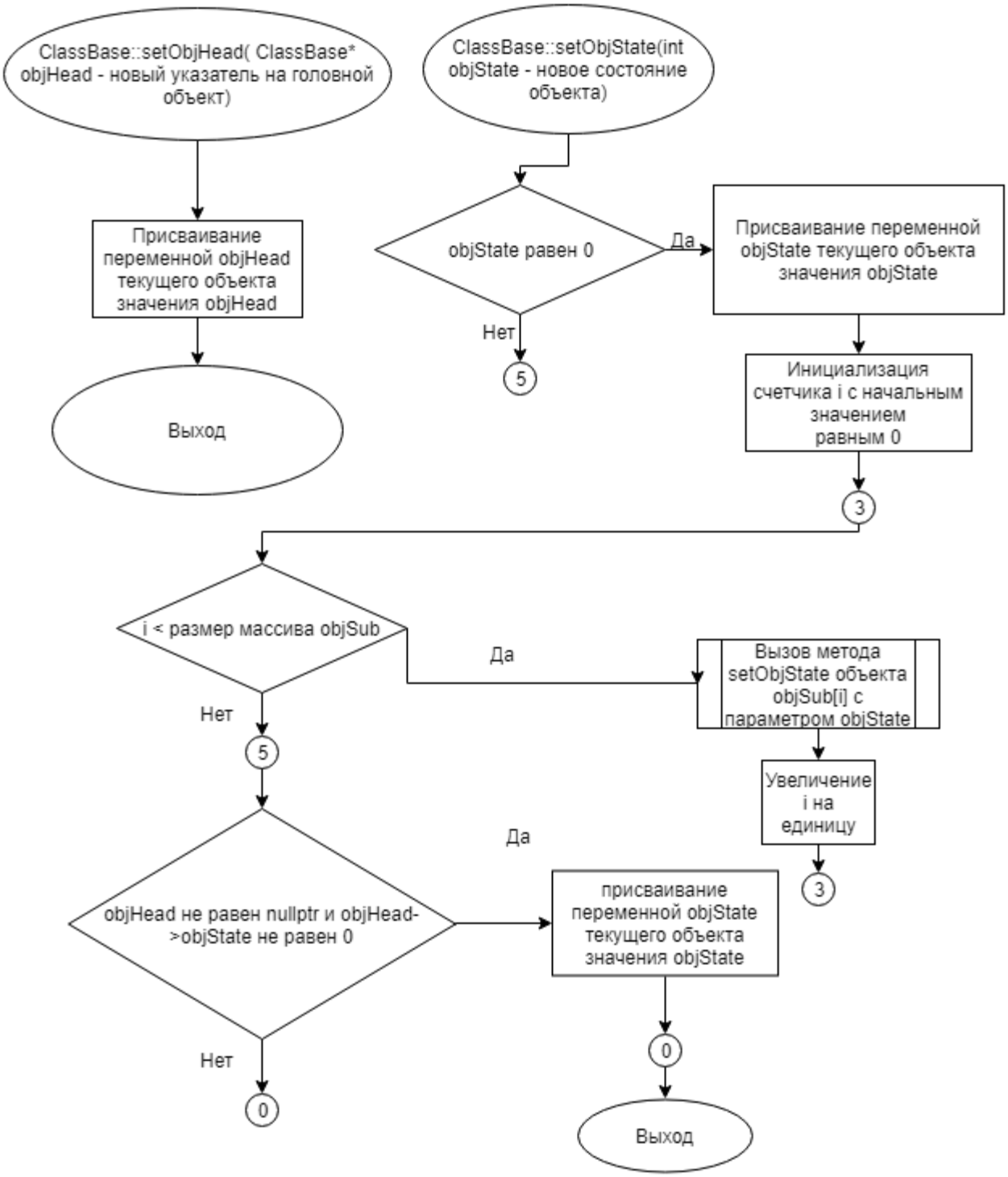

Рисунок 3 – Блок-схема алгоритма

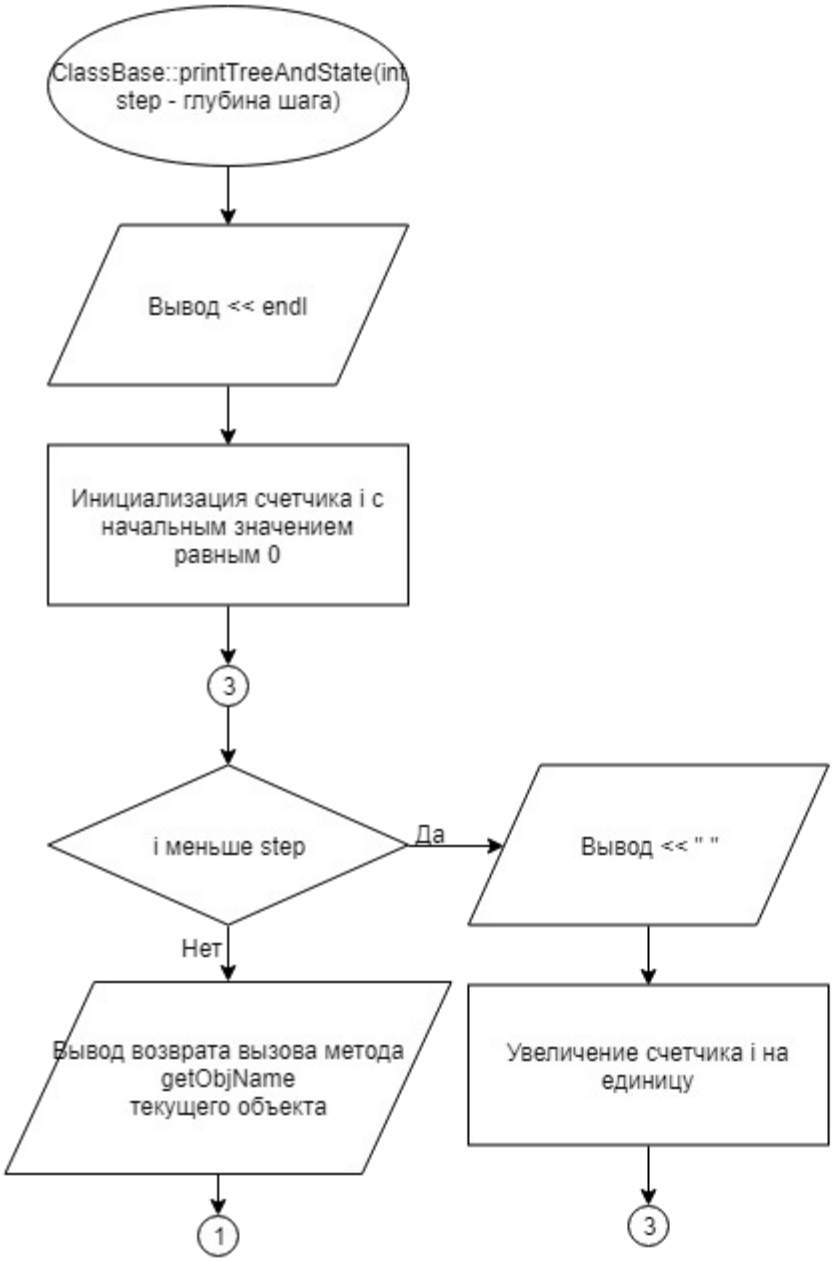

Рисунок 4 – Блок-схема алгоритма

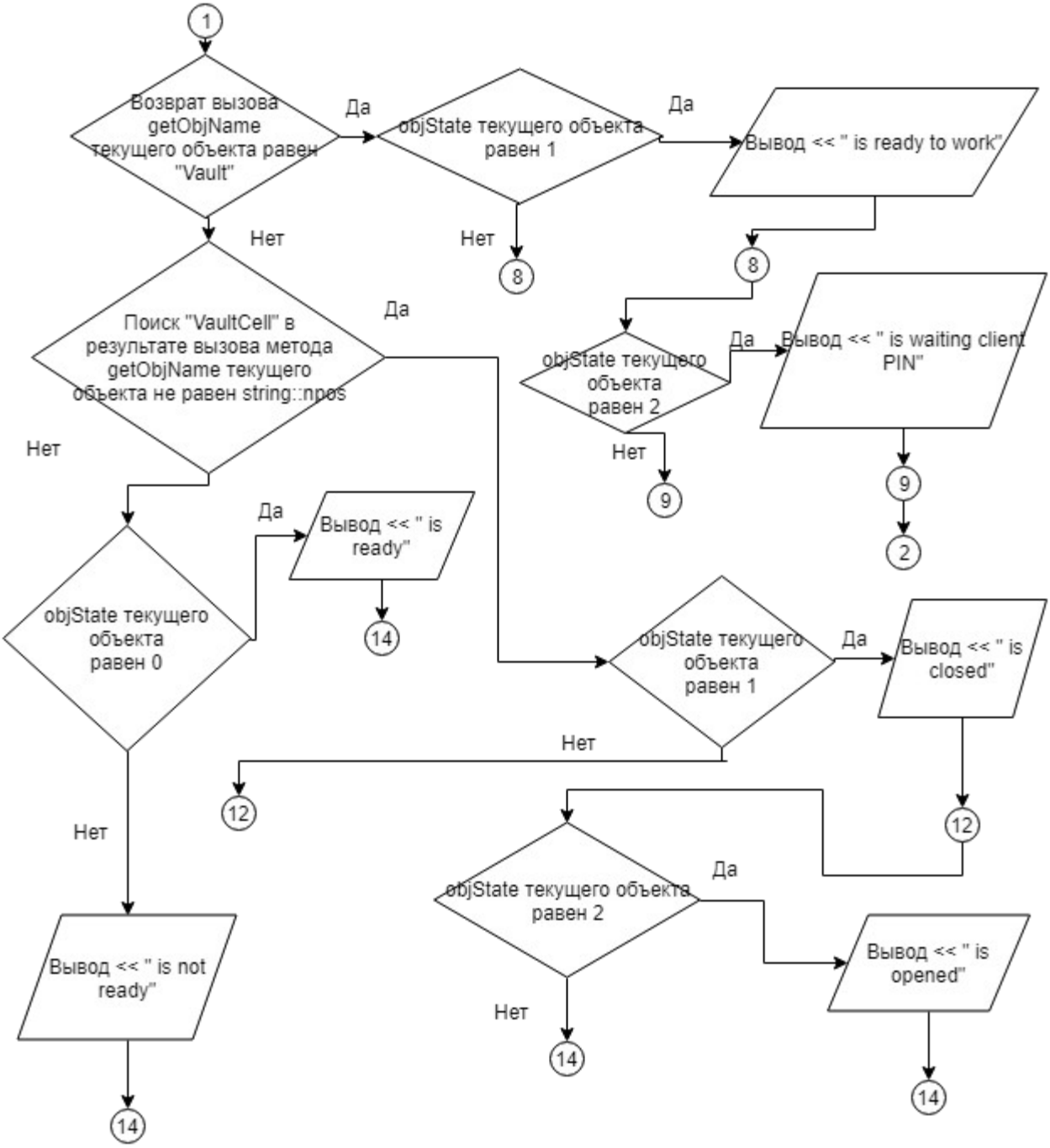

Рисунок 5 – Блок-схема алгоритма

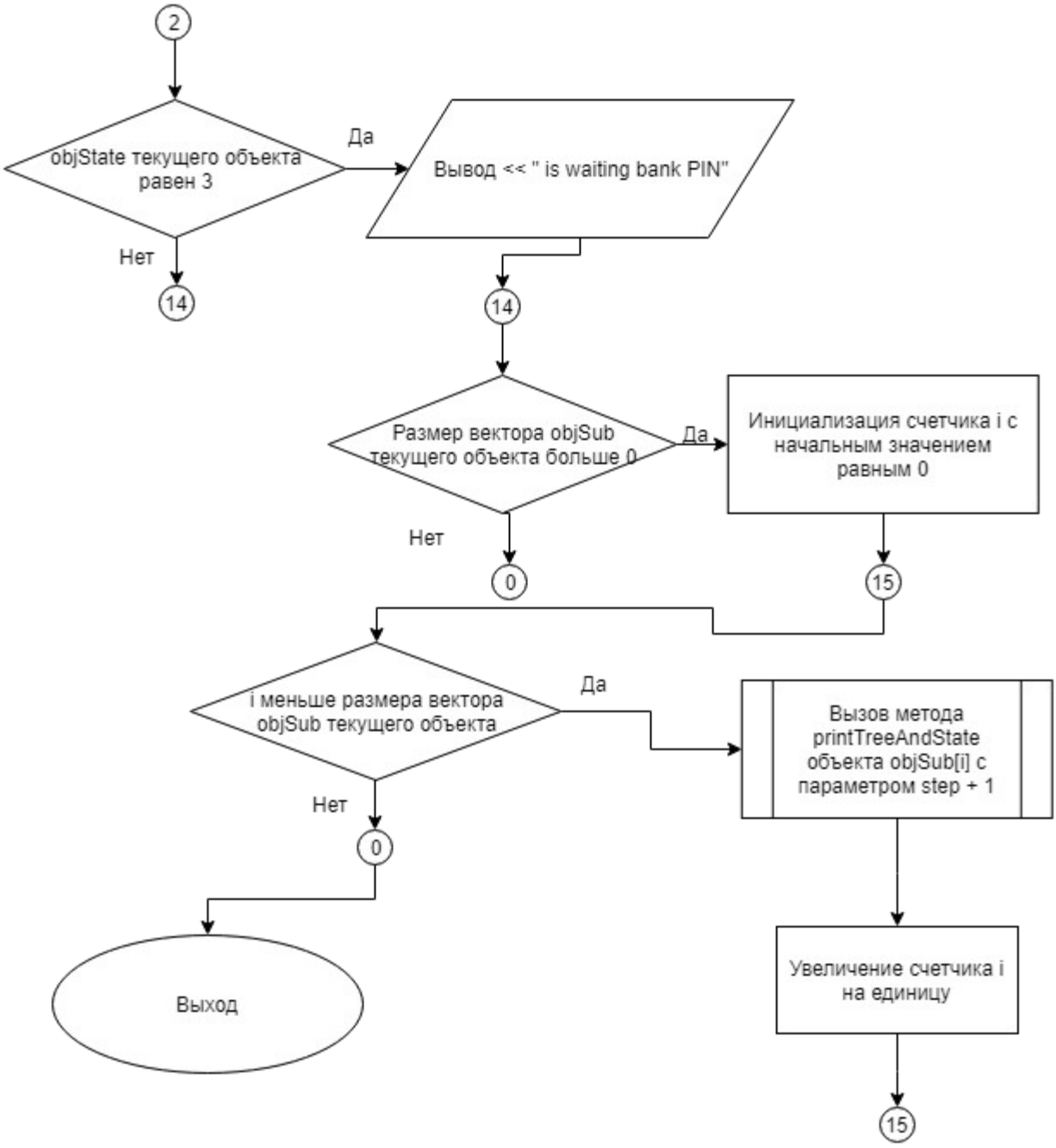

Рисунок 6 – Блок-схема алгоритма

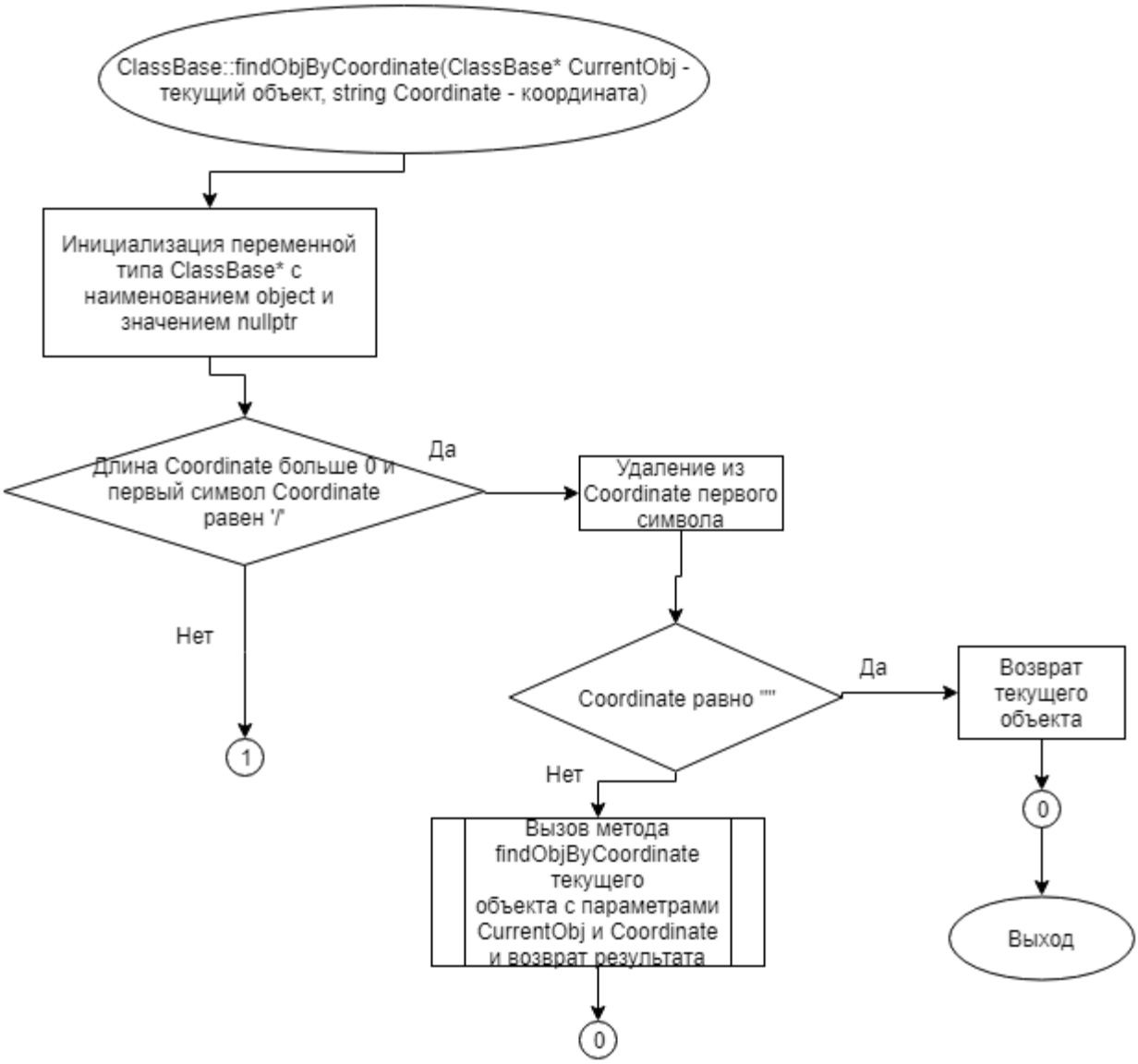

Рисунок 7 – Блок-схема алгоритма

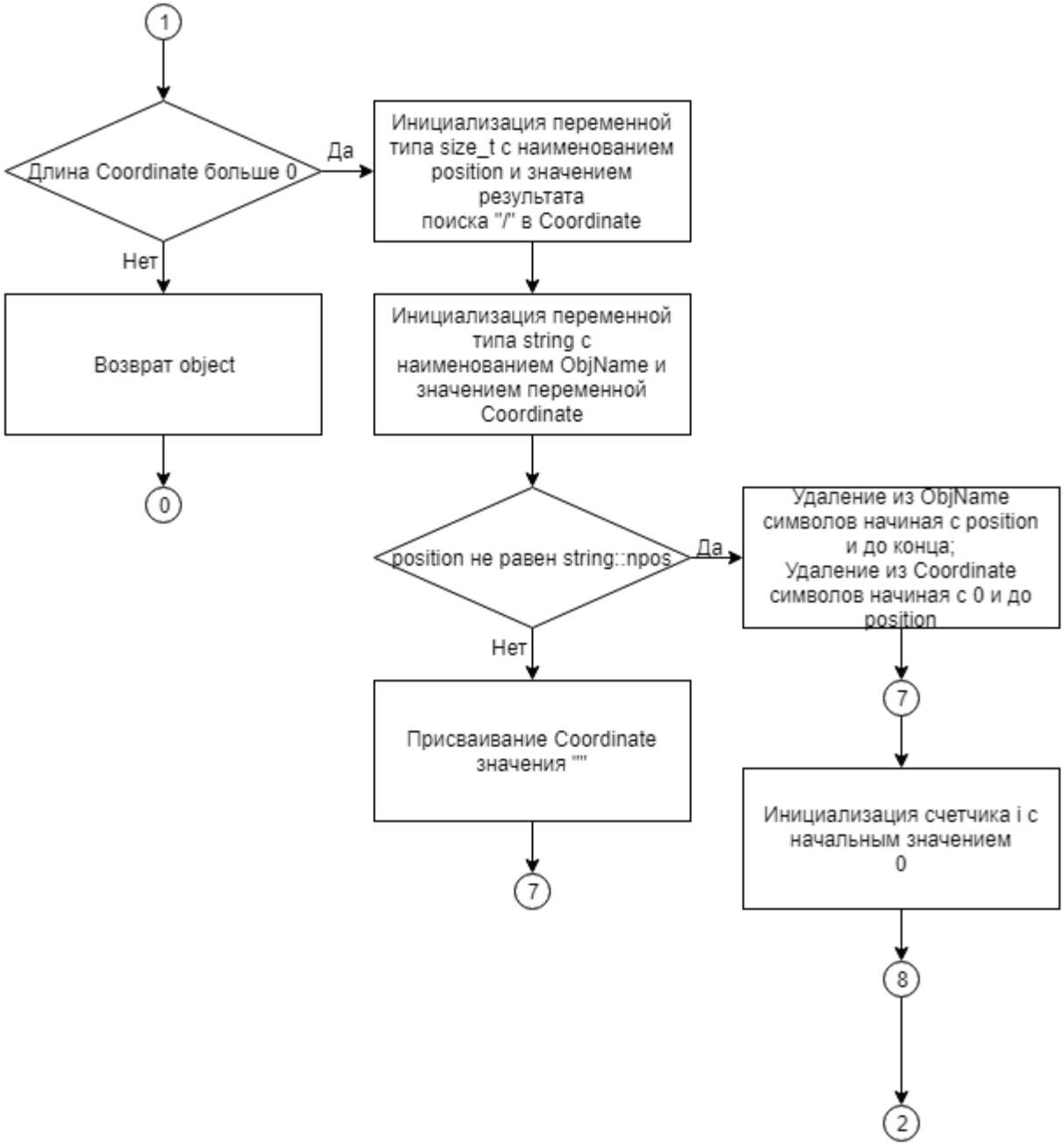

Рисунок 8 – Блок-схема алгоритма

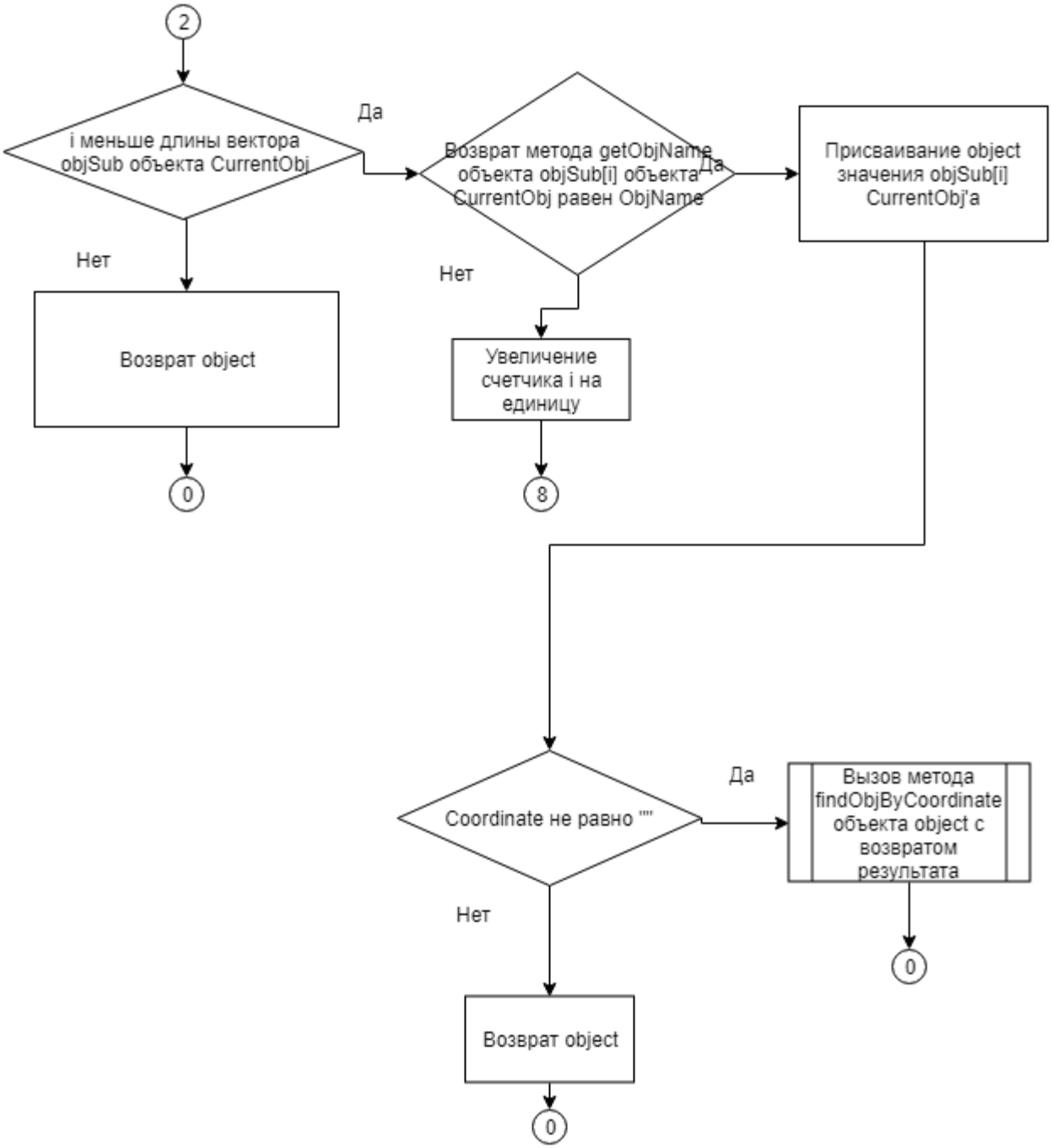

Рисунок 9 – Блок-схема алгоритма

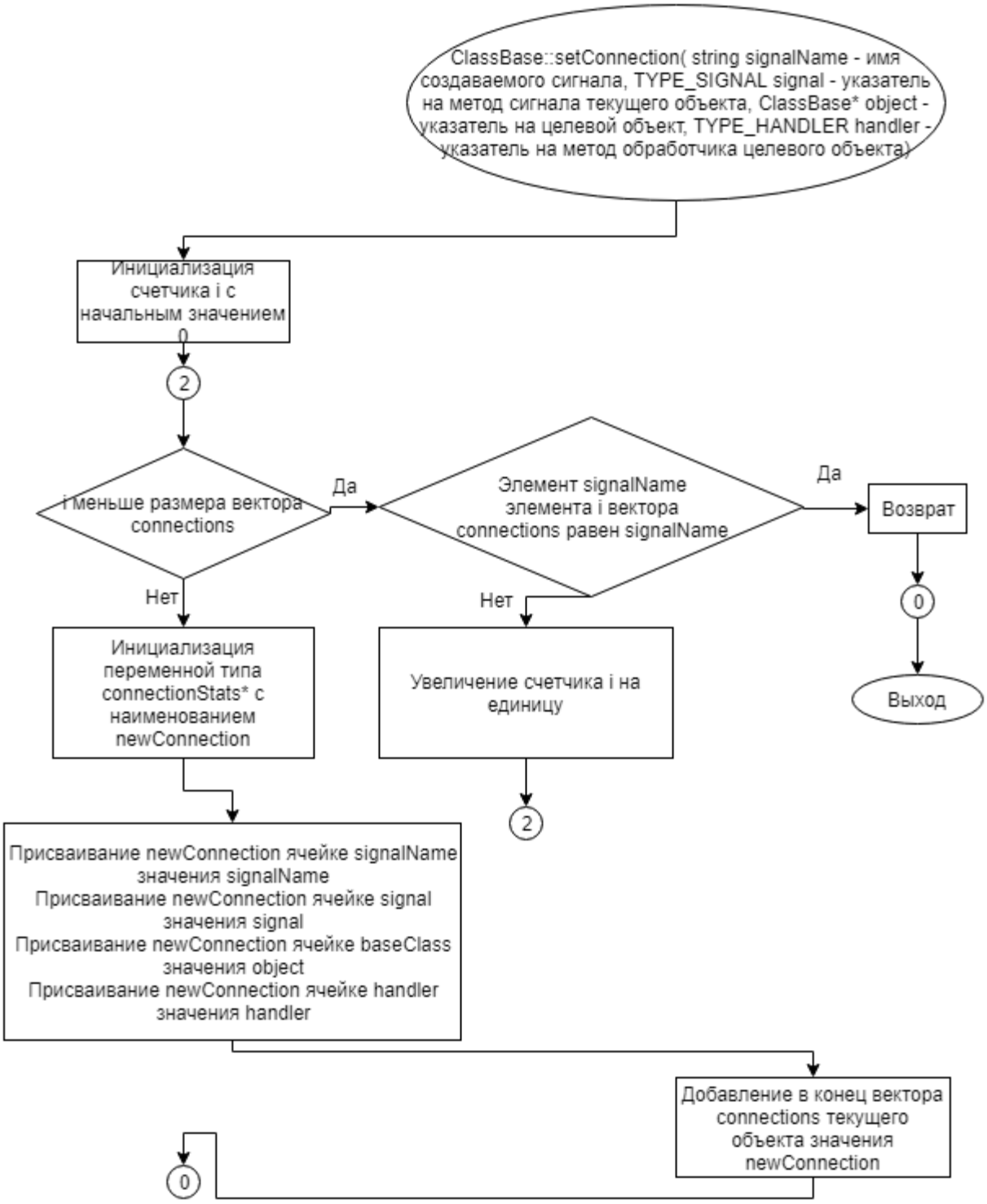

Рисунок 10 – Блок-схема алгоритма

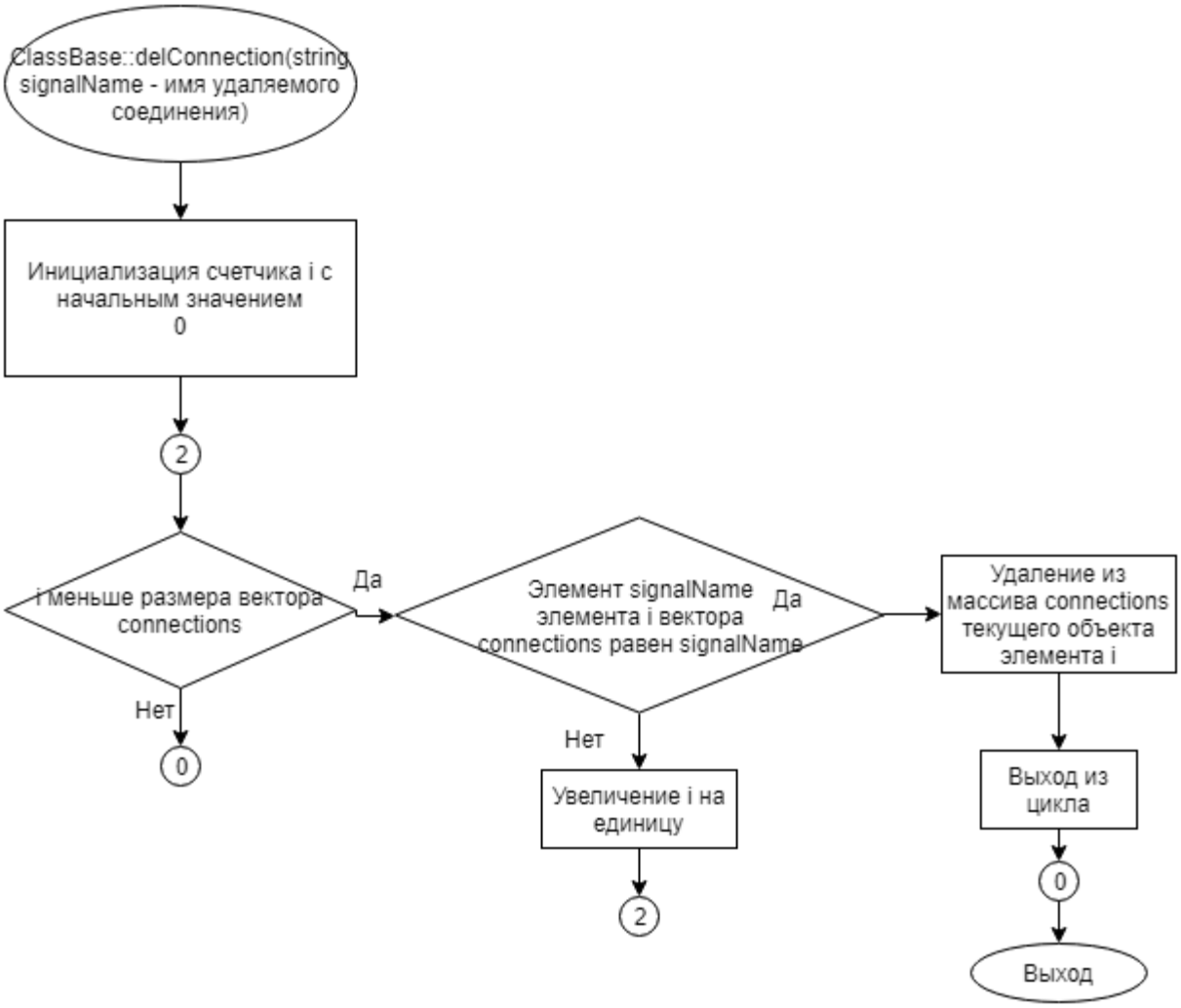

Рисунок 11 – Блок-схема алгоритма

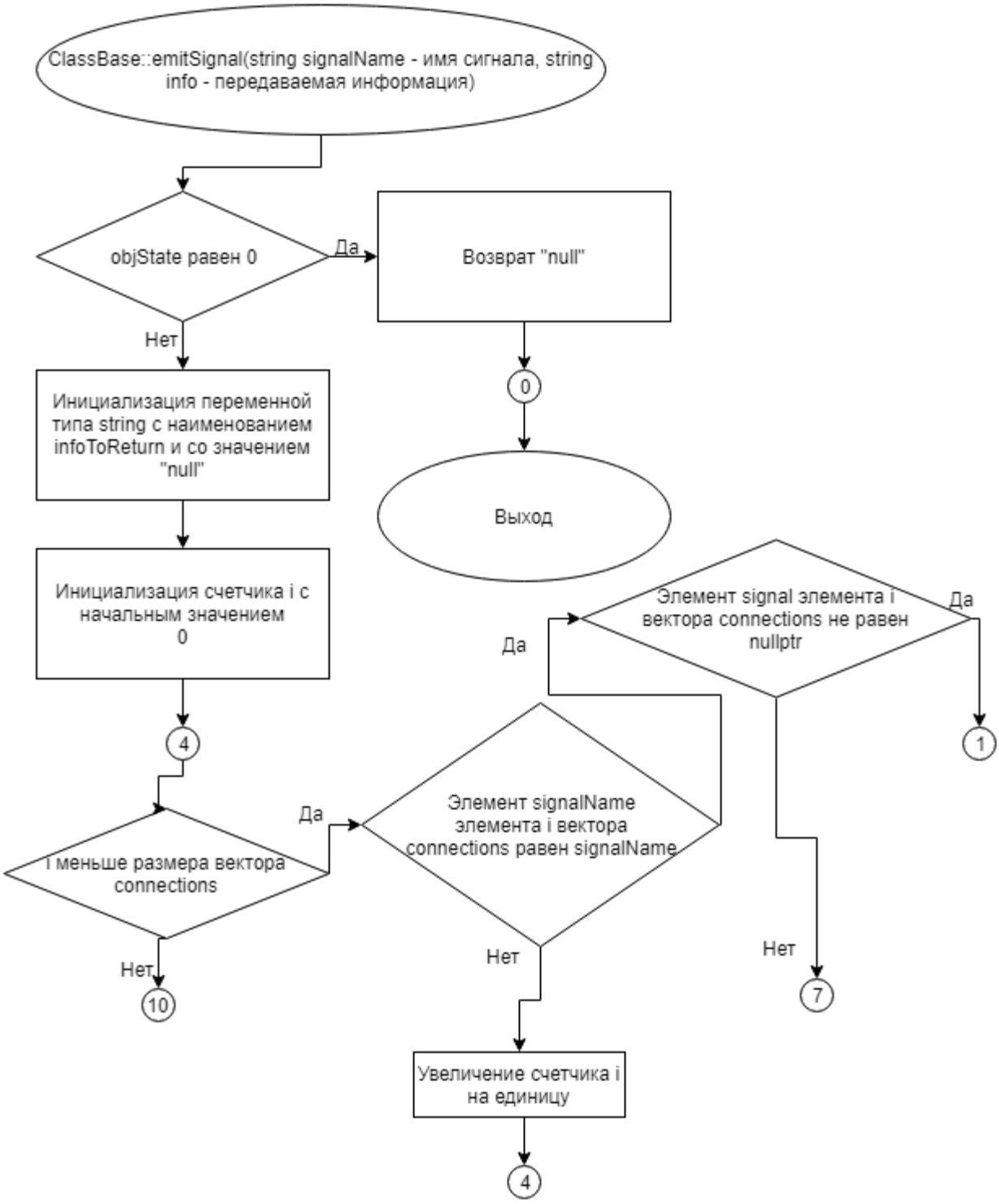

Рисунок 12 – Блок-схема алгоритма

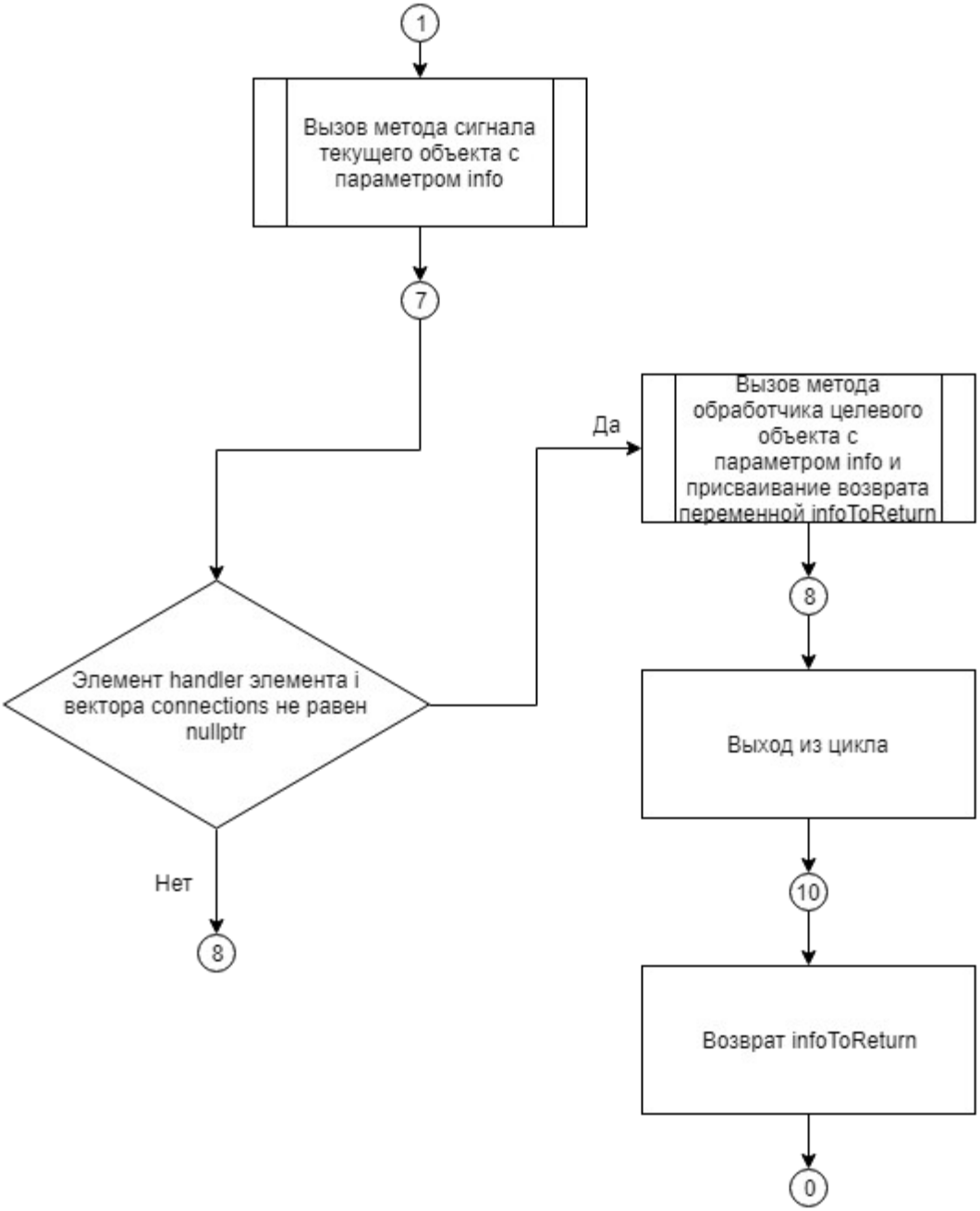

Рисунок 13 – Блок-схема алгоритма

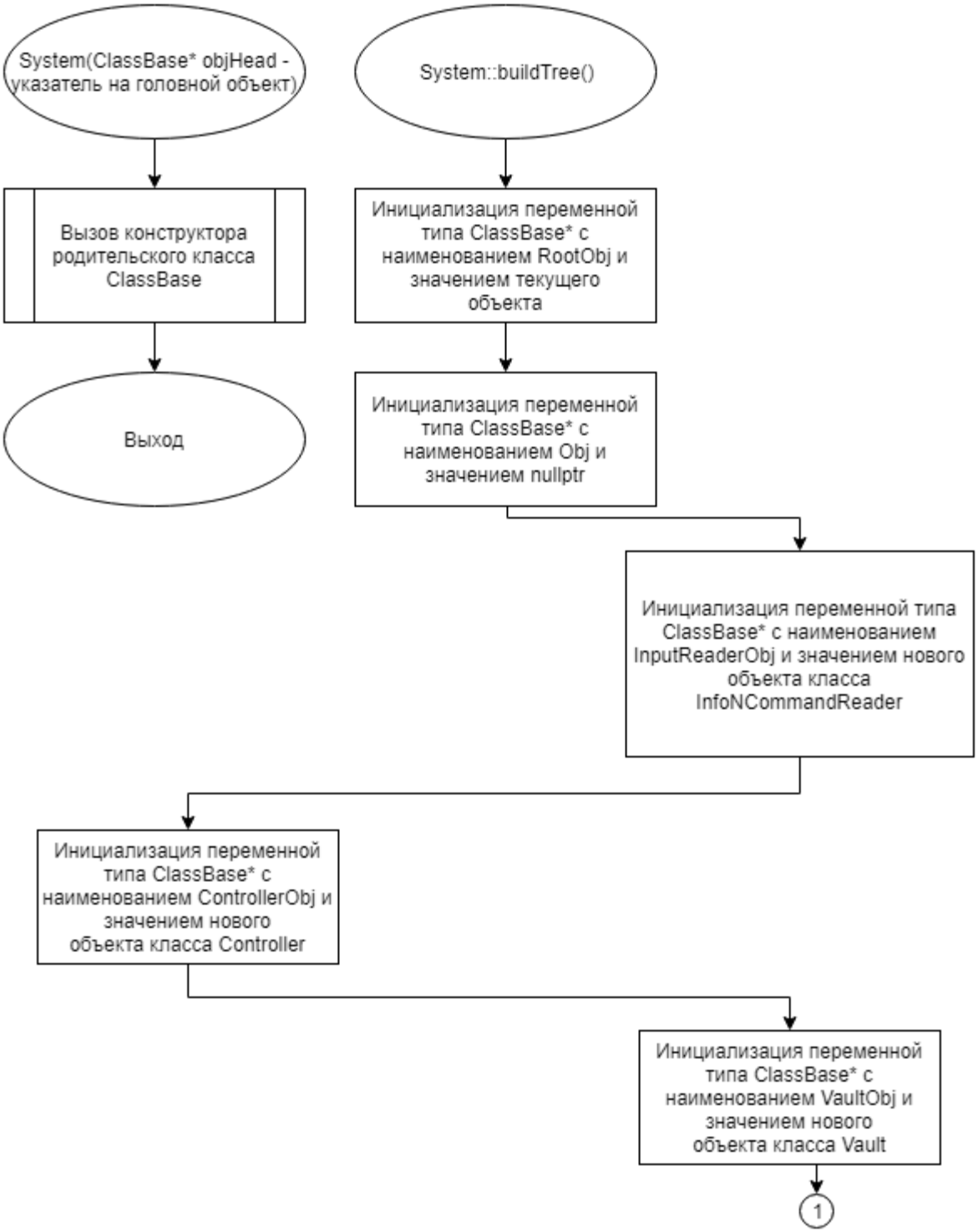

Рисунок 14 – Блок-схема алгоритма

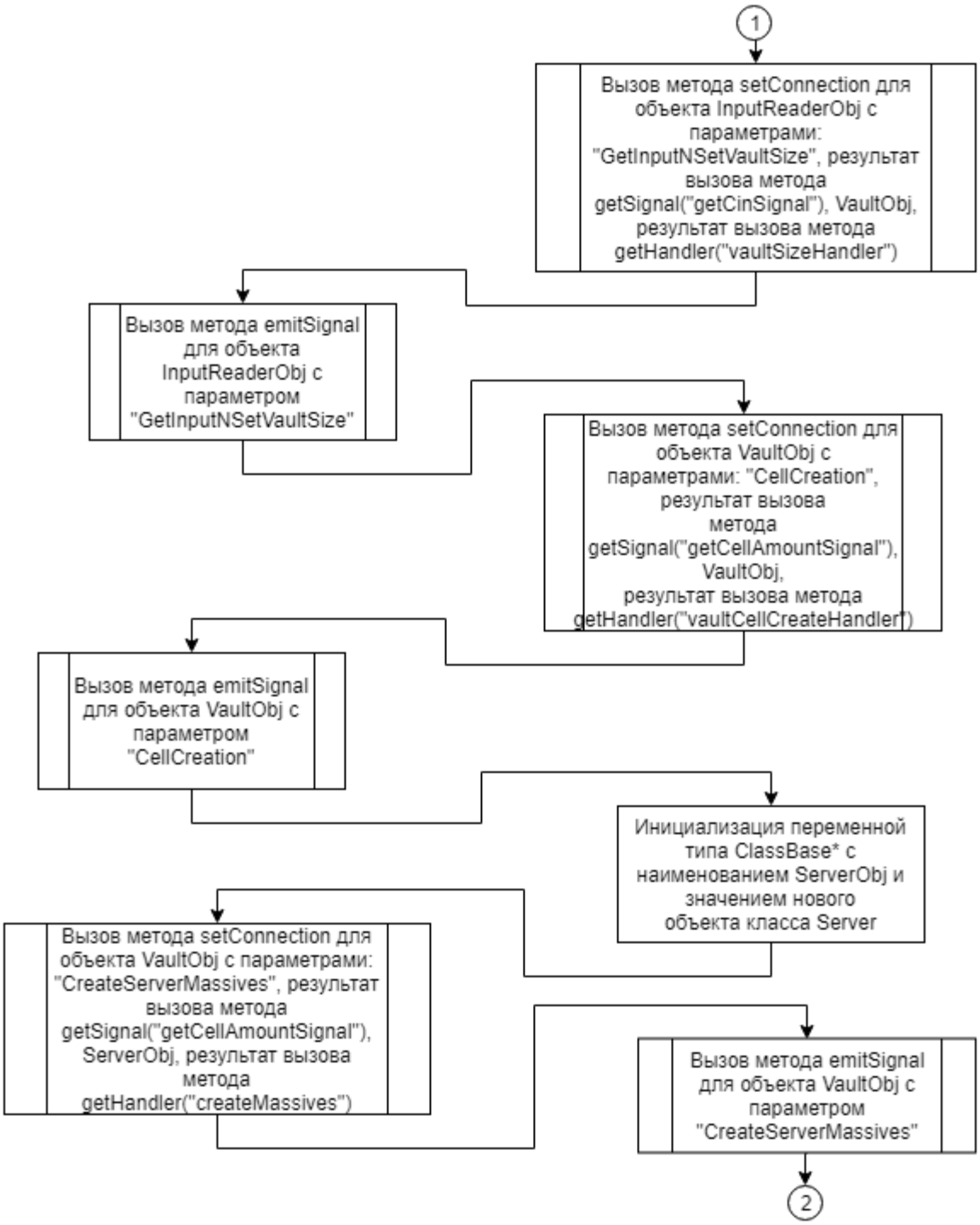

Рисунок 15 – Блок-схема алгоритма

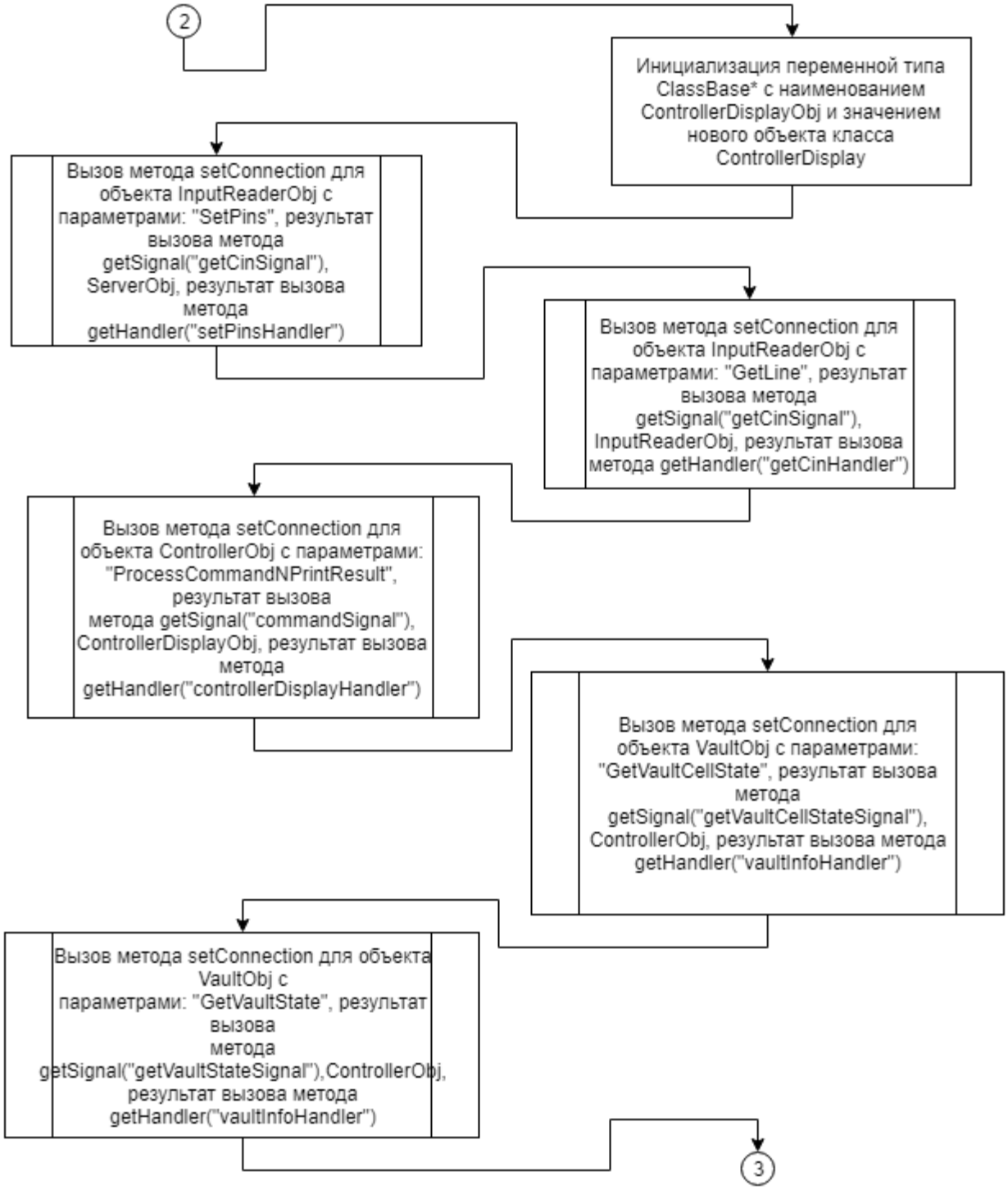

Рисунок 16 – Блок-схема алгоритма

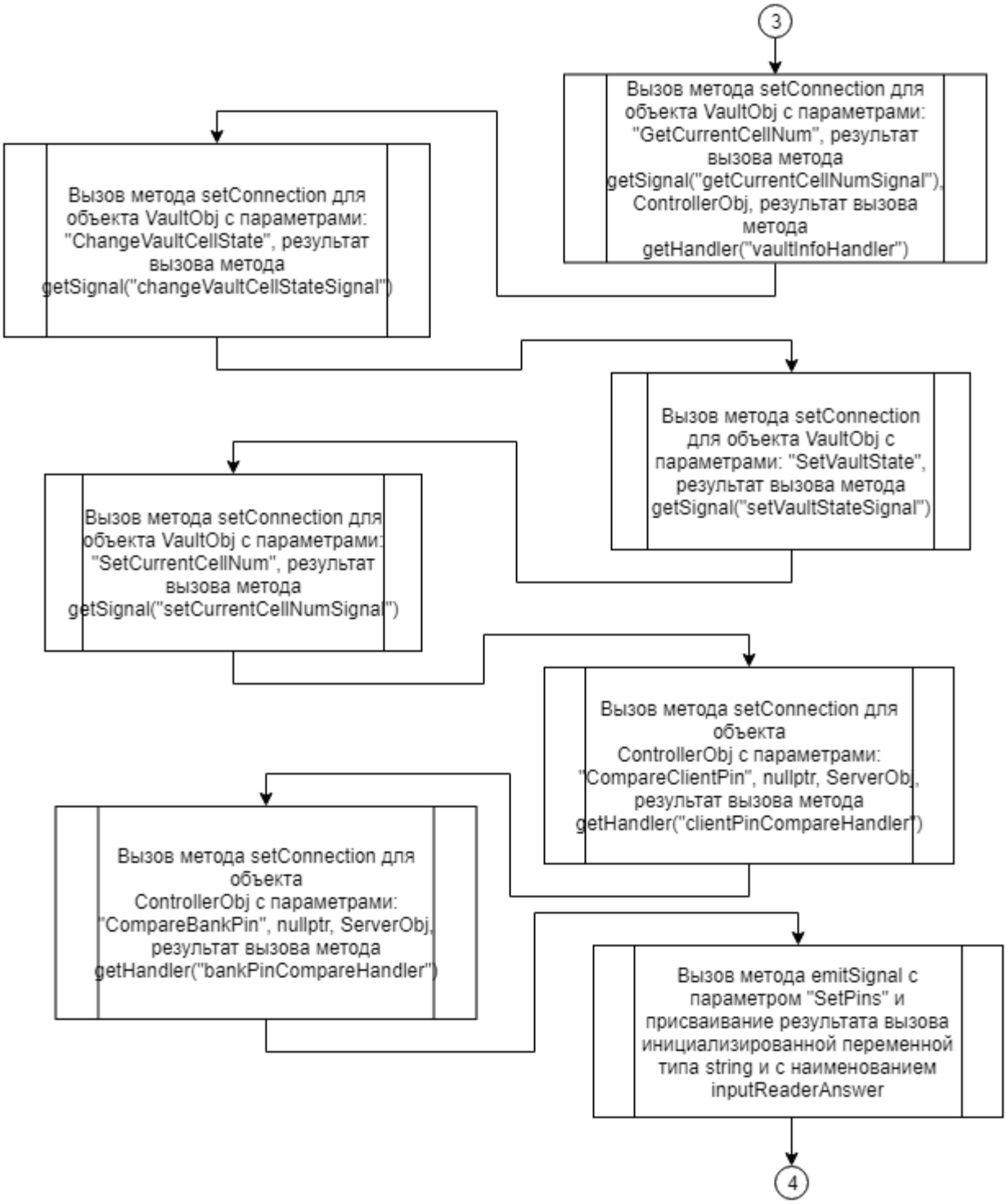

Рисунок 17 – Блок-схема алгоритма

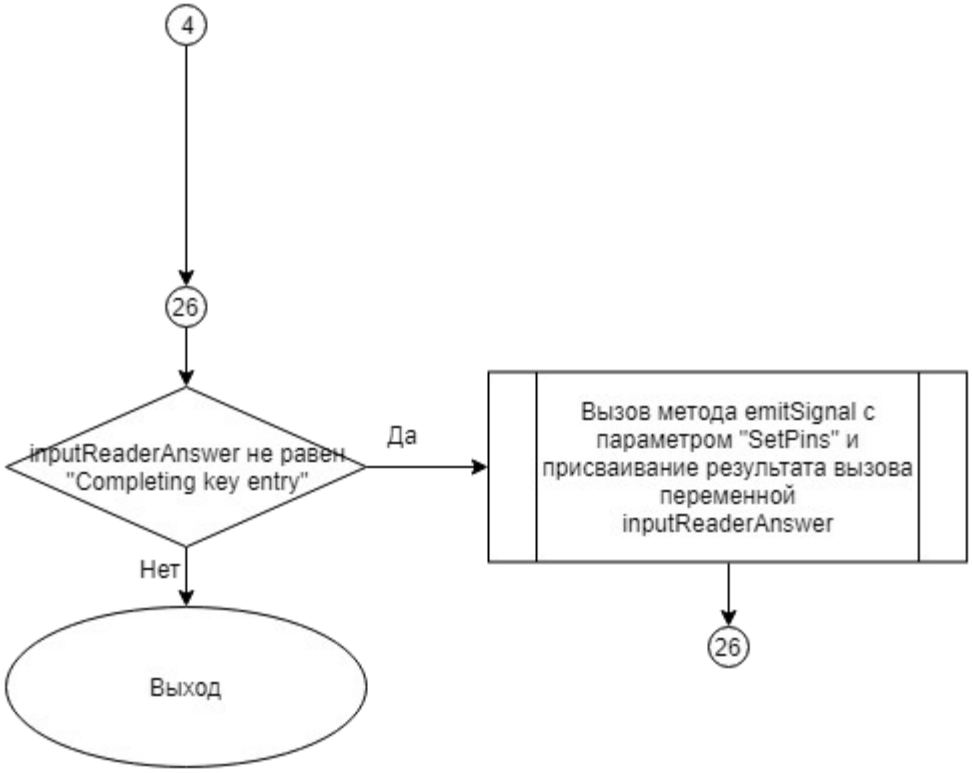

Рисунок 18 – Блок-схема алгоритма

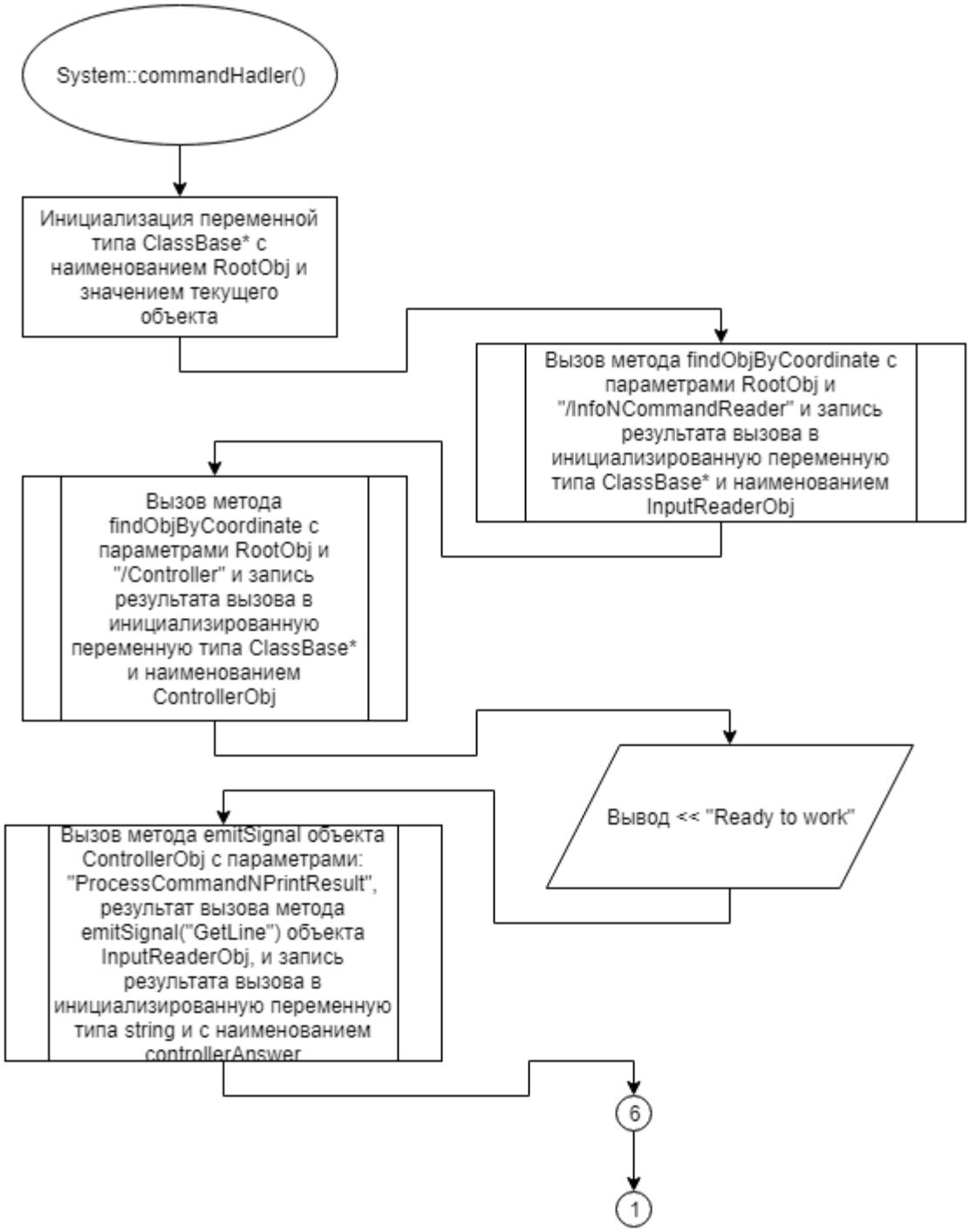

Рисунок 19 – Блок-схема алгоритма

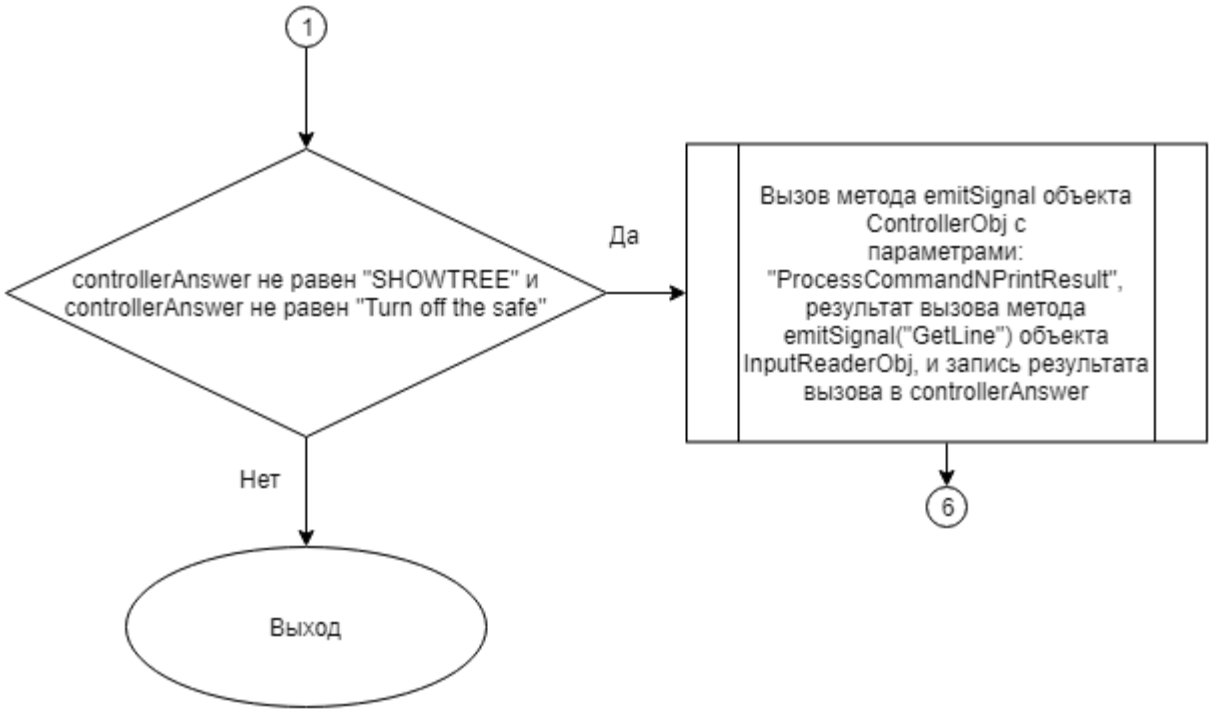

Рисунок 20 – Блок-схема алгоритма

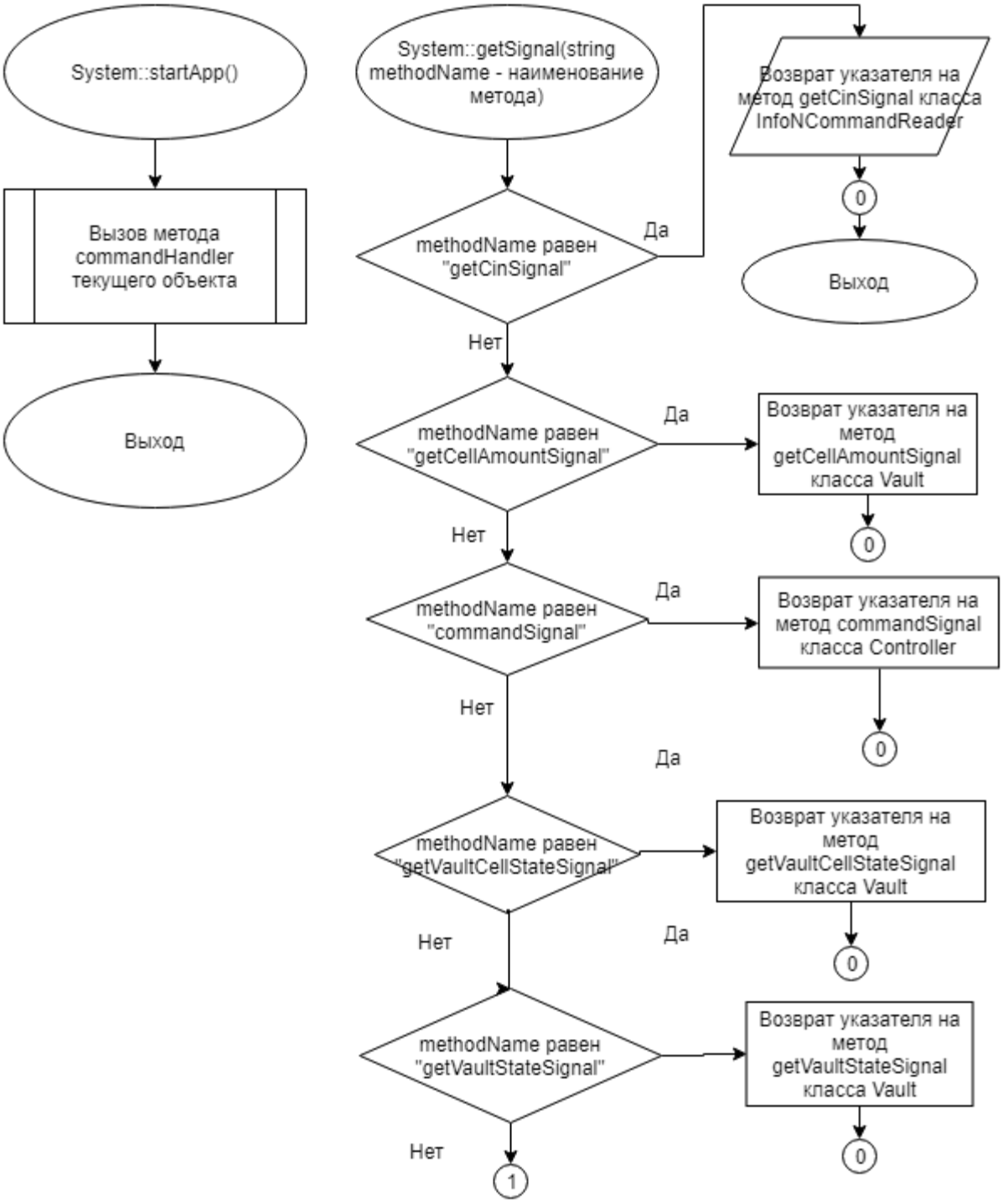

Рисунок 21 – Блок-схема алгоритма

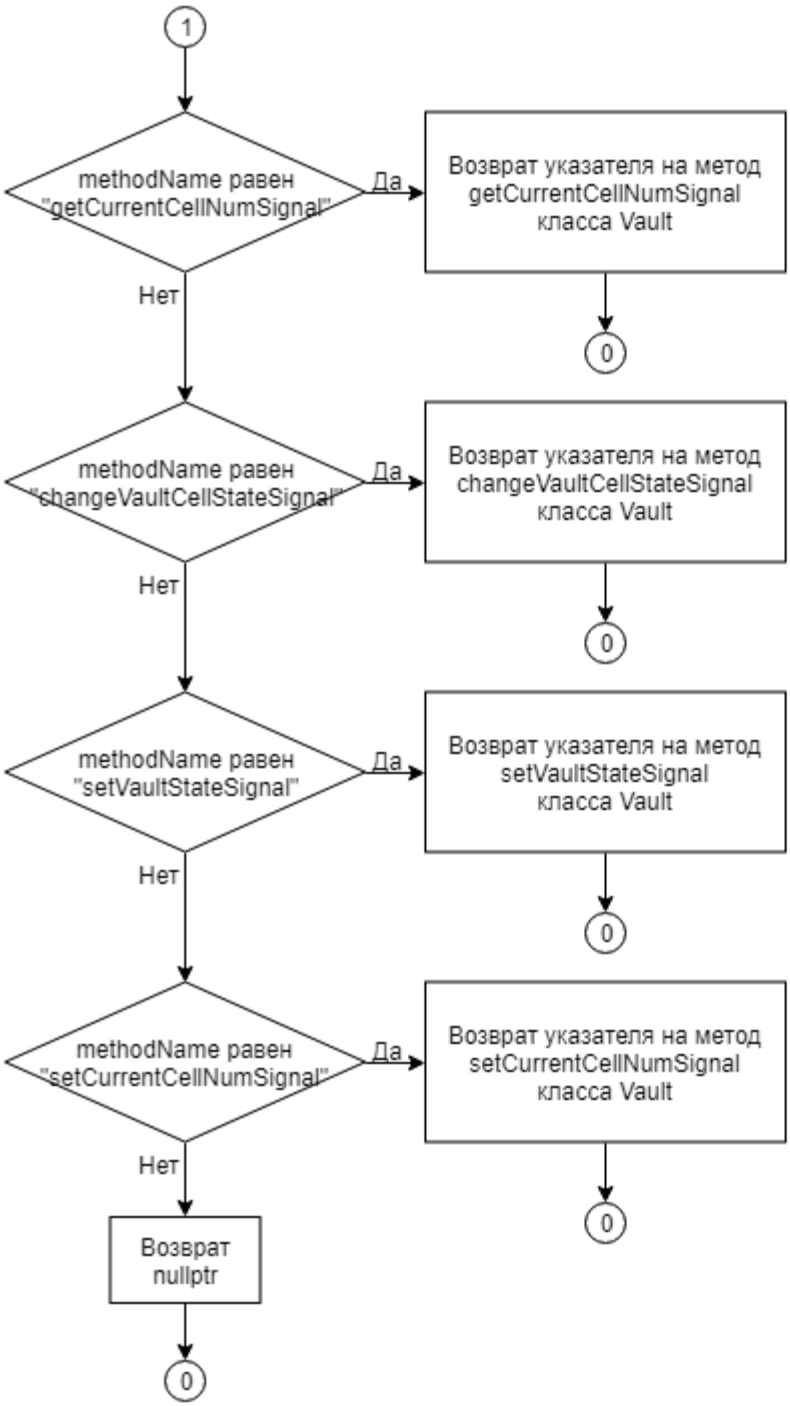

Рисунок 22 – Блок-схема алгоритма

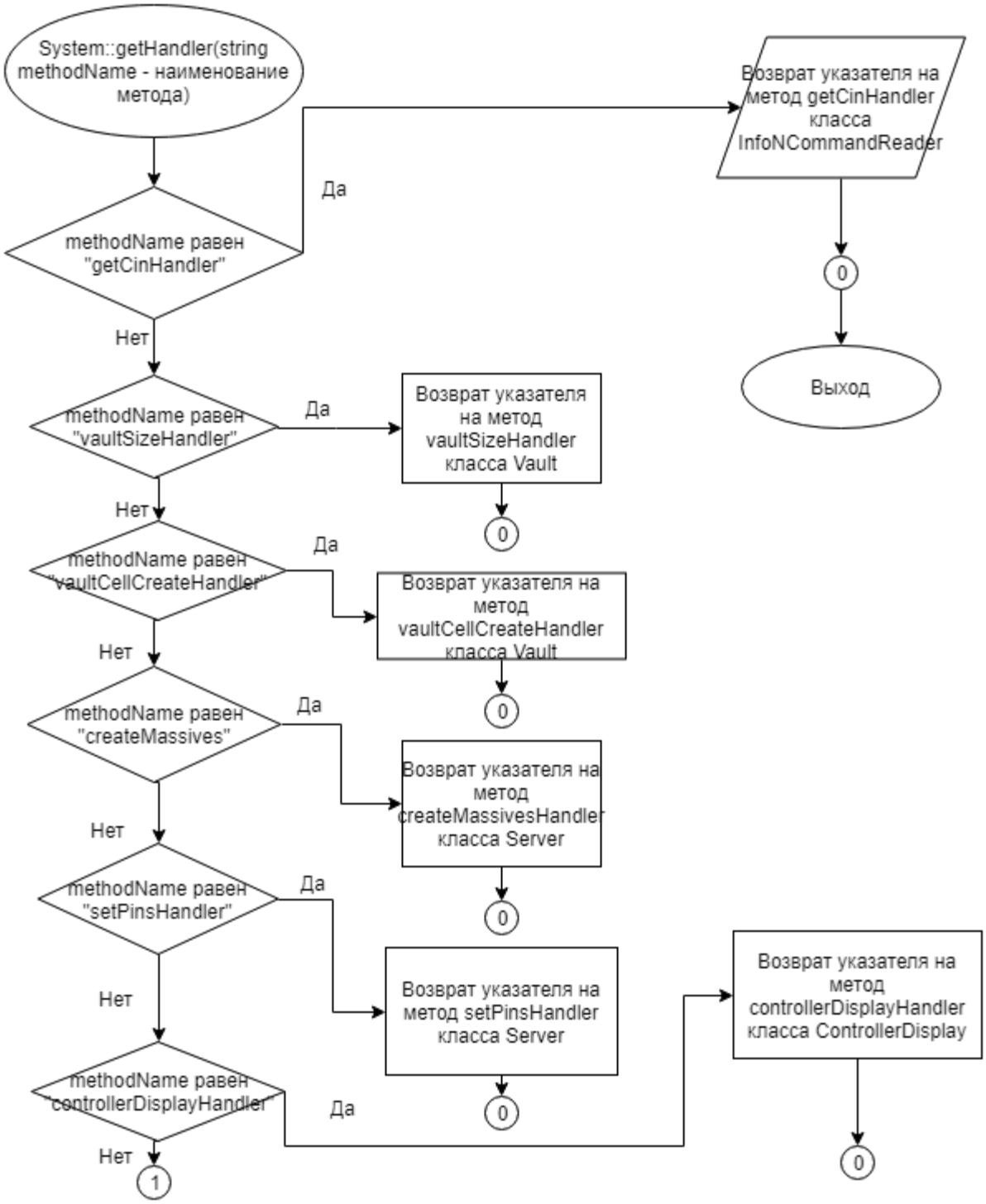

Рисунок 23 – Блок-схема алгоритма

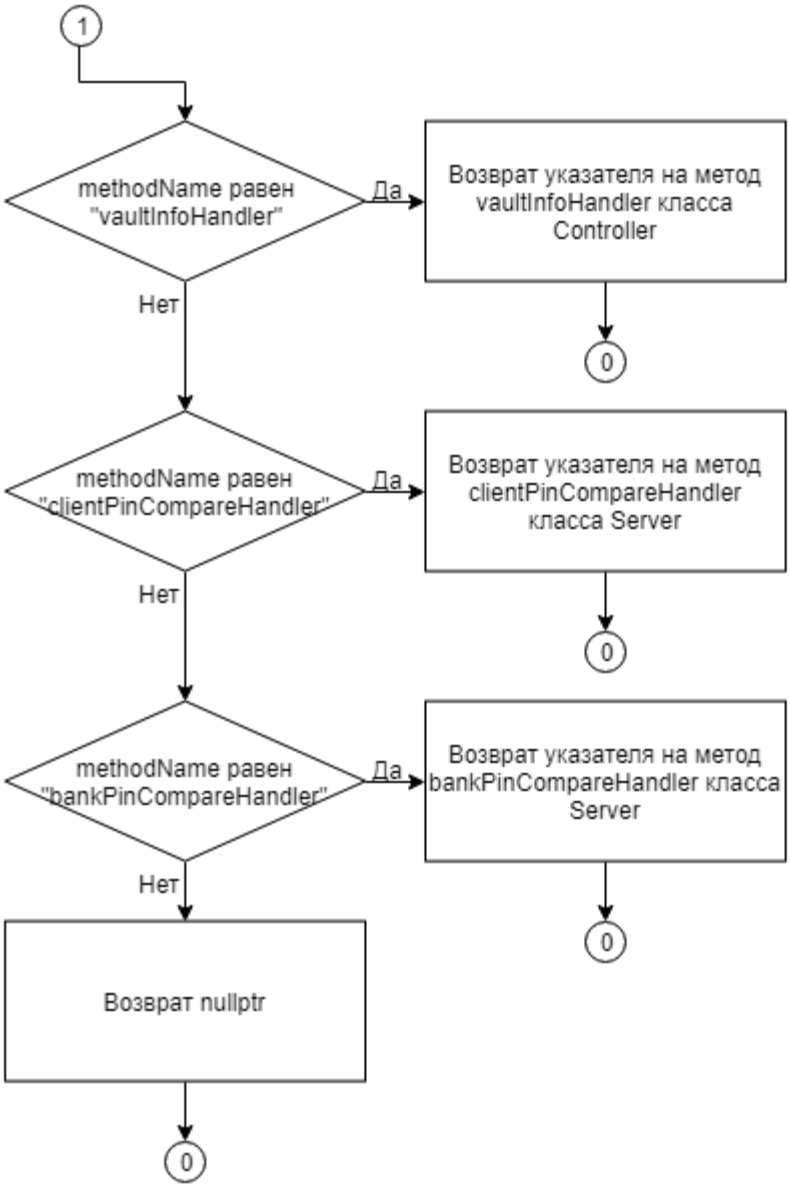

Рисунок 24 – Блок-схема алгоритма

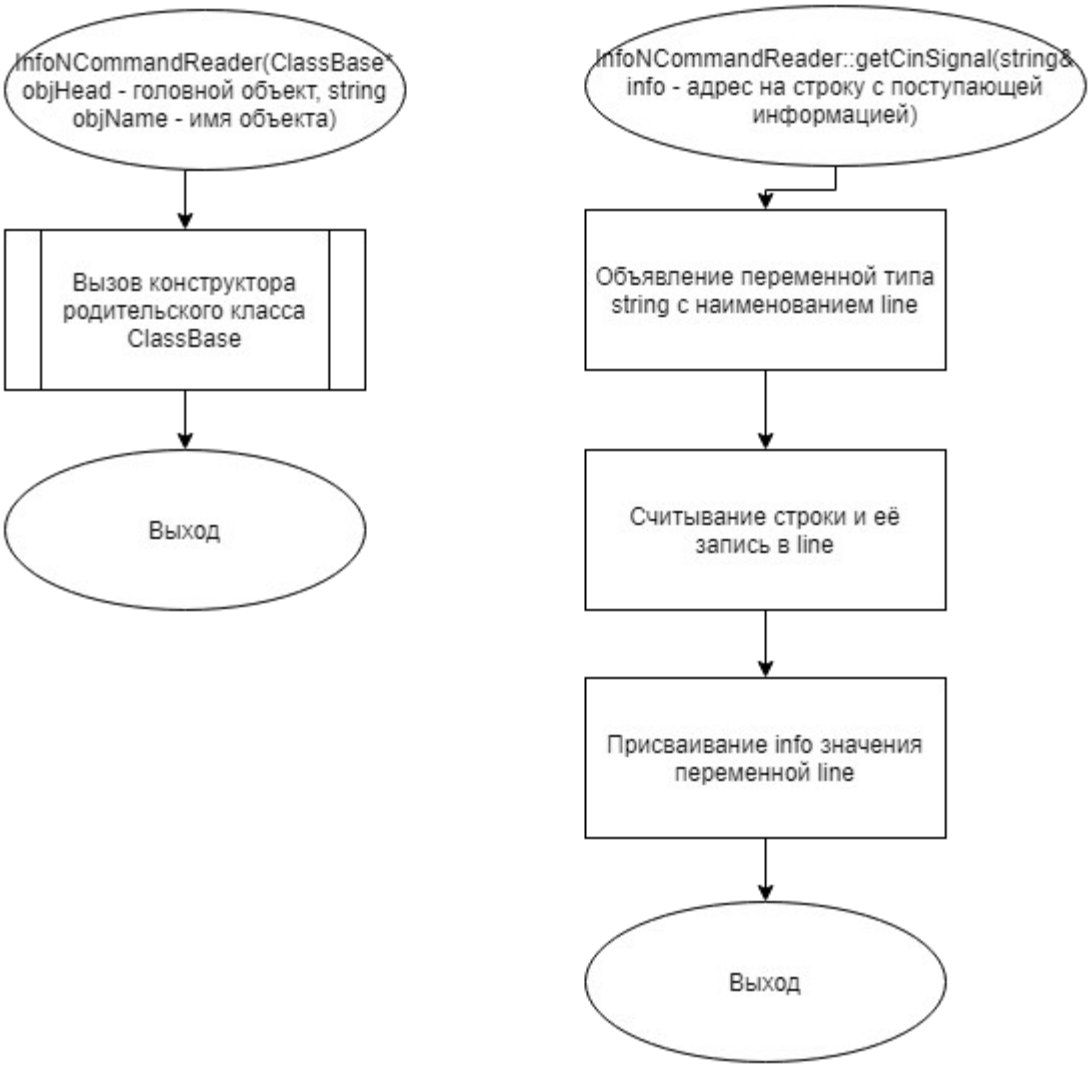

Рисунок 25 – Блок-схема алгоритма

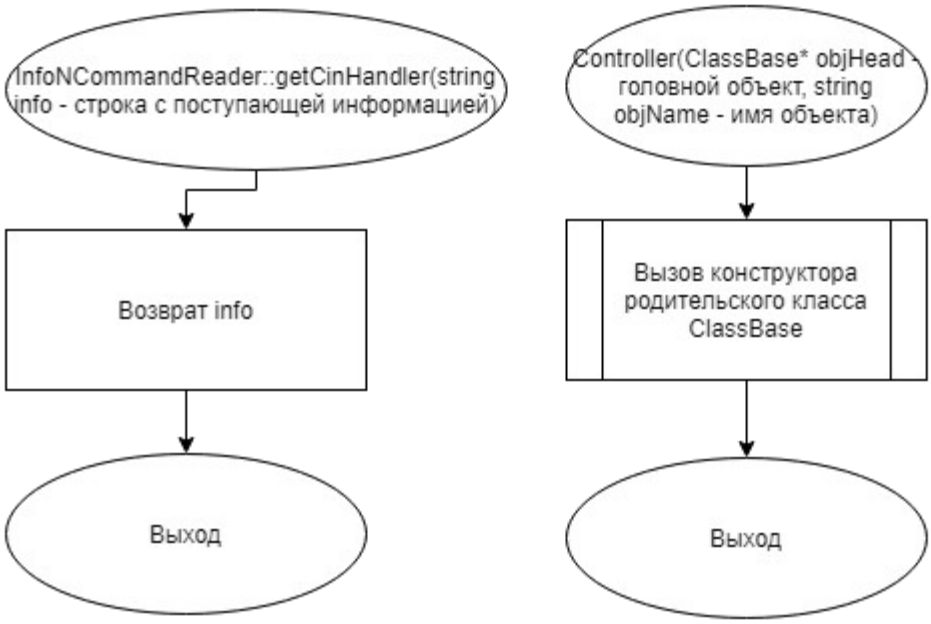

Рисунок 26 – Блок-схема алгоритма

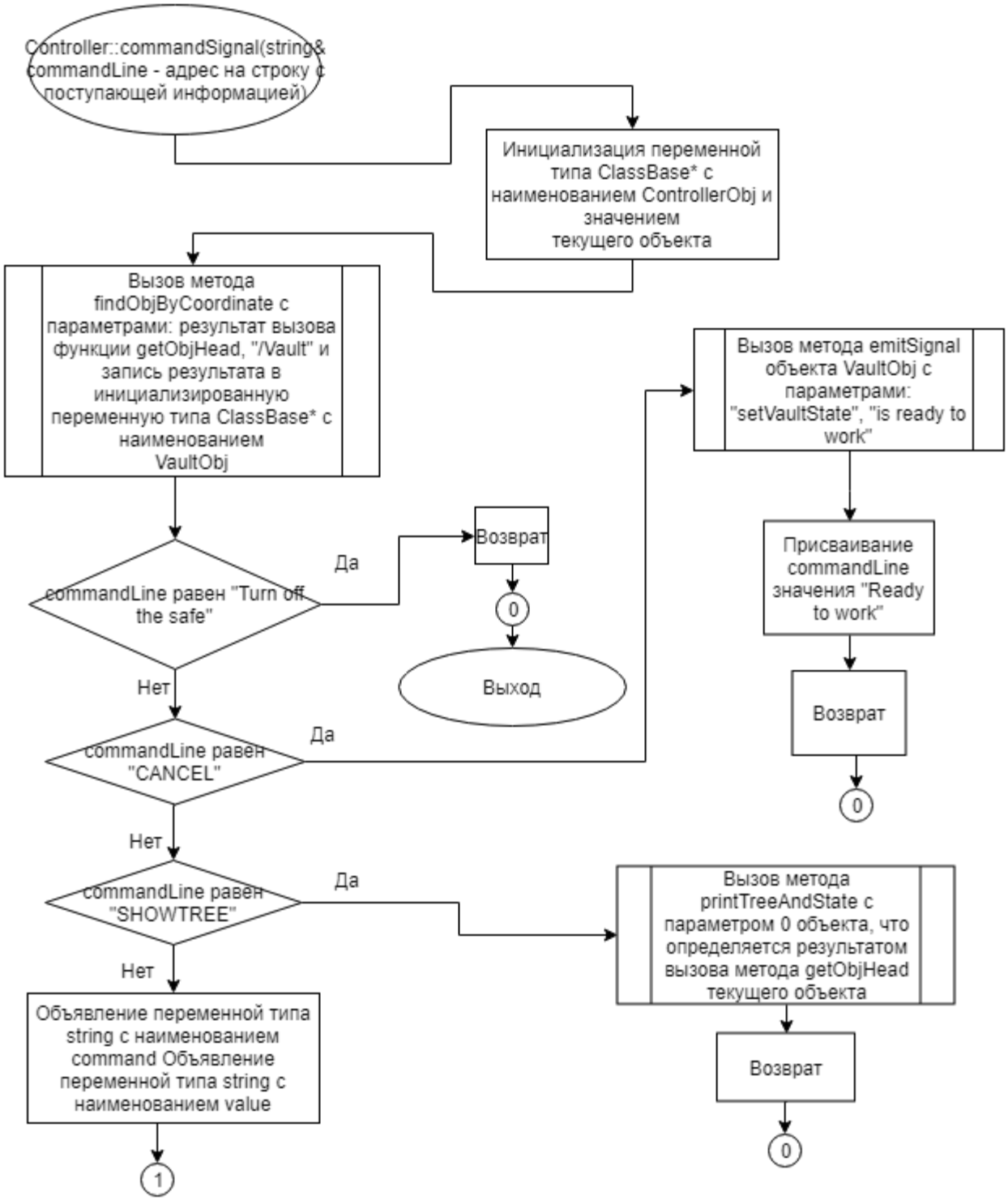

Рисунок 27 – Блок-схема алгоритма

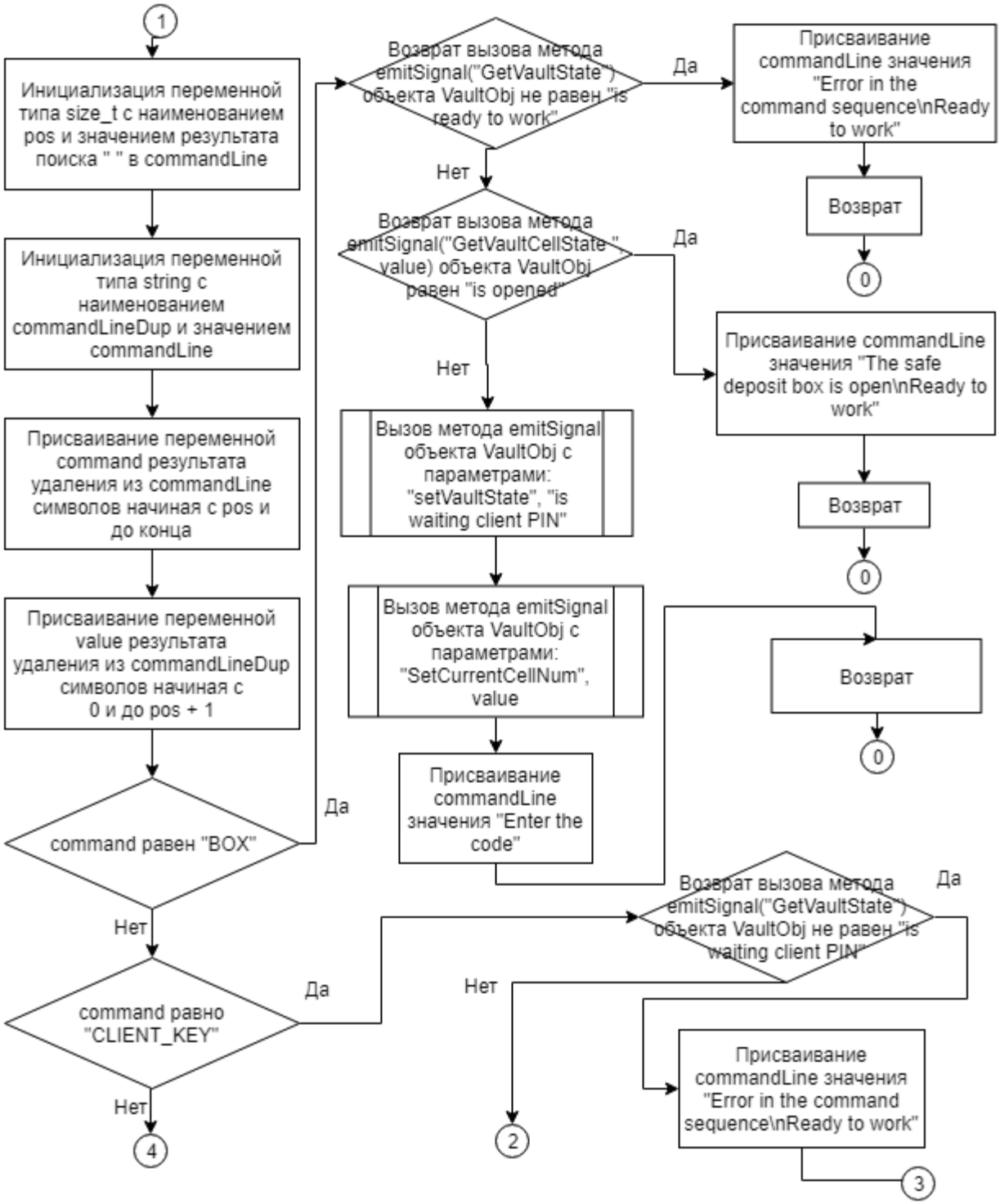

Рисунок 28 – Блок-схема алгоритма

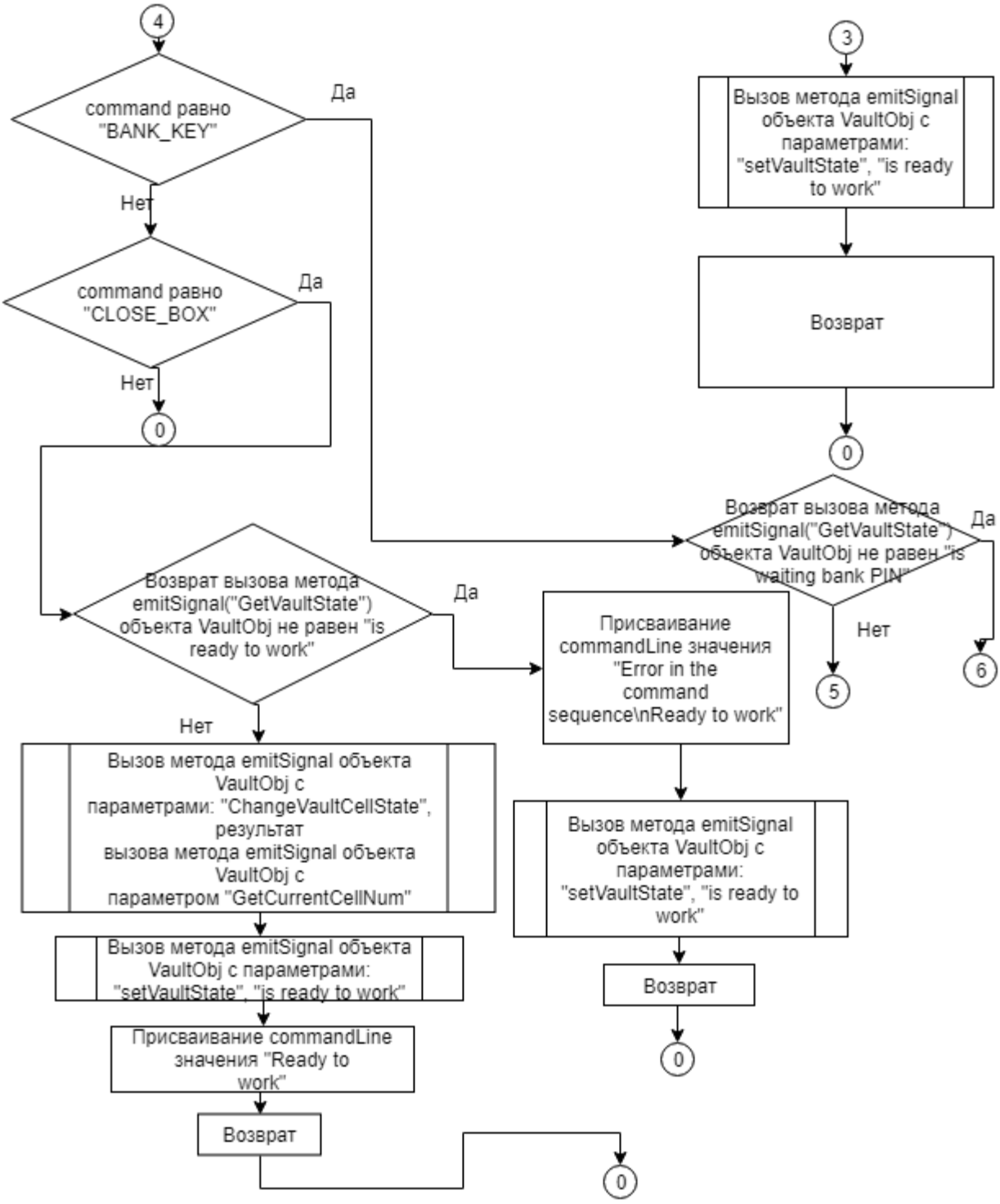

Рисунок 29 – Блок-схема алгоритма

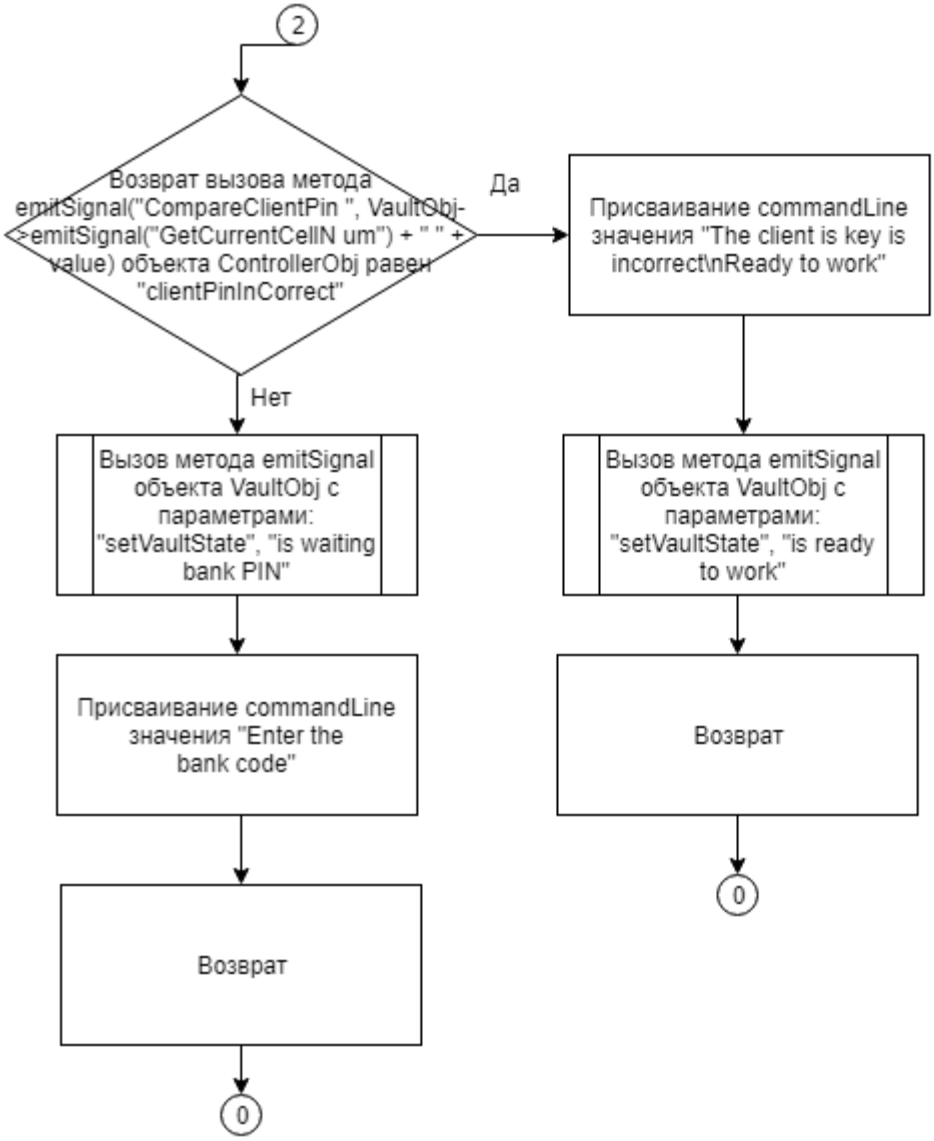

Рисунок 30 – Блок-схема алгоритма

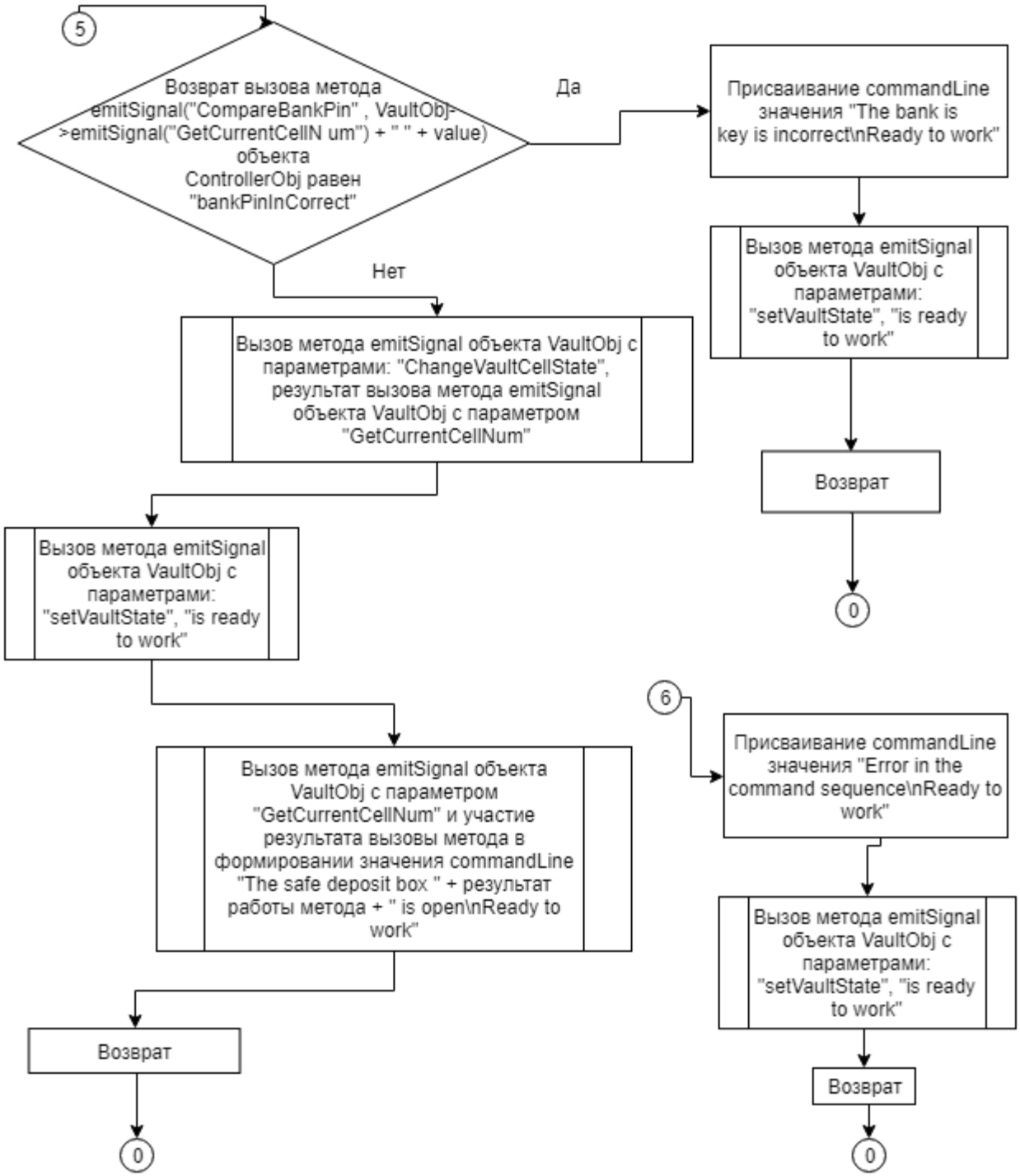

Рисунок 31 – Блок-схема алгоритма

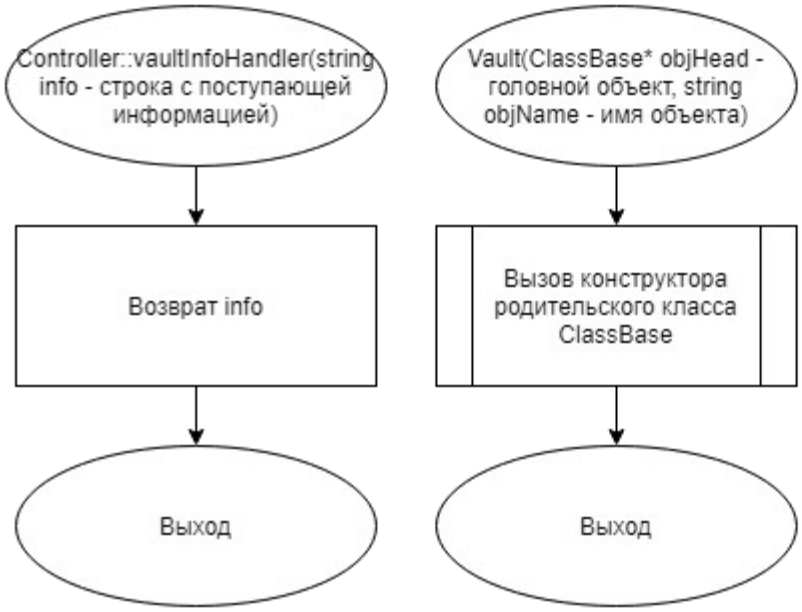

Рисунок 32 – Блок-схема алгоритма

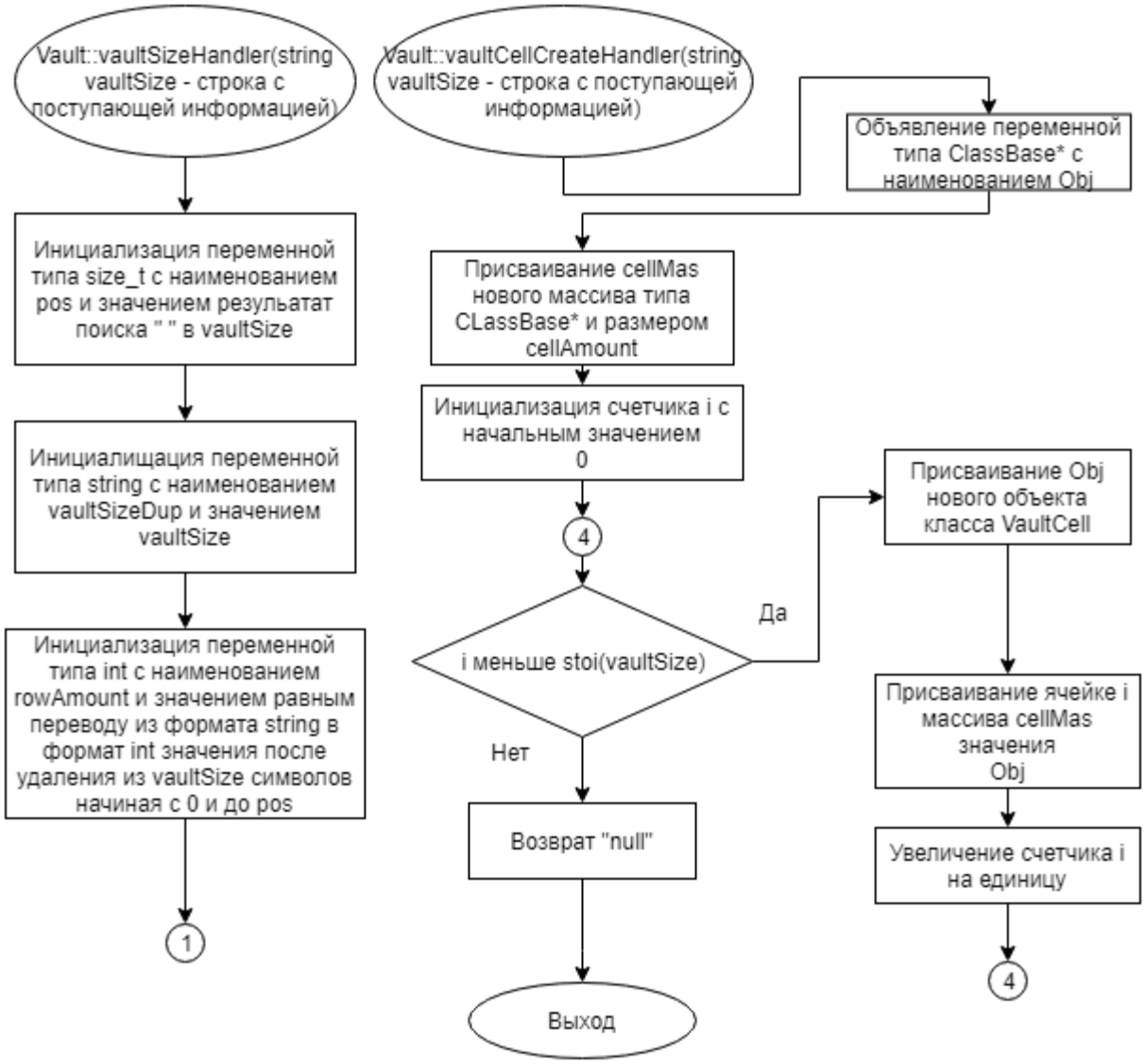

Рисунок 33 – Блок-схема алгоритма

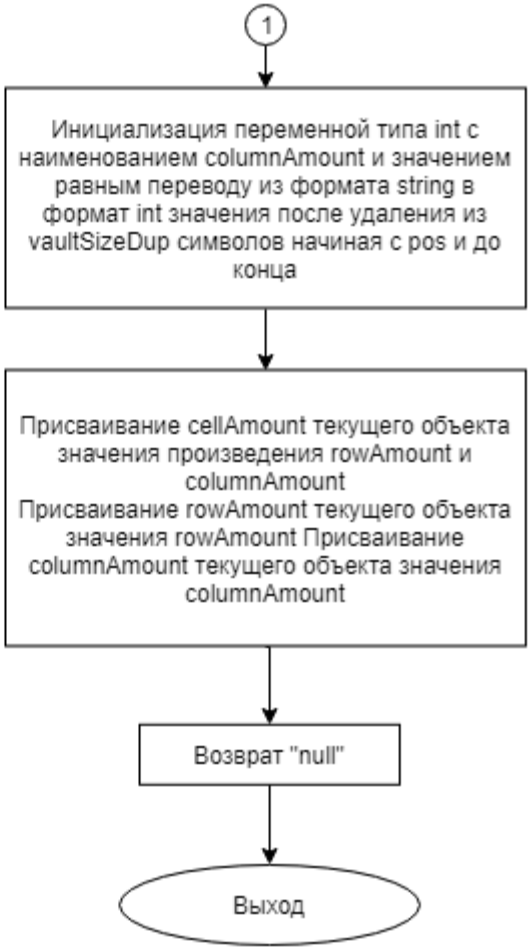

Рисунок 34 – Блок-схема алгоритма

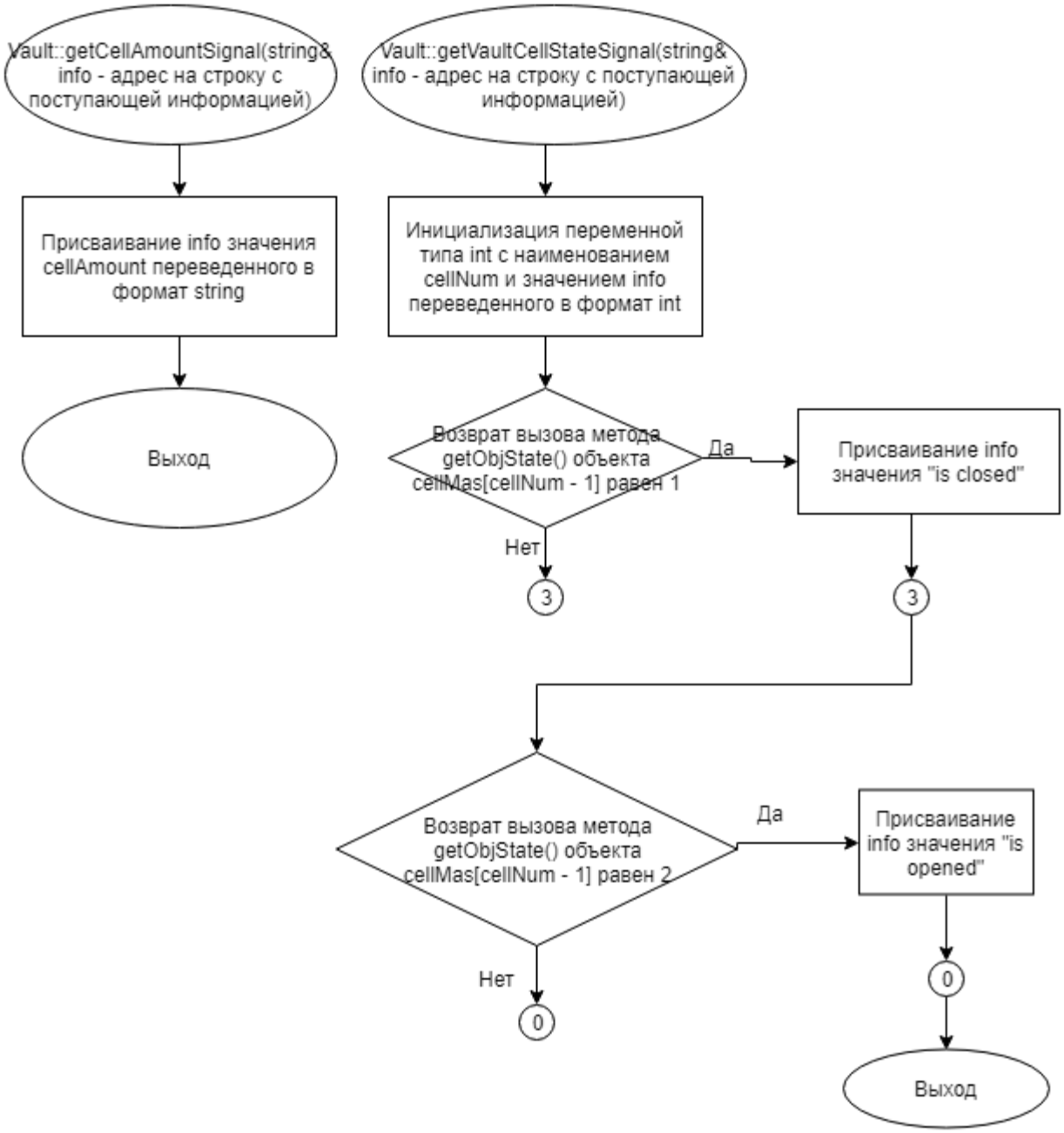

Рисунок 35 – Блок-схема алгоритма

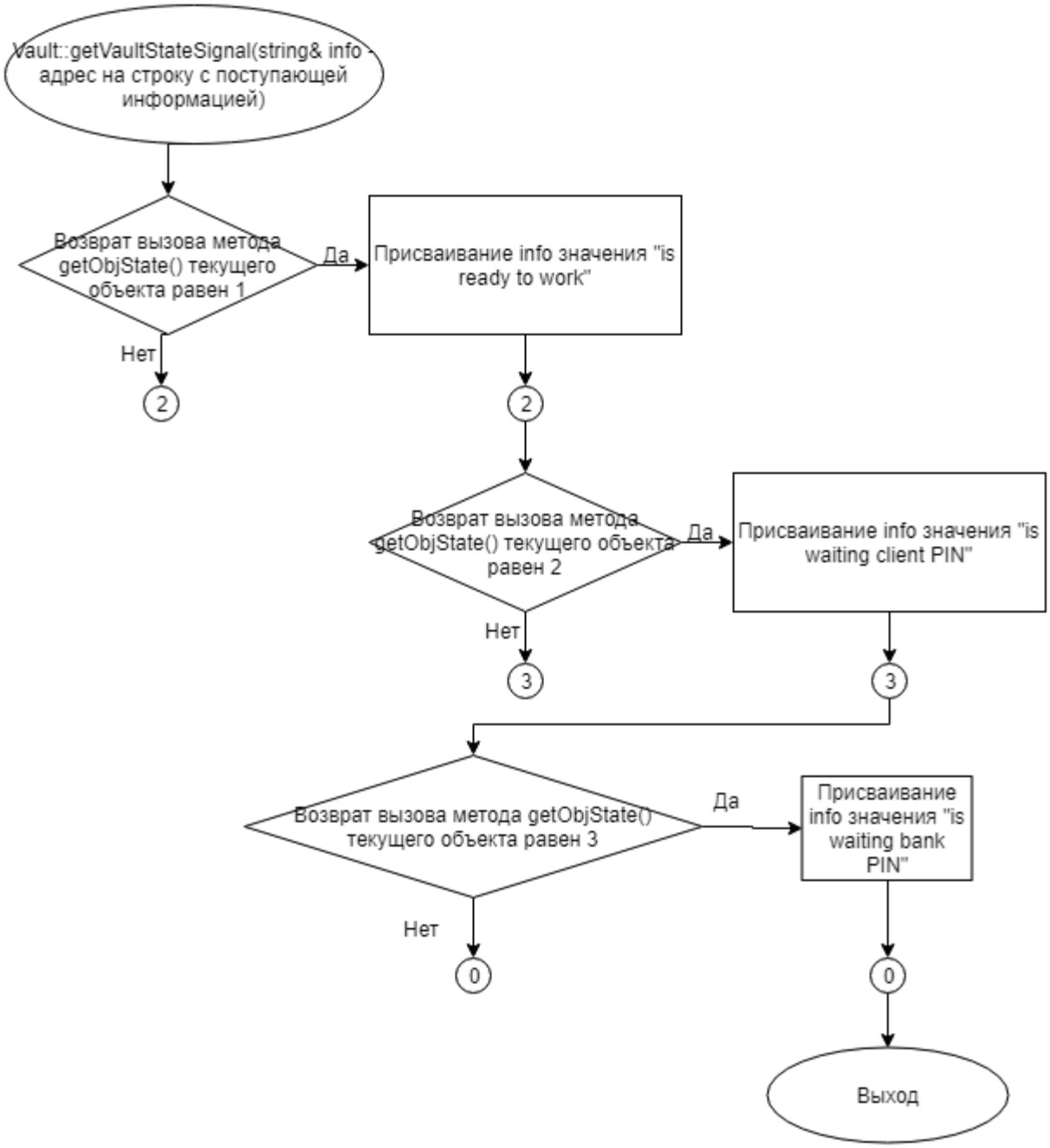

Рисунок 36 – Блок-схема алгоритма

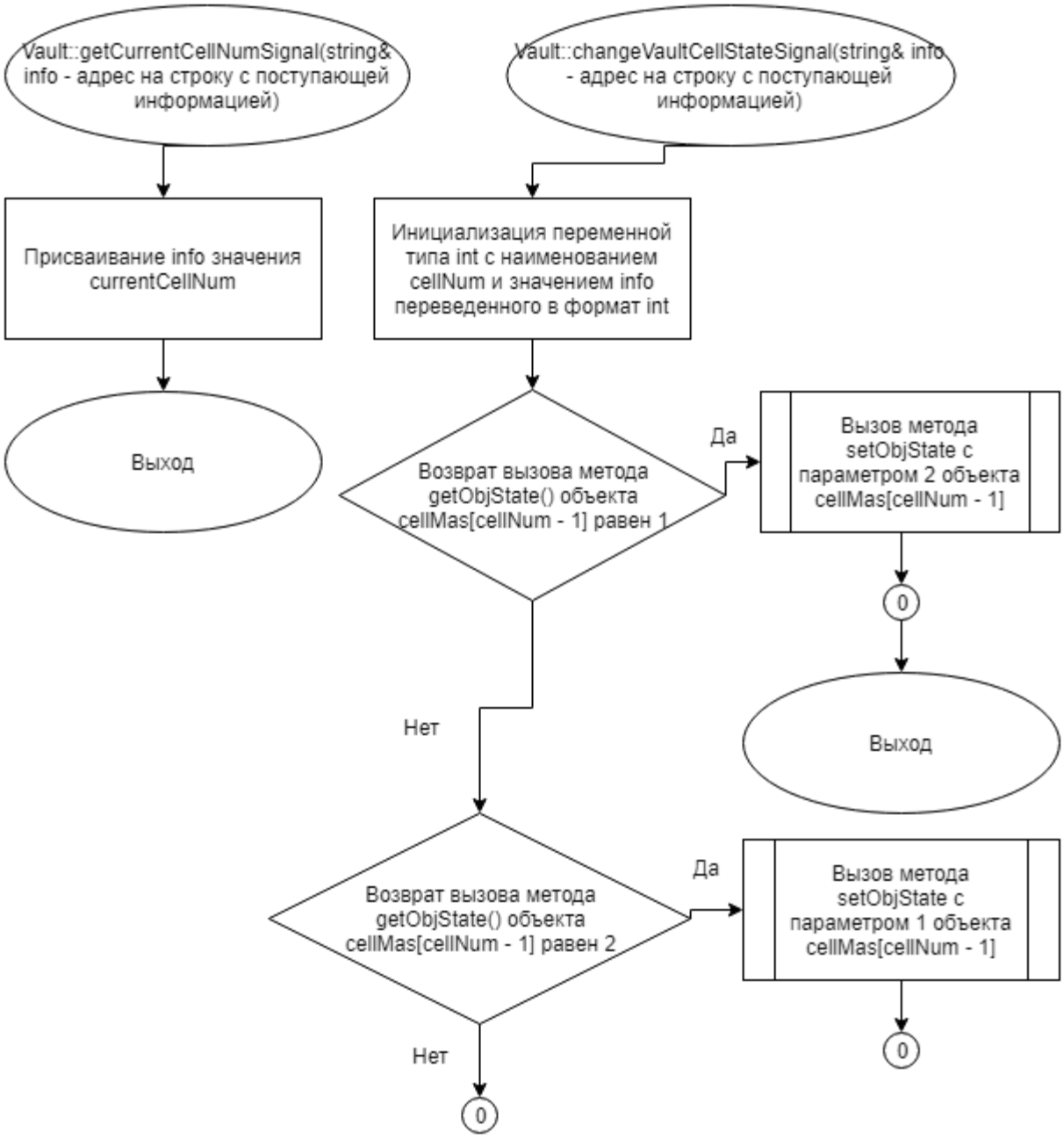

Рисунок 37 – Блок-схема алгоритма

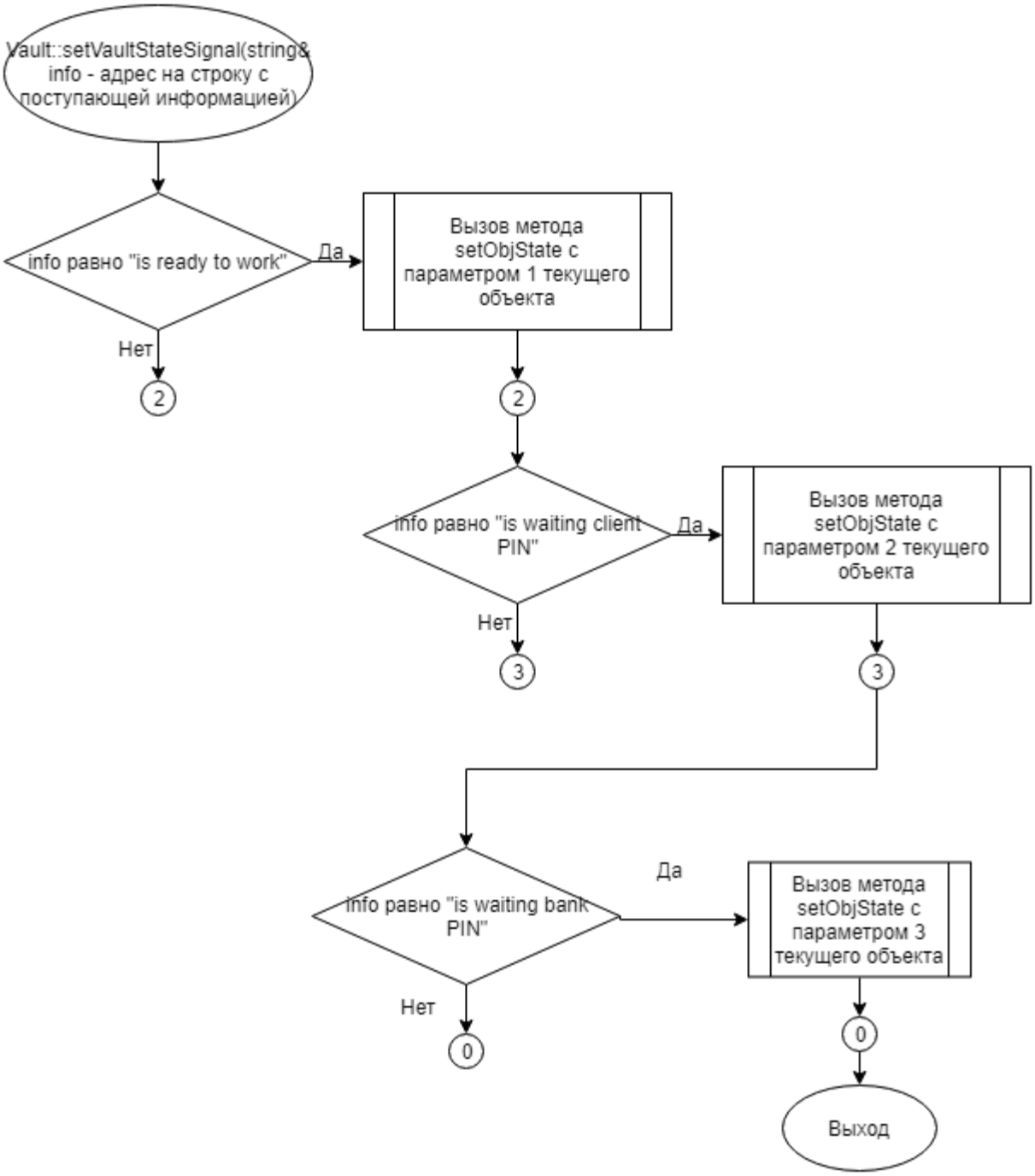

Рисунок 38 – Блок-схема алгоритма

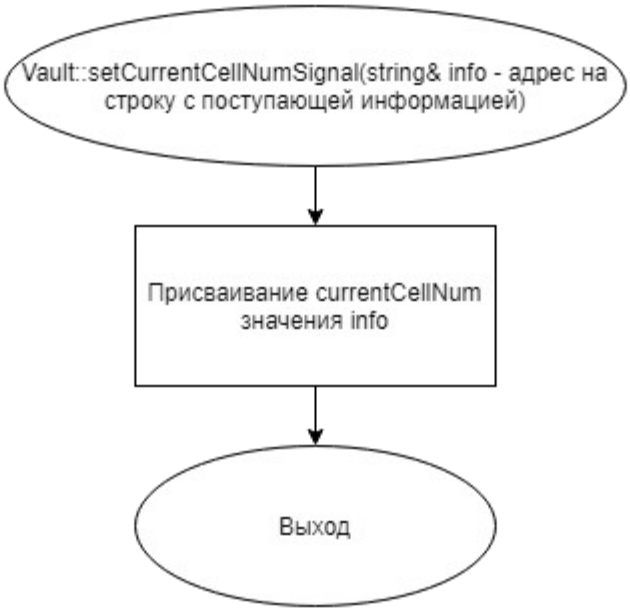

Рисунок 39 – Блок-схема алгоритма

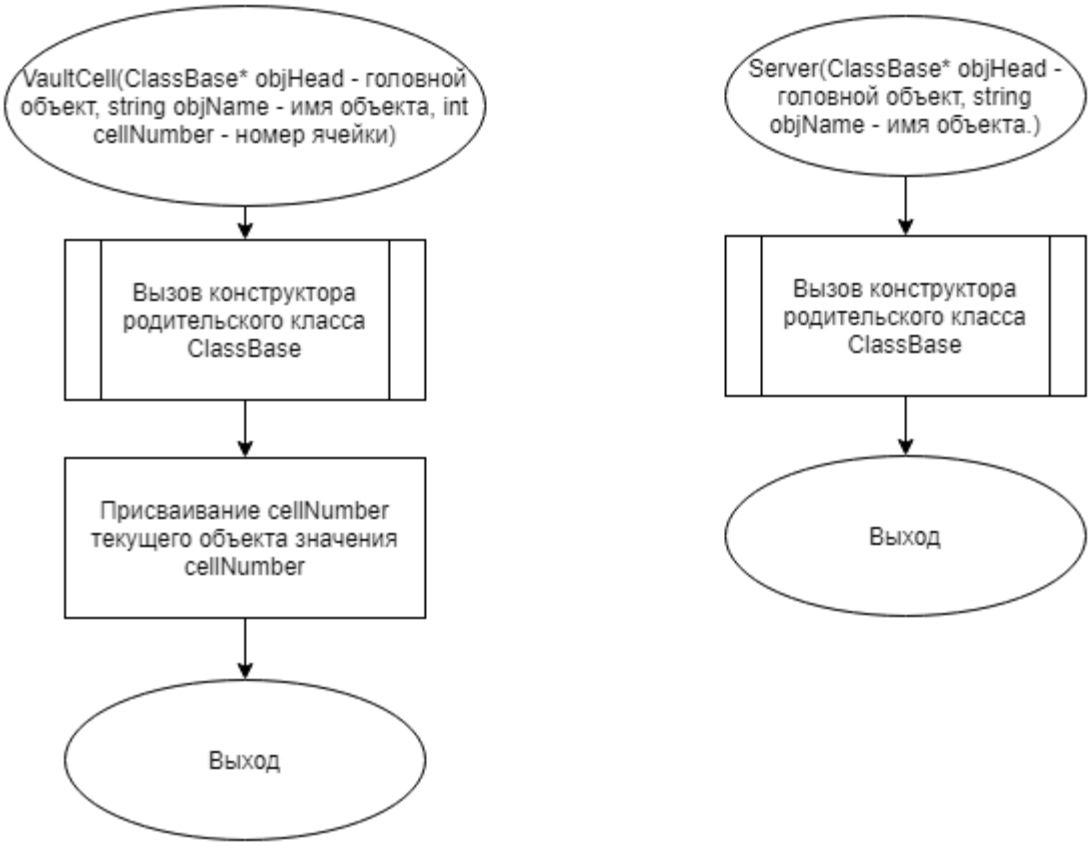

Рисунок 40 – Блок-схема алгоритма

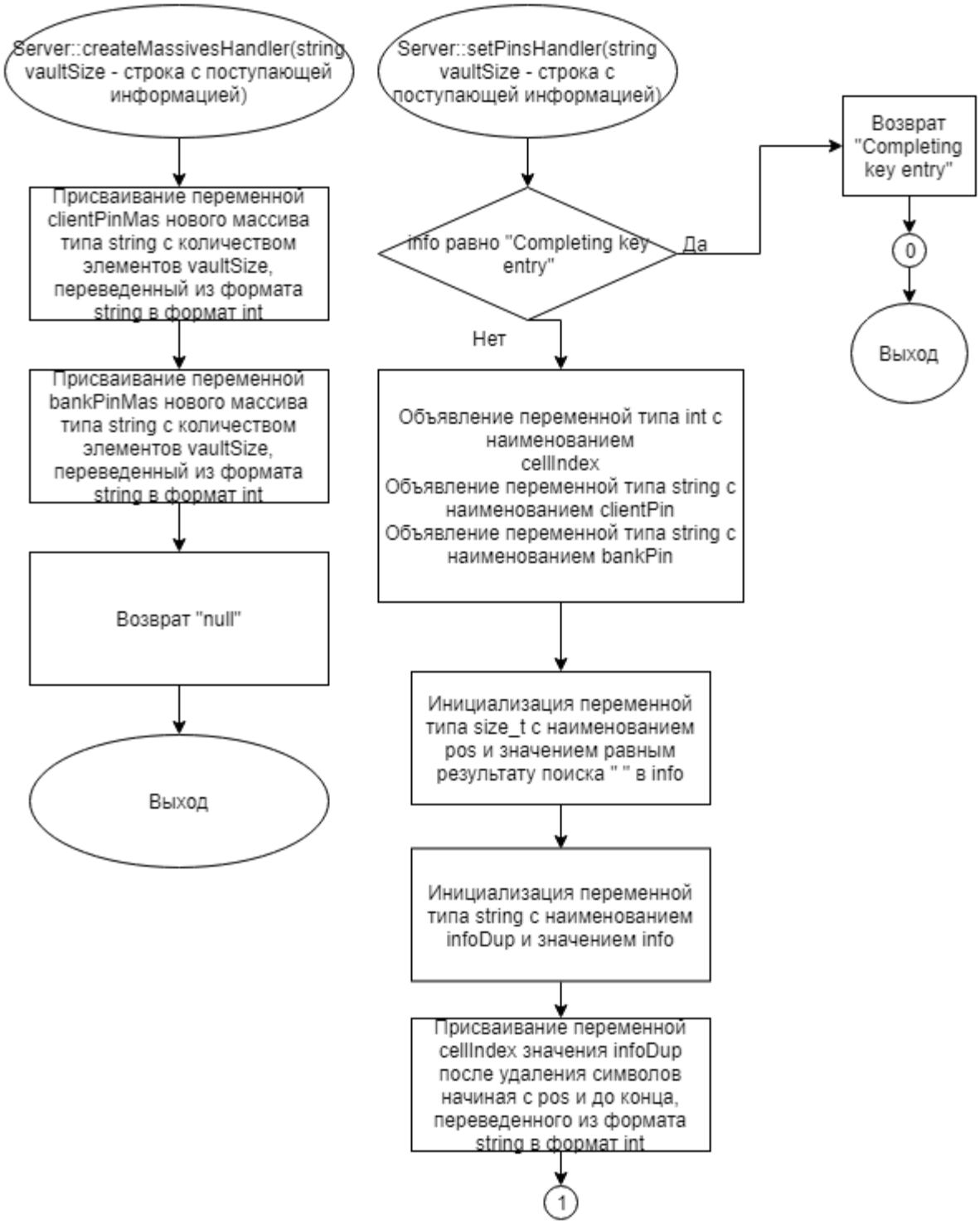

Рисунок 41 – Блок-схема алгоритма

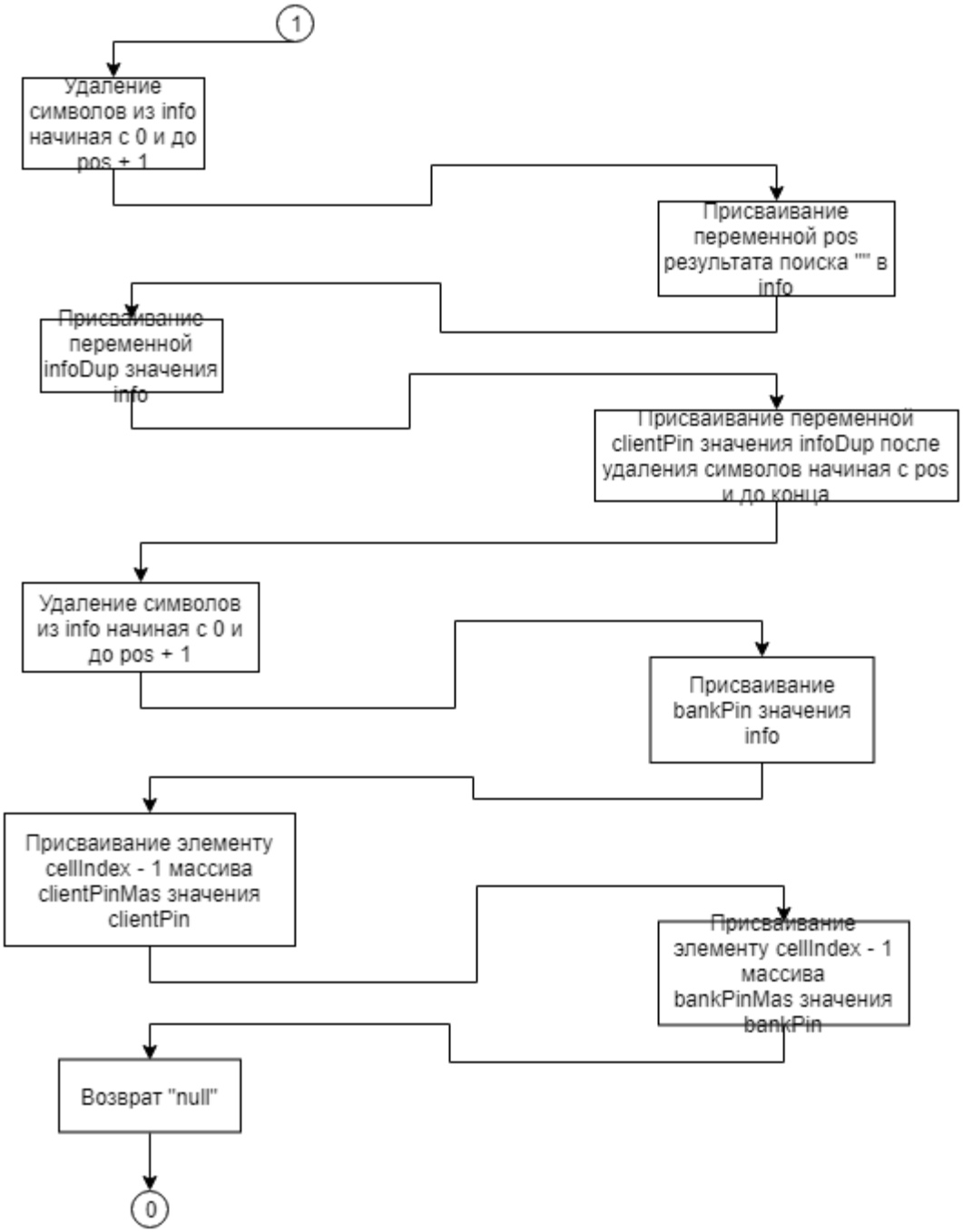

Рисунок 42 – Блок-схема алгоритма

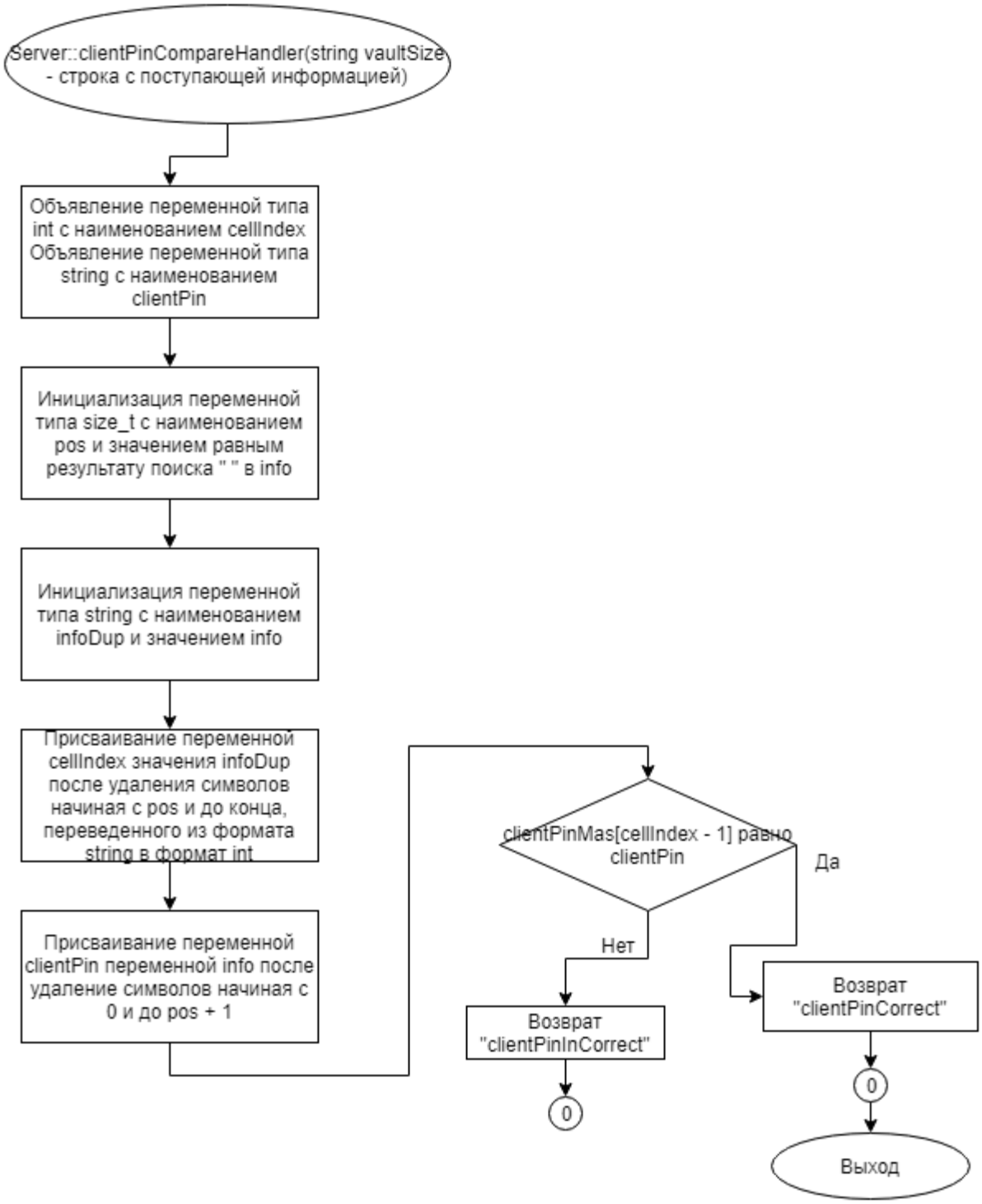

Рисунок 43 – Блок-схема алгоритма

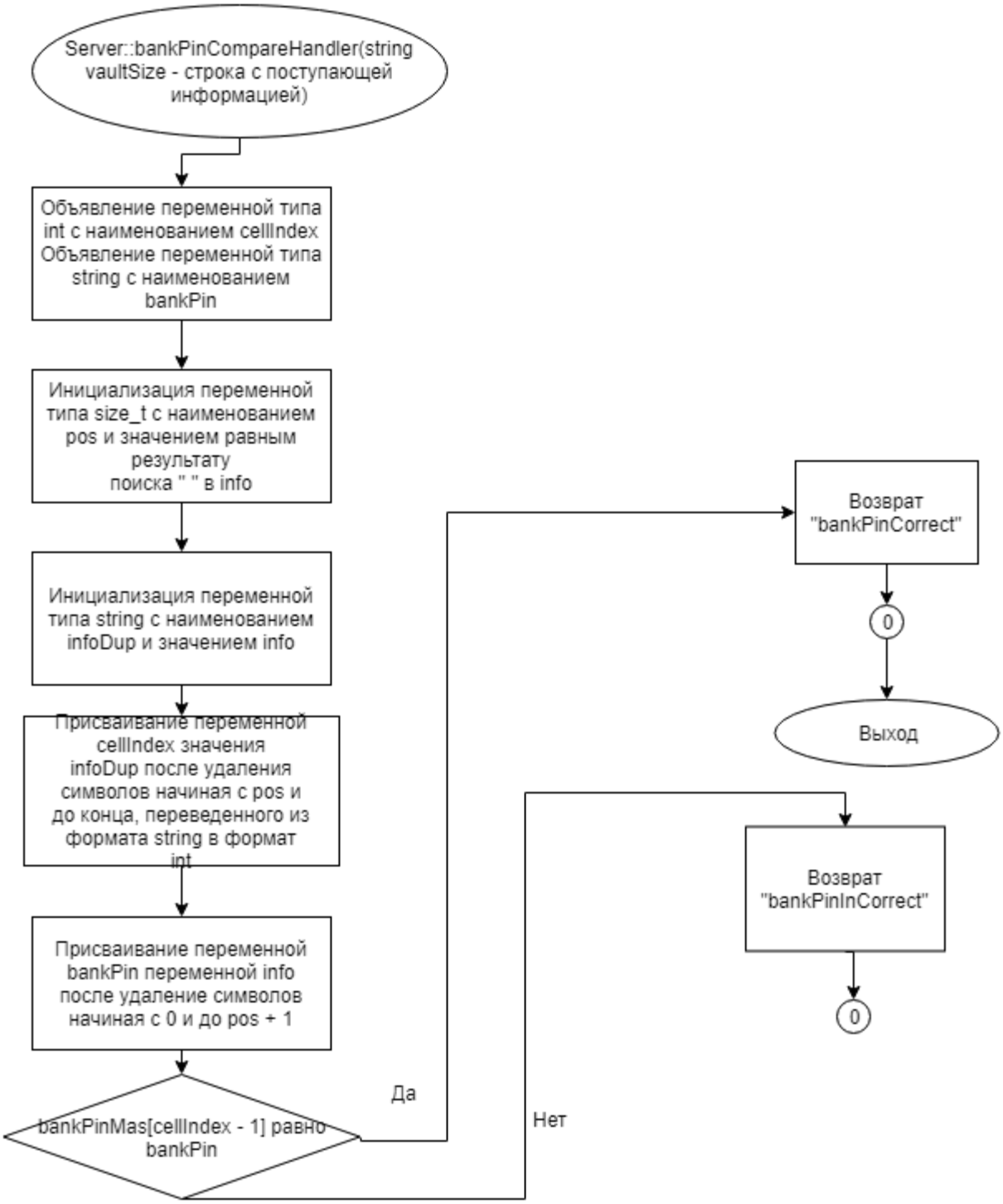

Рисунок 44 – Блок-схема алгоритма

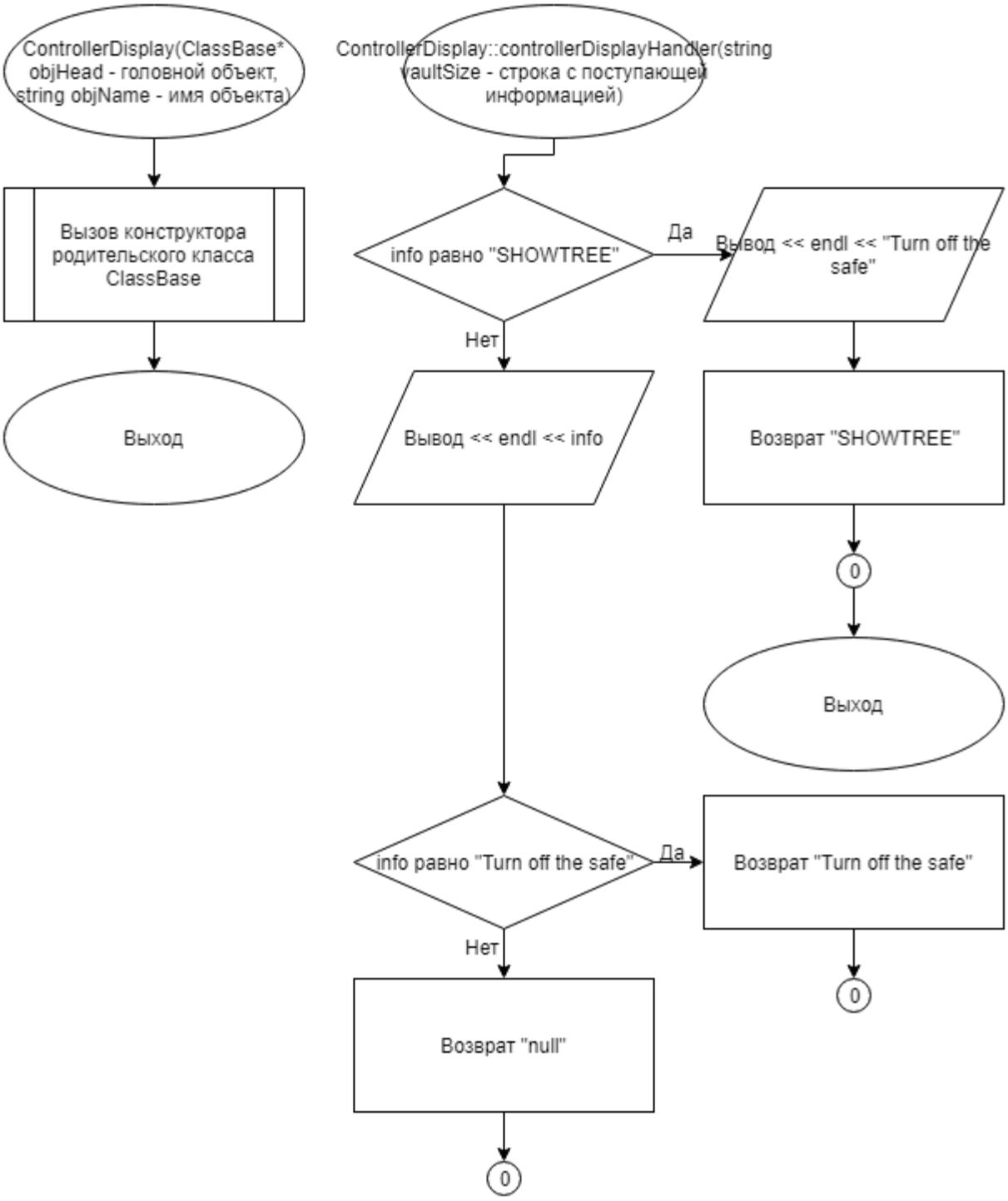

Рисунок 45 – Блок-схема алгоритма

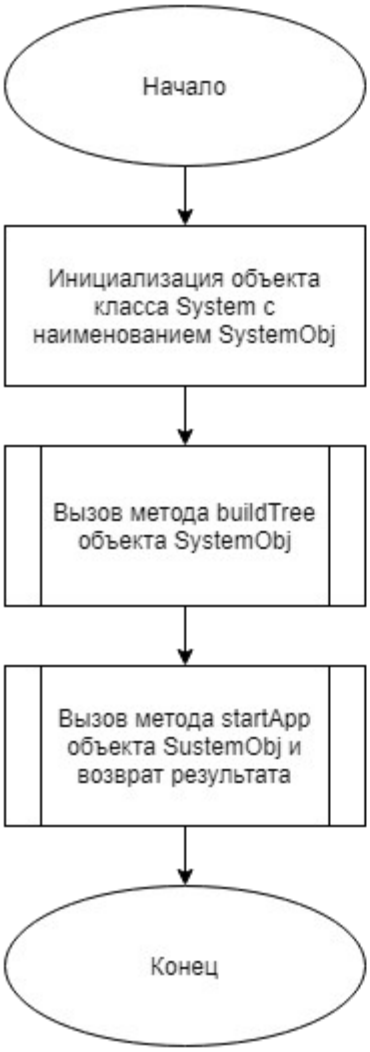

