МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

ОТЧЕТ

по лабораторной работе №1

по дисциплине

Шаблоны проектирования программного обеспечения

РУКОВОДИТЕЛЬ:

_____ _____Жевнерчук Д.В,__
(подпись) (фамилия, и.,о.)

СТУДЕНТ:

_____ ___Сапожников В.О._

_____ _____Сухоруков В.А.____

_____ ____Мосташов В.С.____
(подпись) (фамилия, и.,о.)

____19-ИВТ-3_____
(шифр группы)

Работа защищена «___»_____

С оценкой _____

Нижний Новгород 2021

Вариант 8.

Реализуйте консольную утилиту, позволяющую создавать графовые структуры, добавлять узлы и связи, причем каждый узел определяется именем и типом, а каждая связь — именем узла источника, именем узла приемника, типом. Для узлов определены следующие типы: класс, индивид, атрибут, значение. Для связей определены следующие типы: объектное свойство, свойство данных.

Приложение должно позволять:

1. Создавать одиночные узлы-классы.
2. Создавать нескольких индивидов одного класса, при этом свойство данных «ИмеетИндивида» должно формироваться автоматически и для каждого индивида автоматически создается атрибут «Идентификатор» с уникальным номером в пределах всех узлов-атрибутов текущего индивида.
3. Создание нескольких подклассов одного класса, при этом объектное свойство «Подкласс» должно формироваться автоматически

Полученный граф необходимо распечатать в консоли в произвольной, но понятной текстовой форме.

Проектное решение

Обоснование выбора паттернов

Поскольку целью работы является создание графовой структуры, то в качестве основного паттерна проектирования был выбран компоновщик. Основным классом является абстрактный класс Node, который содержит основные поля и методы необходимые для работы с узлами.

Производными классами являются:

1) ClassNode - может являться корнем, классом или подклассом графа.

2) IndividualMode – узел-индивид, является потомком узла типа ClassNode.

3) AttributeNode – является потомком узла типа IndividualMode и содержит название одного его атрибута.

4) ValueNode – содержит одно значение атрибута, является наследником узла типа AttributeNode.

Для связи узлов создан пакет Property. Реализация больше всего похожа на абстрактную фабрику, т. к. при необходимости можно породить новые типы (классы) рёбер.

Для создания графовой структуры реализован класс GraphBuilder. Является подобием автомата. Решено не использовать паттерн Строитель для него, т. к. граф имеет малое кол-во типов узлов с малым кол-вом параметров. Использования Строителя в данном случае привело бы к порождению новых классов, которые только бы усложнили код.

Для вывода графа в консоль в удобном для прочтения виде, создан класс GrapgPrinter. В основе его работы лежит паттерн singleton (одиночка). Поскольку приложение является однопоточным и вывод самой структуры производиться только 1 раз, то нет необходимости создавать больше одного объекта данного типа.

……

На рис 1. приведена диаграмма классов.