Рисунок 46 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл ClassBase.cpp

Листинг 1 – ClassBase.cpp

```
#include "ClassBase.h"
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

// Определение конструктора класса ClassBase, который принимает указатель на родительский объект и имя объекта
ClassBase :: ClassBase(ClassBase* objHead, string objName) {
   // Установить имя объекта
   setObjName(objName);
   // Если родительский объект существует
   if (objHead != nullptr) {
      // Установить родительский объект
      setObjHead(objHead);
      //  Добавить текущий объект в список подчиненных объектов у родительского объекта
      objHead -> objSub.push_back(this);
   }
}

// Метод возврата имени объекта
string ClassBase :: getObjName() {
   return objName;
}

// Метод возврата родительского объекта
ClassBase* ClassBase :: getObjHead() {
   return objHead;
}

// Метод возврата состояния объекта
int ClassBase :: getObjState() {
   return objState;
}

// Метод установки имени объекта
void ClassBase :: setObjName(string objName) {
```

```
   this -> objName = objName;
}

// Метод установки родительского объекта
void ClassBase :: setObjHead(ClassBase* objHead) {
   this -> objHead = objHead;
}

// Метод установки состояния объекта
void ClassBase :: setObjState(int objState) {
   // Если состояние объекта равно 0
   if (objState == 0) {
      // Установить состояние объекта
      this -> objState = objState;
      // Рекурсивно установить состояние для всех подчиненных объектов
      for (int i = 0; i < objSub.size(); i++) {
         objSub[i] -> setObjState(objState);
      }
   }
   // Если у родительского объекта установлено состояние отличное от 0
   if (objHead != nullptr && objHead -> objState != 0) {
      // Установить состояние объекта
      this -> objState = objState;
   }
}

// Метод вывода дерева объектов и их состояния на экран
void ClassBase :: printTreeAndState(int step) {
   cout << endl; // Выводим символ переноса строки для отделения строк вывода
   for (int i = 0; i < step; i++) { // Цикл по количеству пробелов, равному
уровню вложенности
      cout << " ";
   }
   cout << this -> getObjName(); // Выводим название объекта
   if (this -> getObjName() == "Vault") {
      if (this -> objState == 1) {
         cout << " is ready to work"; // Выводим состояние объекта
      }
      if (this -> objState == 2) {
         cout << " is waiting client PIN"; // Выводим состояние объекта
      }
      if (this -> objState == 3) {
         cout << " is waiting bank PIN"; // Выводим состояние объекта
      }
   }
   else if (this -> getObjName().find("VaultCell") != string::npos) {
      if (this -> objState == 1) {
         cout << " is closed"; // Выводим состояние объекта
      }
      if (this -> objState == 2) {
         cout << " is opened"; // Выводим состояние объекта
      }
   }
   else {
      if (this -> objState != 0) {
```

```
            cout << " is ready"; // Выводим состояние объекта
         }
         else {
            cout << " is not ready"; // Выводим состояние объекта
         }
      }

      if (this -> objSub.size() > 0) {
         for (int i = 0; i < this -> objSub.size(); i++) {
            objSub[i] -> printTreeAndState(step + 1); // Рекурсивно вызываем
метод для подобъектов
         }
      }
   }

   // Метод поиска объекта по координатам
   ClassBase* ClassBase :: findObjByCoordinate(ClassBase* CurrentObj, string
   Coordinate) {
      // Создание указателя на объект и инициализация значением nullptr
      ClassBase* object = nullptr;
      // Проверка длины строки Coordinate и того, что первый символ равен '/'
      if (Coordinate.length() > 0 && Coordinate[0] == '/') {
         // Удаление первого символа из строки Coordinate
         Coordinate.erase(0,1);
         // Если Coordinate стала пустой, вернуть указатель на текущий объект
         if (Coordinate == "") {
            return this;
         }
         // Иначе вызвать рекурсивно метод findObjByCoordinate для объекта
   CurrentObj и обновленной строки Coordinate
         else {
            return this -> findObjByCoordinate(CurrentObj, Coordinate);
         }
      }
      // Если длина строки Coordinate больше 0
      if (Coordinate.length() > 0) {
         // Находим позицию символа '/' в строке Coordinate
         size_t position = Coordinate.find('/');
         // Создаем строку ObjName и присваиваем ей значение Coordinate
         string ObjName = Coordinate;
         // Если нашли символ '/', то удаляем его и все символы после него из
   строки ObjName
         if (position != string::npos) {
            ObjName.erase(position, Coordinate.size());
            Coordinate.erase(0, position);
         }
         // Иначе устанавливаем Coordinate в пустую строку
         else {
            Coordinate = "";
         }
         // Проходим по всем подчиненным объектам объекта CurrentObj
         for (int i = 0; i < CurrentObj -> objSub.size(); i++) {
            // Если имя подчиненного объекта совпадает с ObjName
            if (CurrentObj -> objSub[i] -> getObjName() == ObjName) {
               // Присваиваем объекту object ссылку на найденный объект
```

```
            object = CurrentObj -> objSub[i];
            // Если строка Coordinate не пустая, вызываем рекурсивно метод
findObjByCoordinate для найденного объекта и обновленной строки Coordinate
            if (Coordinate != "") {
               return object -> findObjByCoordinate(object, Coordinate);
            }
            // Иначе возвращаем найденный объект
            return object;
         }
      }
   }
   // Возвращаем найденный объект
   return object;
}
// Метод установки сигнала и обработчика
void ClassBase :: setConnection(string signalName, TYPE_SIGNAL signal,
ClassBase* object, TYPE_HANDLER handler) {
   // Проход по всем соединениям
   for (int i = 0; i < connections.size(); i++) {
      // Если такое соединение уже существует, то прервать
      if (connections[i] -> signalName == signalName) {
         return;
      }
   }
   // Создание нового соединения и добавление его в список соединений
   connectionStats* newConnection = new connectionStats;
   newConnection -> signalName = signalName;
   newConnection -> signal = signal;
   newConnection -> baseClass = object;
   newConnection -> handler = handler;
   connections.push_back(newConnection);
}

// Метод удаления подключения
void ClassBase :: delConnection(string signalName) {
   // Проход по всем соединениям
   for (int i = 0; i < connections.size(); i++) {
      // Если нашли соединение с нужным именем, то удалить его
      if (connections[i] -> signalName == signalName) {
         connections.erase(connections.begin() + i);