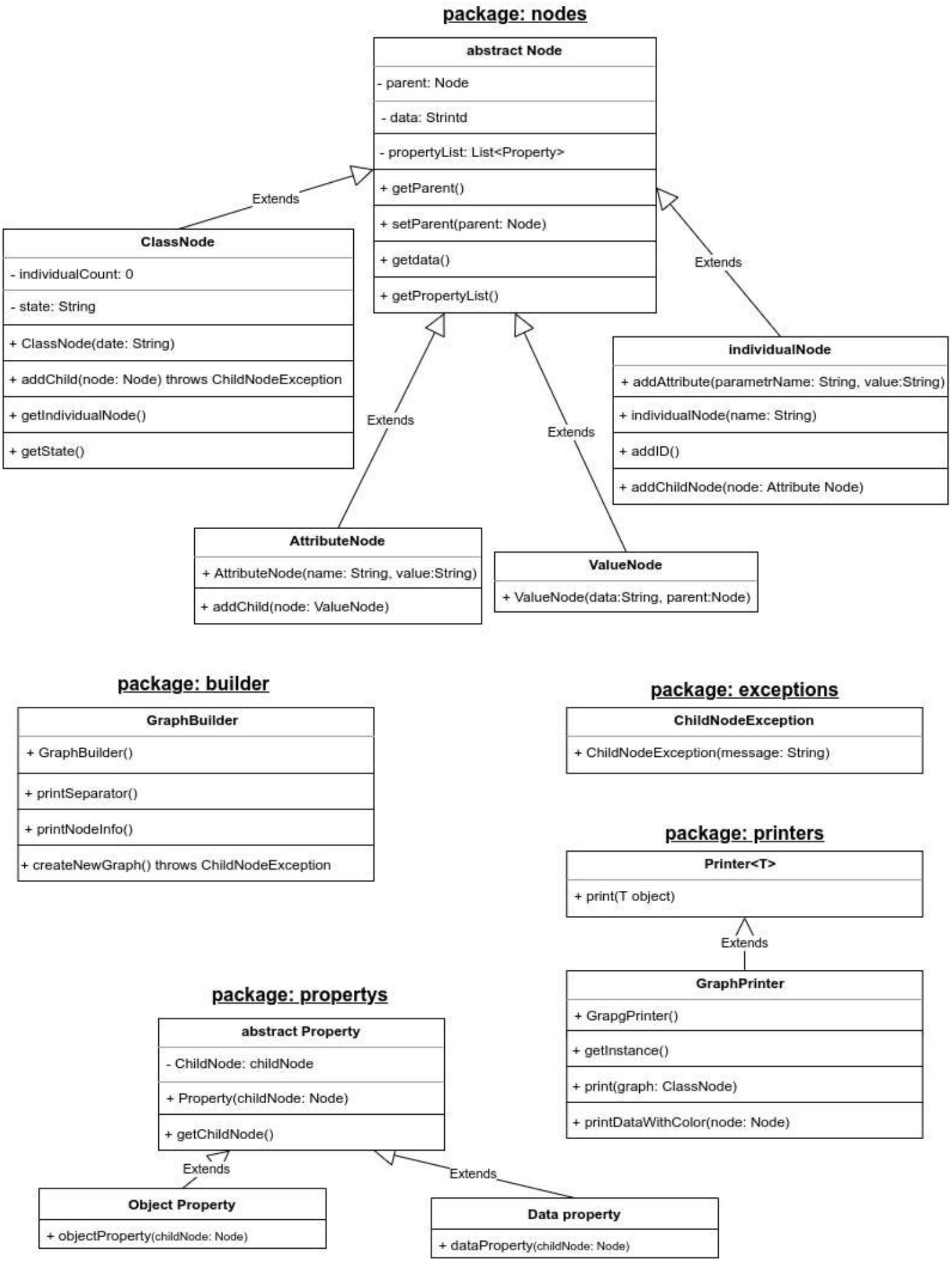

Рис. 1 – Диаграмма классов

Приложение 1

Программный код

Node.java

```
package nodes;
import propertys.Property;
import java.util.LinkedList;
import java.util.List;
/**Абстрактный класс, содержащий основные поля
 * и методы узлов графовой структуры
 * @see ClassNode
 * @see IndividualNode
 * @see AttributeNode
 * @see ValueNode
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @author Valerii Sukhorukov    19-IVT-3
 * @author Vyacheslav Mostashov  19-IVT-3
 * */
public abstract class Node
{
    protected Node parent;    //Поле для хранении ссылки на родительский узел
    protected String data;    //Поле для хранения данных

    //Список связей с дочерними узлами
    protected List<Property> propertyList = new LinkedList<>();

    /**Метод для получения ссылки на родительский узел
     * @return - ссылка на родительский узел
     * */
    public Node getParent(){
        return parent;
    }

    /**Метод для задания ссылки на родительский узел
     * @param parent - ссылка на родительский узел
     * */
    public void setParent(Node parent) {
        this.parent = parent;
    }

    /**Метод для получения данных из узла
     * @return - данные
     * */
    public String getData(){
        return data;
    }

    /* Метод для получения ссылки на список связей с
     * дочерними узлами
     * @return - список связей с дочерними узлами
     * */
    public List<Property> getPropertyList(){
        return propertyList;
    }
}
```