         break;
      }
   }
}

// Метод отправки сигнала
string ClassBase :: emitSignal(string signalName, string info) {
   // Если состояние объекта равно 0, вернуть "null"
   if (objState == 0) {
      return "null";
   }
   string infoToReturn = "null";
   // Проход по всем соединениям
   for (int i = 0; i < connections.size(); i++) {
      // Если нашли соединение с нужным именем сигнала
```

```
        if (connections[i] -> signalName == signalName) {
           // Вызвать функцию-сигнал
           if (connections[i] -> signal != nullptr) {
              (this->*connections[i]->signal)(info);
           }
           // Вызвать обработчик
           if (connections[i] -> handler != nullptr) {
              infoToReturn    =    (connections[i]->baseClass->*connections[i]-
>handler)(info);
           }
           break;
        }
     }
     return infoToReturn;
  }
```

5.2 Файл ClassBase.h

Листинг 2 – ClassBase.h

```
  #ifndef __CLASSBASE_H
  #define __CLASSBASE_H
  #include <string>
  #include <vector>
  #include <iostream>
  using namespace std;
  class ClassBase; // Объявление класса ClassBase для использования его типа
  данных

  typedef  void  (ClassBase::*TYPE_SIGNAL)  (string&);  //  Определение  типа
  указателя  на  функцию-член  класса,  принимающей  строку  и  не  возвращающей
  значения
  typedef  string  (ClassBase::*TYPE_HANDLER)  (string);  //  Определение  типа
  указателя на функцию-член класса, принимающей строку и возвращающей строку

  #define  SIGNAL_D(signal_f)  (TYPE_SIGNAL)  (&signal_f);  //  Макрос  для
  преобразования имени функции-члена в указатель на неё для сигнала
  #define  HANDLER_D(handler_f)  (TYPE_HANDLER)  (&handler_f);  //  Макрос  для
  преобразования имени функции-члена в указатель на неё для обработчика

  class ClassBase {
     private:
        string objName; // Название объекта
        ClassBase* objHead = nullptr; // Родительский объект
        vector <ClassBase*> objSub; // Дочерние объекты
        int objState = 1; // Состояние объекта
        struct connectionStats { // Структура хранения информации о соединениях
           string signalName; // Название сигнала
           TYPE_SIGNAL  signal;  //  Указатель  на  функцию-член,  обрабатывающую
  сигнал
```

```
        ClassBase* baseClass; // Указатель на объект
        TYPE_HANDLER handler; // Указатель на функцию-член, обрабатывающую
сигнал
    };
    vector    <connectionStats*>    connections;    //    Вектор    структур
connectionStats для хранения соединений
  public:
    ClassBase(ClassBase*  ObjHead,  string  ObjName  =  "Root_Obj");  //
Конструктор класса с возможностью задать родительский объект и название
объекта
    string getObjName(); // Метод получения названия объекта
    ClassBase* getObjHead(); // Метод получения родительского объекта
    int getObjState(); // Метод получения состояния объекта
    void  setObjHead(ClassBase*  ObjHead);  //  Метод  задания  родительского
объекта
    void setObjName(string objName); // Метод задания названия объекта
    void setObjState(int ObjState); // Метод задания состояния объекта
    void  printTreeAndState(int  step);  //  Метод  для  вывода  информации  о
дереве объектов и их состояниях
    ClassBase*    findObjByCoordinate(ClassBase*    CurrentObj,    string
Coordinate); // Метод для поиска объекта по координате