ClassNode.java

```
package nodes;

import exceptions.ChildNodeException;
import propertys.DataProperty;
import propertys.ObjectProperty;
import propertys.Property;
```

```java
/*Главный узел графовой структуры - служит корнем графа
 * и/или родителем для узлов типа IndividualNode
 * @see Node
 * @see IndividualNode
 * @see ObjectProperty
 * @see exceptions.ChildNodeException
 */
public class ClassNode extends Node
{
    private int individualCount = 0;     //счетчик индивидов для которых данный
                                         //узел является родительским

    private String state;                //Поле отвечающее за состояния данного узла

    /**Конструктор с параметрами.
     * Поскольку нет родителя, то узел
     * приобритает состояние класс
     * @param data    - имя данного узла
     * */
    public ClassNode(String data) {
        this.data = data;
        this.state = "Класс";
    }

    /**Медот для добавления дочернего узла
     * Формирует связь - свойство объекта
     * @param node                - дочерний узел
     * @throws ChildNodeException - исключение созданий связей
     *                              между узлами в графовой структуре
     * */
    public void addChild (Node node) throws ChildNodeException{
        //Если дочерний узел не принадлежит типу ClassNode или IndividualNode,
        //то выбрасываем исключение связей узлов
        if ((!(node instanceof ClassNode)) && (!(node instanceof IndividualNode)) {
            throw new ChildNodeException("Для узла типа" + this.getClass() +
                    "дочерним узлом могут быть узлы: " + ClassNode.class + " или "
                    + IndividualNode.class);
        }

        Property property;

        //Если дочерний узел принадлжеит классу IndividualNode
        if (node instanceof IndividualNode){
            //Установка родителя при помощи метода setParent класса
            //IndividualNode, где конкретно указано какой тип
            //Родителя может быть.
            //при использовании общего метода setParent последующий метод
            //addID бросает исключение
            node.setParent(this);

            //Увеличиваем счетчик индивидов для данного узла
            individualCount++;

            //Для узла индивида автоматически добавляем поле ID
            ((IndividualNode) node).addID();

            //Если в состоянии узла еще не указано, что он имеет
            //индивида к текущему состоянию приписываем:
            //имеет индивида
            if (!this.state.contains(", имеет индивида")){
                this.state = this.state + ", имеет индивида";
            }

            //Создание св-ва данных
            property = new DataProperty(node);
        }
```

```
            //Иначе если тип переданного узла ClassNode то у него устанавливается
            //статус Подкласс
            else{
                node.setParent(this);
                ((ClassNode) node).state = "Подкласс";

                //Если у текущего узла не указано в состоянии, что он имеет подкласс
                //то дописываем это
                if (!this.state.contains(", имеет подкласс")){
                    this.state = this.state + ", имеет подкласс";
                }

                //создание объектного свойства
                property = new ObjectProperty(node);
            }

            propertyList.add(property);
        }

        /*Метод для получения счетчика индивидов данного класса (узла)
         * @return значение счетчика индивидов
         * */
        public int getIndividualCount() {
            return this.individualCount;
        }



        /*Метод для получения состояния данного узла
         * @return значение счетчика индивидов
         * */
        public String getState(){
            return state;
        }
    }
```

IndividualNode.java

```
package nodes;

import propertys.DataProperty;

/*Класс Индивид - экземпляр класс/подкласса типа
 * ClassNode
 * @see ClassNode
 * @see AttributeNode
 * @see ValueNode
 * */
public class IndividualNode extends Node{
    /*Метод для добавления атрибута индивиду
     * Создает узел атрибут с именем атрибута,
     * так же создается узел
     * Формирует связь типа DataProperty
     *
     * @param parameterName - имя параметра
     * @param value         - значение параметра
     * */
    public void addAttribute(String parameterName, String value) {
        AttributeNode node = new AttributeNode(parameterName, value);
        node.setParent(this);
        addChildNode(node);
    }

    /*Конструктор с параметрами
     *
     * @param name    - имя данного индивида
     * */
```

```
    public IndividualNode(String name){
        this.data = name;
    }

    /*Метод для получения ID от родителя
     */
    public void addID(){
       addAttribute("ID",Integer.toString(((ClassNode)parent).getIndividualCount()));
    }

    /*Метод для добавления дочернего узла
     * Потомком данного вида узла может быть только узел типа
     * AttributeNode
     * @param node - ссылка на потомка
     * */
    private void addChildNode(AttributeNode node){
        DataProperty property = new DataProperty(node);
        propertyList.add(property);
    }
}
```

AttributeNode.java

```
package nodes;

import propertys.DataProperty;

/*Узел атрибут
 * Фомрмируется при вывозе метод addAttribute
 * у узла типа IndividualNode
 * Содержит значение Имя атрибута и ссылку на значение
 * @see Node
 * @see ValueNode
 * */
public class AttributeNode extends Node{
    /* Конструктор с параметрами
     * @param name  - имя данного атрибута
     * @param value - значение, которое будет передано узлу ValueNode
     * */
    public AttributeNode(String name, String value){
        this.data = name;
        ValueNode valueNode = new ValueNode(value, this);
        addChildNode(valueNode);
    }

    /* Метод для добавления дочернего узла
     * Дочерним узлом для данного типа может быть только узел
     * типа ValueNode
     * Формирует связь типа DataProperty
     * @param node - дочерний узел типа ValueNode
     * */
    public void addChildNode(ValueNode node) {
        DataProperty property = new DataProperty(node);
        propertyList.add(property);
    }
}
```

ValueNode.java

```
package nodes;

/**Узел значение
 * Фомрмируется при вывозе метод addAttribute
 * у узла типа IndividualNode
 * Содержит значение Атрибута
 * @see nodes.AttributeNode
 * */
```

```
public class ValueNode extends Node{
    /**Конструктор с параметрами
     * @param data   - значение данного узла
     * @param parent - родитель данного узла
     * */
    public ValueNode(String data, Node parent){
        this.parent = parent;
        this.data = data;
        this.propertyList = null;
    }
}
```

Property.java

```
package propertys;

import nodes.Node;

/**
 * Абстрактный класс, содержащий основные поля
 * и методы связей - свойств между узлами графовой структуры.
 *
 * @see DataProperty
 * @see ObjectProperty
 * @see Node
 * */
public abstract class Property{
    protected Node childNode;    //Ссылка на дочерний узел

    /* Конструктор с параметром
     * @param childNode - ссылка на узел, с которым устанавливается
     *                    связь
     * */
    public Property(Node childNode){
        this.childNode = childNode;
    }

    /*Метод для получения ссылки на дочерний узел
     * @return - ссылка на дочерний узел
     * */
    public Node getChildNode(){
        return childNode;
    }
}
```