    void  setConnection(string  signalName,  TYPE_SIGNAL  signal,  ClassBase*
object = nullptr, TYPE_HANDLER handler = nullptr); // Метод для установки
соединения между сигналом и обработчиком
    void delConnection(string signalName); // Метод для удаления соединения
    string  emitSignal(string  signalName,  string  info  =  "null");  //  Метод
для отправки сигнала
};
#endif
```

5.3 Файл Controller.cpp

Листинг 3 – Controller.cpp

```
#include "Controller.h"
// Метод класса Controller для обработки команды сигнала

Controller   ::   Controller(ClassBase*   objHead,   string   objName)   :
ClassBase(objHead,  objName)  {}  //инициализирует  Controller  с  помощью
конструктора базового класса ClassBase.

void Controller::commandSignal(string& commandLine) {
  // Создание указателя на объект Controller и на объект класса Vault
  ClassBase* ControllerObj = this;
  ClassBase* VaultObj = findObjByCoordinate(this->getObjHead(), "/Vault");
  // Проверка на команду "Turn off the safe"
  if (commandLine == "Turn off the safe") {
    return;
  }
  // Проверка на команду "CANCEL"
```

```
        if (commandLine == "CANCEL") {
           VaultObj->emitSignal("SetVaultState", "is ready to work");
           commandLine = "Ready to work";
           return;
        }
        // Проверка на команду "SHOWTREE"
        if (commandLine == "SHOWTREE") {
           this->getObjHead()->printTreeAndState(0);
           return;
        }
        // Объявление переменных command и value
        string command;
        string value;
        // Поиск позиции первого пробела в строке commandLine
        size_t pos = commandLine.find(" ");
        string commandLineDup = commandLine;
        // Извлечение команды и значения из commandLine
        command = commandLine.erase(pos, commandLine.size());
        value = commandLineDup.erase(0, pos + 1);
        // Обработка команды "BOX"
        if (command == "BOX") {
           // Проверка состояния хранилища Vault
           if (VaultObj->emitSignal("GetVaultState") != "is ready to work") {
              commandLine = "Error in the command sequence\nReady to work";
              return;
           }
           // Проверка состояния ячейки хранилища Vault
           if (VaultObj->emitSignal("GetVaultCellState", value) == "is opened") {
              commandLine = "The safe deposit box is open\nReady to work";
              return;
           }
           // Установка состояния хранилища Vault и выбор текущей ячейки
           VaultObj->emitSignal("SetVaultState", "is waiting client PIN");
           VaultObj->emitSignal("SetCurrentCellNum", value);
           commandLine = "Enter the code";
           return;
        }
        // Обработка команды "CLIENT_KEY"
        if (command == "CLIENT_KEY") {
           // Проверка состояния хранилища Vault
           if (VaultObj->emitSignal("GetVaultState") != "is waiting client PIN") {
              commandLine = "Error in the command sequence\nReady to work";
              VaultObj->emitSignal("SetVaultState", "is ready to work");
              return;
           }
           //  Сравнение  клиентского  PIN-кода  и  обновление  состояния  хранилища
Vault
        if       (ControllerObj->emitSignal("CompareClientPin",       VaultObj-
>emitSignal("GetCurrentCellNum") + " " + value) == "clientPinInCorrect") {