GraphBuilder.java

```
package builder;

import exceptions.ChildNodeException;
import nodes.ClassNode;
import nodes.IndividualNode;
import nodes.Node;

import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

/* Класс, реализующий автомат для создания графовой структуры
 * @see nodes.Node
 * @see nodes.ClassNode
 * @see nodes.IndividualNode
 * @see nodes.AttributeNode
 * @see nodes.ValueNode
 * */
public class GraphBuilder{
    //Константы для хранения последовательностей для
```

```java
//изменения цвета текста в консоли
private static final String RESET = "\u001B[0m";
private static final String RED = "\u001B[31m";
private static final String PURPLE = "\u001B[35m";
private static final String CYAN = "\u001B[36m";
private static final String GREEN = "\u001B[32m";

/**
 * Поскольку для создания графа нет необходимости
 * создавать объект данного класса, то запрещаем
 * создание объекта
 * */
private GraphBuilder() { }

/* Метод выводящий разделитель при создании узлов
 * Скрыт, т.к. не используется напрямую
 * */
private static void printSeparator()
{
    System.out.println("-----------------------------------------------------");
    System.out.println("\t\t\t\t\t\t\t\t\t" + GREEN + "Создание нового узла" +
                       RESET);
    System.out.println("-----------------------------------------------------");
}

/*Метод выводящий информацию о переданном узле.
 * Скрыт, т.к. не используется напрямую
 * */
private static void printNodeInfo(Node node) {
    //Для каждого создающегося в данный момент узла
    //выводим информацию о родителе
    System.out.print(CYAN + "Родитель: " + RESET + node.getData());

    //Если тип данного узла ClassNode, то выводим еще и состояние
    if (node instanceof ClassNode){
        System.out.print(" | " + PURPLE + "Статус родителя: " + RESET +
                         ((ClassNode)node).getState());
    }
    System.out.println();
    System.out.println("-----------------------------------------------------");
}

/*"Автомат создания графа (но это не точно автомат)"
 * @return - ссылку на новую графовую структуру
 * */
public static ClassNode createNewGraph() throws ChildNodeException {
    //"Состояние автомата"
    //Создаем очередь для хранения узлов
    Queue<Node> nodeQueue = new LinkedList<>();

    //Создание ссылки на узел общего типа
    Node tempNode;

    //создаем ссылку на корень
    ClassNode graphRoot = null;

    //Открытие потока ввода
    Scanner scanner = new Scanner(System.in);

    String input;    //Временные ссылки для хранения
    String name;     //введенных строк

    printSeparator();

    //Создание 1ого узла - корня графовой структуры
    //Механизм do while предалагает пользователю повторный ввод
    //при неверных введенных данных
```

```
do {
    System.out.println("1. - Создать узел типа Класс(ClassNode)");
    System.out.println(RED + "q." + RESET + " - Завершить ввод на данном
                       уровне");
    System.out.println();
    System.out.print("Ввод: ");

    input = scanner.nextLine();
    System.out.println();

    switch (input){
        case ("1"): {
            System.out.print("Введите имя узла: ");
            name = scanner.nextLine();

            graphRoot = new ClassNode(name);
            nodeQueue.offer(graphRoot);

            //Когда закончили ввод, то устанавливает флаг - выход
            input = "Выход";
            break;
        }
        case ("q"): {
            //Если мы вышли на данном шаге, то graphRoot = null
            //поэтому сразу возращаем null
            return null;
        }
        default: {
            System.out.println(RED + "Ошибка ввода..." + RESET);
            break;
        }
    }
}
while (!input.equals("Выход"));
System.out.println();

//Цикл do while
//пока очередь не опустеет
//делаем "шаги автомата"
do {
    printSeparator();
    tempNode = nodeQueue.poll();    //"Вытаскиваем" узел из очереди

    assert tempNode != null;        //Проверка на null

    printNodeInfo(tempNode);        //Вывод информации о "вытащенном" узле

    //Если "вытащенный" узел имеет тип ClassNode
    //то предлагается создать дочерний подкласс или индивид
    if (tempNode instanceof ClassNode)
    {
        //Механизм do while предалагает пользователю повторный ввод
        //при неверных введенных данных
        do {
            System.out.println("1. - Создать узел типа Подкласс(ClassNode)");
            System.out.println("2. - Создать узел типа Индивид
                               (IndividualNode)");
            System.out.println(RED + "q." + RESET + " - Завершить ввод на
                               данном уровне");
            System.out.println();
            System.out.print("Ввод: ");
            input = scanner.nextLine();
            System.out.println();

            switch (input) {
                //Создание нового подкласса
                case ("1"): {
```

```
                    System.out.print("Введите имя узла: ");
                    name = scanner.nextLine();

                    ClassNode node = new ClassNode(name);

                    //Приводим общий узел к типу ClassNode и вызываем
                    //метод addChild
                    ((ClassNode) tempNode).addChild(node);

                    //Добавляем новый созданный узел в очередь
                    nodeQueue.offer(node);
                    break;
                }
                //Создание нового индивида
                case ("2"): {
                    System.out.print("Введите имя индивида: ");
                    name = scanner.nextLine();

                    IndividualNode node = new IndividualNode(name);

                    //Приводим общий узел к типу ClassNode и вызываем
                    //метод addChild
                    ((ClassNode) tempNode).addChild(node);

                    //Добавляем новый созданный узел в очередь
                    nodeQueue.offer(node);
                    break;
                }
                case ("q"): {
                    break;
                }
                default: {
                    System.out.println(RED + "Ошибка ввода..." + RESET);
                    break;
                }
            }
        }
        while (!input.equals("q"));
    }
    System.out.println();

    //Если "вытащенный" узел имеет тип IndividualNode
    //то предлагается добавить атрибут для данного индивида
    if (tempNode instanceof IndividualNode)
    {
        //Механизм do while предалагает пользователю повторный ввод
        //при неверных введенных данных
        do {
            System.out.println("1. - Создать узел типа Атрибут
                                (AttributeNode)");
            System.out.println(RED + "q." + RESET + " - Завершить ввод на
                                данном уровне");
            System.out.println();
            System.out.print("Ввод: ");

            input = scanner.nextLine();
            System.out.println();

            switch (input) {
                //Создание нового атрибута
                case ("1"): {
                    String data;

                    System.out.print("Введите имя атрибута: ");
                    name = scanner.nextLine();
                    System.out.print("Введите значение атрибута: ");
                    data = scanner.nextLine();
```

```
                                //Приводим общий узел к типу IndividualNode и вызываем
                                //метод addAttribute
                                ((IndividualNode) tempNode).addAttribute(name, data);

                                //Поскольку узла типа Атрибут и значение конечные,
                                //то их уже не добавляем в очередь
                                break;
                            }
                            case ("q"): {
                                break;
                            }
                            default: {
                                System.out.println(RED + "Ошибка ввода..." + RESET);
                                break;
                            }
                        }
                    }
                    while (!input.equals("q"));
                }
            }
            while (!nodeQueue.isEmpty());
            return graphRoot;
        }
    }
```