           commandLine = "The client is key is incorrect\nReady to work";
           VaultObj->emitSignal("SetVaultState", "is ready to work");
           return;
        }
        // Установка состояния "ожидание банковского PIN" и запрос кода банка
        VaultObj->emitSignal("SetVaultState", "is waiting bank PIN");
```

```
            commandLine = "Enter the bank code";
            return;
        }

        // Обработка команды "BANK_KEY" и дальнейшие операции
        if (command == "BANK_KEY") {
            // Проверка состояния хранилища Vault
            if (VaultObj->emitSignal("GetVaultState") != "is waiting bank PIN") {
                commandLine = "Error in the command sequence\nReady to work";
                VaultObj->emitSignal("SetVaultState", "is ready to work");
                return;
            }
            //  Сравнение банковского PIN-кода и обновление состояния хранилища
Vault
            if        (ControllerObj->emitSignal("CompareBankPin",        VaultObj-
>emitSignal("GetCurrentCellNum") + " " + value) == "bankPinInCorrect") {
                commandLine = "The bank is key is incorrect\nReady to work";
                VaultObj->emitSignal("SetVaultState", "is ready to work");
                return;
            }
            // Изменение состояния ячейки хранилища и обновление текущего состояния
            VaultObj->emitSignal("ChangeVaultCellState",                  VaultObj-
>emitSignal("GetCurrentCellNum"));
            VaultObj->emitSignal("SetVaultState", "is ready to work");
            commandLine    =    "The    safe    deposit    box    "    +    VaultObj-
>emitSignal("GetCurrentCellNum") + " is open\nReady to work";
            return;
        }

        // Обработка команды "CLOSE_BOX" и дальнейшие действия
        if (command == "CLOSE_BOX") {
            // Проверка состояния хранилища Vault
            if (VaultObj->emitSignal("GetVaultState") != "is ready to work") {
                commandLine = "Error in the command sequence\nReady to work";
                VaultObj->emitSignal("SetVaultState", "is ready to work");
                return;
            }
            // Изменение состояния ячейки хранилища и обновление текущего состояния
            VaultObj->emitSignal("ChangeVaultCellState",                  VaultObj-
>emitSignal("GetCurrentCellNum"));
            VaultObj->emitSignal("SetVaultState", "is ready to work");
            commandLine = "Ready to work";
            return;
        }
    }
    string Controller::vaultInfoHandler(string info) {
        return info;
    }
```

5.4 Файл ControllerDisplay.cpp

Листинг 4 – ControllerDisplay.cpp

```
#include "ControllerDisplay.h"
ControllerDisplay::ControllerDisplay(ClassBase* objHead, string objName) : ClassBase(objHead, objName) {}//инициализирует ControllerDisplay с помощью конструктора базового класса ClassBase.
string ControllerDisplay::controllerDisplayHandler(string info) {
   if (info == "SHOWTREE"){ // Если значение переменной info равно "SHOWTREE"
      cout << endl << "Turn off the safe"; // Вывести строку "Turn off the safe" на экран
      return "SHOWTREE"; // Вернуть строку "SHOWTREE"
   }
   cout << endl << info; // Вывести значение переменной info на экран
   if (info == "Turn off the safe"){ // Если значение переменной info равно "Turn off the safe"
      return "Turn off the safe"; // Вернуть строку "Turn off the safe"
   }
   return "null"; // Вернуть строку "null" если ни одно из условий не выполнено
}
```

5.5 Файл ControllerDisplay.h

Листинг 5 – ControllerDisplay.h

```
#ifndef __CONTROLLERDISPLAY_H
#define __CONTROLLERDISPLAY_H
#include "ClassBase.h"
// Объявление класса ControllerDisplay, который наследуется от класса ClassBase
class ControllerDisplay : public ClassBase {
   public:
      // Конструктор класса ControllerDisplay с параметрами objHead и objName, которые передаются в конструктор базового класса ClassBase
      ControllerDisplay(ClassBase* objHead, string objName);
      // Объявление метода controllerDisplayHandler, который возвращает строку и принимает строку info в качестве параметра
      string controllerDisplayHandler(string info);
};
#endif
```

5.6 Файл Controller.h

Листинг 6 – Controller.h

```
#ifndef __CONTROLLER_H
#define __CONTROLLER_H
#include "ClassBase.h"
class Controller : public ClassBase {
// Определение класса Controller, который наследуется от класса ClassBase
   public:
   // Область общедоступных членов класса
      Controller(ClassBase* objHead, string objName);
      // Конструктор класса Controller, который принимает указатель на объект
ClassBase и строку objName
      void commandSignal(string& commandLine);
      // Объявление метода commandSignal, который принимает ссылку на строку
commandLine в качестве параметра.
      string vaultInfoHandler(string info);
      // Объявление метода vaultInfoHandler, который принимает строку info и
возвращает строку.

};
#endif
```

5.7 Файл InfoNCommandReader.cpp

Листинг 7 – InfoNCommandReader.cpp

```
#include "InfoNCommandReader.h"
InfoNCommandReader::InfoNCommandReader(ClassBase*      objHead,      string
objName) :ClassBase(objHead, objName){}//инициализирует InfoNCommandReader с
помощью конструктора базового класса ClassBase.
//  Объявление  метода  getCinSignal  класса  InfoNCommandReader,  который
принимает строку по ссылке и записывает в нее ввод пользователя
void InfoNCommandReader::getCinSignal(string& info) {
   string line; // Объявление переменной line типа string
   getline(cin,  line);  //  Считывание  строки  из  стандартного  ввода  и
сохранение в переменную line
   info = line; // Присвоение считанной строки переменной info
}
//  Объявление  метода  getCinHandler  класса  InfoNCommandReader,  который
принимает строку и возвращает ее без изменений
string InfoNCommandReader::getCinHandler(string info) {
   return info; // Возвращение полученной строки без изменений
}
```

5.8 Файл InfoNCommandReader.h

Листинг 8 – InfoNCommandReader.h

```
#ifndef __INFONCOMMANDREADER__H
#define __INFONCOMMANDREADER__H
#include "ClassBase.h"
// Определение класса InfoNCommandReader, который наследуется от класса ClassBase
class InfoNCommandReader : public ClassBase {
  public:
     // Конструктор класса InfoNCommandReader, принимающий указатель на объект класса ClassBase и строку objName
     InfoNCommandReader(ClassBase* objHead, string objName); // Вызов конструктора базового класса ClassBase с передачей objHead и objName
     // Объявление метода getCinSignal, который принимает строку по ссылке
     void getCinSignal(string& info);
     // Объявление метода getCinHandler, который принимает строку и возвращает ее без изменений
     string getCinHandler(string info);
};
#endif
```

5.9 Файл main.cpp

Листинг 9 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include "System.h"
#include <iostream>
using namespace std;

int main()
{
  System RootObj(nullptr);  // Создание объекта класса System с передачей указателя nullptr в качестве аргумента конструктору
  RootObj.buildTree();  // Вызов метода buildTree() объекта RootObj, который строит дерево объектов
  return RootObj.startApp();   // Запуск приложения путем вызова метода startApp() объекта RootObj и возврат его результата
}
```

5.10 Файл Server.cpp

Листинг 10 – Server.cpp

```
#include "Server.h"
Server::Server(ClassBase* objHead, string objName) : ClassBase(objHead, objName) {}//инициализирует Server с помощью конструктора базового класса ClassBase.
string Server::createMassivesHandler(string vaultSize) {
  clientPinMas = new string[stoi(vaultSize)];   // Создание динамического массива строк clientPinMas размером, заданным параметром vaultSize
  bankPinMas = new string[stoi(vaultSize)];   // Создание динамического массива строк bankPinMas размером, заданным параметром vaultSize
  return "null";  // Возврат строки "null"
}

string Server::setPinsHandler(string info) {
  if (info == "Completing key entry") {  // Если значение параметра info равно "Completing key entry"
     return "Completing key entry";   // Возврат строки "Completing key entry"
  }
  int cellIndex;  // Объявление переменной cellIndex типа int
  string clientPin;  // Объявление переменной clientPin типа string
  string bankPin;  // Объявление переменной bankPin типа string
  size_t pos = info.find(" ");  // Поиск позиции первого пробела в строке info и сохранение его в переменной pos
  string infoDup = info;   // Создание копии строки info и сохранение в переменной infoDup
  cellIndex = stoi(infoDup.erase(pos, infoDup.size()));  // Преобразование подстроки до первого пробела в infoDup в int и сохранение в cellIndex
  info.erase(0, pos + 1);  // Удаление подстроки до первого пробела в строке info
  pos = info.find(" ");  // Поиск позиции первого пробела в строке info и сохранение его в переменной pos
  infoDup = info;  // Создание копии строки info и сохранение в переменной infoDup
  clientPin = infoDup.erase(pos, infoDup.size());  // Сохранение подстроки до первого пробела в infoDup в переменную clientPin
  info.erase(0, pos + 1);  // Удаление подстроки до первого пробела в строке info
  bankPin = info;  // Сохранение оставшейся части строки info в переменную bankPin
  clientPinMas[cellIndex - 1] = clientPin;    // Присваивание значения clientPin элементу массива clientPinMas с индексом cellIndex - 1
  bankPinMas[cellIndex - 1] = bankPin;   // Присваивание значения bankPin элементу массива bankPinMas с индексом cellIndex - 1
  return "null";  // Возврат строки "null"
}

string Server::clientPinCompareHandler(string info) {
  int cellIndex;  // Объявление переменной cellIndex типа int
  string clientPin;  // Объявление переменной clientPin типа string
  size_t pos = info.find(" ");  // Поиск позиции первого пробела в строке
```

```
info и сохранение его в переменной pos
   string infoDup = info;   // Создание копии строки info и сохране-ние в
переменной infoDup
   cellIndex = stoi(infoDup.erase(pos, infoDup.size()));   // Преобразование
подстроки до первого пробела в infoDup в int и сохранение в cellIndex
   clientPin = info.erase(0, pos + 1);  // Сохранение оставшейся части строки
info в переменную clientPin
   if (clientPinMas[cellIndex - 1] == clientPin) {  // Если значение элемента
массива clientPinMas с индексом cellIndex - 1 равно значению clientPin
      return "clientPinCorrect";  // Возврат строки "clientPinCorrect"
   }
   else {
      return "clientPinInCorrect";  // Возврат строки "clientPinInCorrect"
   }
}

string Server::bankPinCompareHandler(string info) {
   int cellIndex;  // Объявление переменной cellIndex типа int
   string bankPin;  // Объявление переменной bankPin типа string
   size_t pos = info.find(" ");  // Поиск позиции первого пробела в строке
info и сохранение его в переменной pos
   string infoDup = info;   // Создание копии строки info и сохранение в
переменной infoDup
   cellIndex = stoi(infoDup.erase(pos, infoDup.size()));   // Преобразование
подстроки до первого пробела в infoDup в int и сохранение в cellIndex
   bankPin = info.erase(0, pos + 1);  // Сохранение оставшейся части строки
info в переменную bankPin
   if (bankPinMas[cellIndex - 1] == bankPin) {   // Если значение элемента
массива bankPinMas с индексом cellIndex - 1 равно значению bankPin
      return "bankPinCorrect";  // Возврат строки "bankPinCorrect"
   }
   else {
      return "bankPinInCorrect";  // Возврат строки "bankPinInCorrect"
   }
}
```

5.11 Файл Server.h

Листинг 11 – Server.h

```
#ifndef __SERVER__H
#define __SERVER__H
#include "ClassBase.h"
class Server : public ClassBase {
   private:
      string* clientPinMas;    // Указатель на массив строк для хранения
клиентских PIN-кодов
      string* bankPinMas;    // Указатель на массив строк для хранения
банковских PIN-кодов
   public:
      Server(ClassBase* objHead, string objName);    // Конструктор класса
```

```
Server, инициализирующий объект базового класса ClassBase
     string createMassivesHandler(string vaultSize);  // Метод для создания
массивов заданного размера
     string setPinsHandler(string info);  // Метод для установки PIN-кодов в
массивы
     string clientPinCompareHandler(string info);   // Метод для сравнения
клиентского PIN-кода
     string   bankPinCompareHandler(string);     //   Метод   для   сравнения
банковского PIN-кода
};
#endif
```

5.12 Файл System.cpp

Листинг 12 – System.cpp

```
#include "System.h"
#include "InfoNCommandReader.h"
#include "Controller.h"
#include "Vault.h"
#include "VaultCell.h"
#include "Server.h"
#include "ControllerDisplay.h"
#include <iostream>
#include <string>
using namespace std;
System::System(ClassBase*    objHead)    :    ClassBase(objHead,    "System")
{}//инициализирует System с помощью конструктора базового класса ClassBase.
void System::buildTree() {
   //------------------Создание дерева объектов
   ClassBase* RootObj = this;
   ClassBase* Obj = nullptr;
   //Создание объекта считывания строк
   ClassBase*     InputReaderObj     =     new     InfoNCommandReader(RootObj,
"InfoNCommandReader");
   //Создание объекта пульта управления
   ClassBase* ControllerObj = new Controller(RootObj, "Controller");
   //Создание объекта хранилища
   ClassBase* VaultObj = new Vault(RootObj, "Vault");
   InputReaderObj->setConnection("GetInputNSetVaultSize",
getSignal("getCinSignal"), VaultObj, getHandler("vaultSizeHandler"));
   InputReaderObj->emitSignal("GetInputNSetVaultSize");
   //Цикл создания ячеек хранилища
   VaultObj->setConnection("CellCreation",  getSignal("getCellAmountSignal"),
VaultObj, getHandler("vaultCellCreateHandler"));
   VaultObj->emitSignal("CellCreation");
   //Создание сервера
   ClassBase* ServerObj = new Server(RootObj, "Server");
   VaultObj->setConnection("CreateServerMassives",
getSignal("getCellAmountSignal"), ServerObj, getHandler("createMassives"));
```

```
   VaultObj->emitSignal("CreateServerMassives");
   //Создание дисплея пульта управления
   ClassBase*   ControllerDisplayObj   =   new   ControllerDisplay(RootObj,
"ControllerDisplay");
   //-----------------Установка необходимых связей
   InputReaderObj->setConnection("SetPins",        getSignal("getCinSignal"),
ServerObj, getHandler("setPinsHandler"));
   InputReaderObj->setConnection("GetLine",        getSignal("getCinSignal"),
InputReaderObj, getHandler("getCinHandler"));
   ControllerObj->setConnection("ProcessCommandNPrintResult",
getSignal("commandSignal"),                          ControllerDisplayObj,
getHandler("controllerDisplayHandler"));
   VaultObj->setConnection("GetVaultCellState",
getSignal("getVaultCellStateSignal"),                          ControllerObj,
getHandler("vaultInfoHandler"));
   VaultObj->setConnection("GetVaultState", getSignal("getVaultStateSignal"),
ControllerObj, getHandler("vaultInfoHandler"));
   VaultObj->setConnection("GetCurrentCellNum",
getSignal("getCurrentCellNumSignal"),                          ControllerObj,
getHandler("vaultInfoHandler"));
   VaultObj->setConnection("ChangeVaultCellState",
getSignal("changeVaultCellStateSignal"));
   VaultObj->setConnection("SetVaultState",
getSignal("setVaultStateSignal"));
   VaultObj->setConnection("SetCurrentCellNum",
getSignal("setCurrentCellNumSignal"));
   ControllerObj->setConnection("CompareClientPin",    nullptr,    ServerObj,
getHandler("clientPinCompareHandler"));
   ControllerObj->setConnection("CompareBankPin",     nullptr,     ServerObj,
getHandler("bankPinCompareHandler"));

   // Настройка системы (ввод PIN-кодов для ячеек)
   string inputReaderAnswer = InputReaderObj->emitSignal("SetPins");
   while (inputReaderAnswer != "Completing key entry") {
      inputReaderAnswer = InputReaderObj->emitSignal("SetPins");
   }
}

void System::commandHandler() {
   // Обработка команд пользователей
   ClassBase* RootObj = this;
   ClassBase*       InputReaderObj       =       findObjByCoordinate(RootObj,
"/InfoNCommandReader");
   ClassBase* ControllerObj = findObjByCoordinate(RootObj, "/Controller");
   cout << "Ready to work";
   string            controllerAnswer            =            ControllerObj-
>emitSignal("ProcessCommandNPrintResult",                    InputReaderObj-
>emitSignal("GetLine"));
   while (controllerAnswer != "SHOWTREE" && controllerAnswer != "Turn off the
safe") {
      controllerAnswer                     =                    ControllerObj-
>emitSignal("ProcessCommandNPrintResult",                    InputReaderObj-
>emitSignal("GetLine"));
   }
}
```

```
int System::startApp() {
   // Запуск обработки команд
   this->commandHandler();
   return 0;
}

TYPE_SIGNAL System::getSignal(string methodName) {
   // Возвращает сигнал для указанного метода
   if (methodName == "getCinSignal") {
      return SIGNAL_D(InfoNCommandReader::getCinSignal);
   }
   else if (methodName == "getCellAmountSignal") {
      return SIGNAL_D(Vault::getCellAmountSignal);
   }
   else if (methodName == "commandSignal") {
      return SIGNAL_D(Controller::commandSignal);
   }
   else if (methodName == "getVaultCellStateSignal") {
      return SIGNAL_D(Vault::getVaultCellStateSignal);
   }
   else if (methodName == "getVaultStateSignal") {
      return SIGNAL_D(Vault::getVaultStateSignal);
   }
   else if (methodName == "getCurrentCellNumSignal") {
      return SIGNAL_D(Vault::getCurrentCellNumSignal);
   }
   else if (methodName == "changeVaultCellStateSignal") {
      return SIGNAL_D(Vault::changeVaultCellStateSignal);
   }
   else if (methodName == "setVaultStateSignal") {
      return SIGNAL_D(Vault::setVaultStateSignal);
   }
   else if (methodName == "setCurrentCellNumSignal") {
      return SIGNAL_D(Vault::setCurrentCellNumSignal);
   }
   else {
      return nullptr;
   }
}
TYPE_HANDLER System::getHandler(string methodName) {
   // Функция getHandler возвращает указатель на обработчик для указанного
метода
   if (methodName == "getCinHandler") {
      return HANDLER_D(InfoNCommandReader::getCinHandler);
   } else if (methodName == "vaultSizeHandler") {
      return HANDLER_D(Vault::vaultSizeHandler);
   } else if (methodName == "vaultCellCreateHandler") {
      return HANDLER_D(Vault::vaultCellCreateHandler);
   } else if (methodName == "createMassives") {
      return HANDLER_D(Server::createMassivesHandler);
   } else if (methodName == "setPinsHandler") {
      return HANDLER_D(Server::setPinsHandler);
   } else if (methodName == "controllerDisplayHandler") {
      return HANDLER_D(ControllerDisplay::controllerDisplayHandler);
```

```
   } else if (methodName == "vaultInfoHandler") {
      return HANDLER_D(Controller::vaultInfoHandler);
   } else if (methodName == "clientPinCompareHandler") {
      return HANDLER_D(Server::clientPinCompareHandler);
   } else if (methodName == "bankPinCompareHandler") {
      return HANDLER_D(Server::bankPinCompareHandler);
   } else {
      return nullptr; // Возвращаем nullptr, если обработчик не найден
   }
}
```

5.13 Файл System.h

Листинг 13 – System.h

```
#ifndef __SYSTEM__H
#define __SYSTEM__H
#include "ClassBase.h"
class System : public ClassBase {
// Объявление класса System, который наследуется от класса ClassBase
  public:
     System(ClassBase* objHead);
     // Конструктор класса System, принимающий указатель на объект класса
ClassBase и вызывающий конструктор базового класса с передачей указателя и
строкового значения "System"
     void buildTree();
     // Объявление метода buildTree без возвращаемого значения
     void commandHandler();
     // Объявление метода commandHandler без возвращаемого значения
     int startApp();
     // Объявление метода startApp с возвращаемым значением типа int
     TYPE_SIGNAL getSignal(string methodName);
     // Объявление метода getSignal, принимающего строку methodName и
возвращающего тип TYPE_SIGNAL
     TYPE_HANDLER getHandler(string methodName);
     // Объявление метода getHandler, принимающего строку methodName и
возвращающего тип TYPE_HANDLER
};
#endif
```

5.14 Файл VaultCell.cpp

Листинг 14 – VaultCell.cpp

```
#include "VaultCell.h"
```

```
VaultCell::VaultCell(ClassBase* objHead, string objName, int cellNumber) :
ClassBase(objHead, objName) {
   // Определение конструктора класса VaultCell, принимающего указатель на
объект класса ClassBase, строку objName и целочисленное значение cellNumber
   // Начало инициализации конструктора с вызовом конструктора базового
класса ClassBase с параметрами objHead и objName
   this->cellNumber = cellNumber;
   // Присвоение значению приватного поля cellNumber значение параметра
cellNumber
}
```

5.15 Файл VaultCell.h

Листинг 15 – VaultCell.h

```
#ifndef __VAULTCELL__H
#define __VAULTCELL__H
#include "ClassBase.h"
class VaultCell : public ClassBase {
// Объявление класса VaultCell, который наследуется от класса ClassBase
   private:
      int cellNumber = 0;
      // Приватное поле класса VaultCell типа int с инициализацией значения 0
   public:
      VaultCell(ClassBase* objHead, string objName, int cellNumber);
      // Публичный конструктор класса VaultCell, принимающий указатель на
объект класса ClassBase, строку objName и целочисленное значение cellNumber
};
#endif
```

5.16 Файл Vault.cpp

Листинг 16 – Vault.cpp

```
#include "Vault.h"
#include "VaultCell.h"
Vault::Vault(ClassBase*  objHead,  string  objName)  :  ClassBase(objHead,
objName) {}//инициализирует Vault с помощью конструктора базового класса
ClassBase.
string Vault::vaultSizeHandler(string vaultSize) {
      // Метод для обработки информации о размере хранилища
      size_t pos = vaultSize.find(" ");
      // Находим позицию первого пробела в строке vaultSize
      string vaultSizeDup = vaultSize;
      // Создаем копию строки vaultSize
```

```
        int rowAmount = stoi(vaultSize.erase(0, pos));
        // Преобразуем подстроку до первого пробела в целое число - это
количество рядов
        int columnAmount = stoi(vaultSizeDup.erase(pos,
vaultSizeDup.length()));
        // Преобразуем подстроку после первого пробела в целое число - это
количество столбцов
        this->cellAmount = rowAmount * columnAmount;
        // Вычисляем общее количество ячеек
        this->rowAmount = rowAmount;
        // Устанавливаем количество рядов
        this->columnAmount = columnAmount;
        // Устанавливаем количество столбцов
        return "null";
        // Возвращаем пустую строку
    }

    string Vault::vaultCellCreateHandler(string vaultSize) {
      // Метод для создания ячеек хранилища
      ClassBase* Obj;
      // Создаем указатель на объект класса ClassBase
      cellMas = new ClassBase*[cellAmount];
      // Выделяем память под массив указателей на объекты класса ClassBase
      for (int i = 0; i < stoi(vaultSize); i++) {
        // Цикл по количеству ячеек
        Obj = new VaultCell(this, "VaultCell_" + to_string(i + 1), i + 1);
        // Создаем объект класса VaultCell и добавляем его в массив cellMas
        cellMas[i] = Obj;
      }
      return "null";
      // Возвращаем пустую строку
    }

    void Vault::getCellAmountSignal(string& info) {
      // Метод для получения информации о количестве ячеек
      info = to_string(cellAmount);
      // Записываем в переменную info количество ячеек в виде строки
    }

    void Vault::getVaultCellStateSignal(string& info) {
      // Метод для получения состояния ячейки хранилища
      int cellNum = stoi(info);
      // Преобразуем строку в число - номер ячейки
      if (cellMas[cellNum - 1]->getObjState() == 1) {
        info = "is closed";
        // Если состояние ячейки равно 1, записываем "is closed" в переменную
info
      }
      if (cellMas[cellNum - 1]->getObjState() == 2) {
        info = "is opened";
        // Если состояние ячейки равно 2, записываем "is opened" в переменную
info
      }
    }
```

```
void Vault::getVaultStateSignal(string& info) {
   // Метод для получения состояния хранилища
   if (this->getObjState() == 1) {
      info = "is ready to work";
   // Если состояние хранилища равно 1, записываем "is ready to work" в
переменную info
   }
   if (this->getObjState() == 2) {
      info = "is waiting client PIN";
      // Если состояние хранилища равно 2, записываем "is waiting client PIN"
в переменную info
   }
   if (this->getObjState() == 3) {
      info = "is waiting bank PIN";
      // Если состояние хранилища равно 3, записываем "is waiting bank PIN" в
переменную info
   }
}

void Vault::getCurrentCellNumSignal(string& info) {
   // Метод для получения текущего номера ячейки
   info = currentCellNum;
   // Записываем текущий номер ячейки в переменную info
}

void Vault::changeVaultCellStateSignal(string& info) {
   // Метод для изменения состояния ячейки хранилища
   int cellNum = stoi(info);
   // Преобразуем строку в число - номер ячейки
   if (cellMas[cellNum - 1]->getObjState() == 1) {
      cellMas[cellNum - 1]->setObjState(2);
      // Если состояние ячейки равно 1, устанавливаем состояние 2
   } else if (cellMas[cellNum - 1]->getObjState() == 2) {
      cellMas[cellNum - 1]->setObjState(1);
      // Если состояние ячейки равно 2, устанавливаем состояние 1
   }
}

void Vault::setVaultStateSignal(string& info) {
   // Метод для установки состояния хранилища
   if (info == "is ready to work") {
      this->setObjState(1);
      // Если переданное состояние "is ready to work", устанавливаем
состояние хранилища 1
   }
   if (info == "is waiting client PIN") {
      this->setObjState(2);
      // Если переданное состояние "is waiting client PIN", устанавливаем
состояние хранилища 2
   }
   if (info == "is waiting bank PIN") {
      this->setObjState(3);
      // Если переданное состояние "is waiting bank PIN", устанавливаем
состояние хранилища 3
   }
```

```
}

void Vault::setCurrentCellNumSignal(string& info) {
   // Метод для установки текущего номера ячейки
   currentCellNum = info;
   // Устанавливаем текущий номер ячейки
}
```

5.17 Файл Vault.h

Листинг 17 – Vault.h

```
#ifndef __VAULT__H
#define __VAULT__H
#include "ClassBase.h"
class Vault : public ClassBase {
// Объявление класса Vault, унаследованного от класса ClassBase
   private:
      ClassBase** cellMas;
      int cellAmount = -1;
      int rowAmount = -1;
      int columnAmount = -1;
      string currentCellNum = "";
      // Приватные члены класса Vault: указатель на указатель объектов класса ClassBase, целочисленные переменные cellAmount, rowAmount, columnAmount и строковая переменная currentCellNum
   public:
      Vault(ClassBase* objHead, string objName);
      // Публичный конструктор класса Vault, принимающий указатель на объект класса ClassBase и строку objName, вызывающий конструктор базового класса ClassBase с параметрами objHead и objName

      string vaultSizeHandler(string vaultSize);
      string vaultCellCreateHandler(string vaultSize);
      void getCellAmountSignal(string& info);
      void getVaultCellStateSignal(string& info);
      void getVaultStateSignal(string& info);
      void getCurrentCellNumSignal(string& info);
      void changeVaultCellStateSignal(string& info);
      void setVaultStateSignal(string& vaultState);
      void setCurrentCellNumSignal(string& currentCellNum);
      // Публичные методы класса Vault для обработки информации о размере хранилища, создания ячеек хранилища, получения информации о количестве ячеек, состоянии ячейки, состоянии хранилища, текущем номере ячейки, изменении состояния ячейки, установке состояния хранилища и установке текущего номера ячейки
};
#endif
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 45.

Таблица 45 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
3 3 1 100001 1001001 2 200001 2002001 3 300001 3003001 4 400001 4004001 5 500001 5005001 6 600001 6006001 7 700001 7007001 8 800001 8008001 9 900001 9009001 Completing key entry BOX 3 CLIENT_KEY 300001 BANK_KEY 3003001 BOX 3 BOX 1 CLIENT_KEY 100001 CLIENT_KEY 200001 BOX 1 CLIENT_KEY 100017 CANCEL BOX 1 CLIENT_KEY 100001 BANK_KEY 1003001 Turn off the safe	Ready to work Enter the code Enter the bank code The safe deposit box 3 is open Ready to work The safe deposit box is open Ready to work Enter the code Enter the bank code Error in the command sequence Ready to work Enter the code The client is key is incorrect Ready to work Ready to work Enter the code Enter the bank code The bank is key is incorrect Ready to work Turn off the safe	Ready to work Enter the code Enter the bank code The safe deposit box 3 is open Ready to work The safe deposit box is open Ready to work Enter the code Enter the bank code Error in the command sequence Ready to work Enter the code The client is key is incorrect Ready to work Ready to work Enter the code Enter the bank code The bank is key is incorrect Ready to work Turn off the safe
3 3 1 100001 1001001 2 200001 2002001 3 300001 3003001 4 400001 4004001 5 500001 5005001 6 600001 6006001 7 700001 7007001 8 800001 8008001 9 900001 9009001 Completing key entry BOX 4 CLIENT_KEY 400001 BANK_KEY 4004001 Turn off the safe	Ready to work Enter the code Enter the bank code The safe deposit box 4 is open Ready to work Turn off the safe	Ready to work Enter the code Enter the bank code The safe deposit box 4 is open Ready to work Turn off the safe
3 3 1 100001 1001001	Ready to work Enter the code	Ready to work Enter the code

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
2 200001 2002001 3 300001 3003001 4 400001 4004001 5 500001 5005001 6 600001 6006001 7 700001 7007001 8 800001 8008001 9 900001 9009001 Completing key entry BOX 8 CLIENT_KEY 800001 BANK_KEY 8008002 Turn off the safe	Enter the bank code The bank is key is incorrect Ready to work Turn off the safe	Enter the bank code The bank is key is incorrect Ready to work Turn off the safe

ЗАКЛЮЧЕНИЕ

Была осуществлена разработка системы, реализующая работу банковского сейфа, каждая ячейка которой открывается при наличии двух ключей. В процессе реализации задачи было использовано 8 классов: ClassBase, System, InfoNCommandReader, Controller, Vault, VaultCell, Server, ControllerDisplay, каждый из которых иммитирует определенную часть системы банковского сейфа с двумя ключами. Все необходимые методы, сигналы и обработчки были реализованы. Функционал программы протестирован созданными тестами.

В ходе выполнения работы были усвоены навыки объектно-ориентированного программирования и разработки на языке C++. Получен опыт применения сигналов и обработчиков, составления документации, описания блок-схем и алгоритмов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19 Единая система программной документации.

2. Методическое пособие студента для выполнения 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).

3. 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).

4. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2019. — 624 с.

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).