Приложение 2

Результаты тестирования

```
--------------------------------------------------------------------------------
                              Создание нового узла
--------------------------------------------------------------------------------
1. - Создать узел типа Класс(ClassNode)
q. - Завершить ввод на данном уровне

Ввод: 1

Введите имя узла: Звери

--------------------------------------------------------------------------------
                              Создание нового узла
--------------------------------------------------------------------------------
Родитель: Звери | Статус родителя: Класс
--------------------------------------------------------------------------------
1. - Создать узел типа Подкласс(ClassNode)
2. - Создать узел типа Индивид (IndividualNode)
q. - Завершить ввод на данном уровне

Ввод: 1

Введите имя узла: Хищники
1. - Создать узел типа Подкласс(ClassNode)
2. - Создать узел типа Индивид (IndividualNode)
q. - Завершить ввод на данном уровне

Ввод: 1

Введите имя узла: Травоядные
1. - Создать узел типа Подкласс(ClassNode)
2. - Создать узел типа Индивид (IndividualNode)
q. - Завершить ввод на данном уровне

Ввод: 2

Введите имя индивида: Крот
1. - Создать узел типа Подкласс(ClassNode)
2. - Создать узел типа Индивид (IndividualNode)
q. - Завершить ввод на данном уровне
```

```
Ввод: q


--------------------------------------------------------------------------------
                              Создание нового узла
--------------------------------------------------------------------------------
Родитель: Хищники | Статус родителя: Подкласс
--------------------------------------------------------------------------------
1. - Создать узел типа Подкласс(ClassNode)
2. - Создать узел типа Индивид (IndividualNode)
q. - Завершить ввод на данном уровне

Ввод: 2
```

```
Введите имя индивида: Лев
1. - Создать узел типа Подкласс(ClassNode)
2. - Создать узел типа Индивид (IndividualNode)
q. - Завершить ввод на данном уровне

Ввод: Медведь

Ошибка ввода...
1. - Создать узел типа Подкласс(ClassNode)
2. - Создать узел типа Индивид (IndividualNode)
q. - Завершить ввод на данном уровне

Ввод: 2

Введите имя индивида: Медведь
1. - Создать узел типа Подкласс(ClassNode)
2. - Создать узел типа Индивид (IndividualNode)
q. - Завершить ввод на данном уровне

Ввод: q


--------------------------------------------------------------------------------------------------------
                                    Создание нового узла
--------------------------------------------------------------------------------------------------------
Родитель: Травоядные | Статус родителя: Подкласс
--------------------------------------------------------------------------------------------------------
1. - Создать узел типа Подкласс(ClassNode)
2. - Создать узел типа Индивид (IndividualNode)
q. - Завершить ввод на данном уровне

Ввод: 2

Введите имя индивида: Коза
1. - Создать узел типа Подкласс(ClassNode)
2. - Создать узел типа Индивид (IndividualNode)
q. - Завершить ввод на данном уровне

Ввод: 2
```

```
Введите имя индивида: Зебра
1. - Создать узел типа Подкласс(ClassNode)
2. - Создать узел типа Индивид (IndividualNode)
q. - Завершить ввод на данном уровне

Ввод: q


--------------------------------------------------------------------------------------------------------
                                    Создание нового узла
--------------------------------------------------------------------------------------------------------
Родитель: Крот
--------------------------------------------------------------------------------------------------------

1. - Создать узел типа Атрибут(AttributeNode)
q. - Завершить ввод на данном уровне

Ввод: q
```

```
--------------------------------------------------------------------------------
                            Создание нового узла
--------------------------------------------------------------------------------
Родитель: Лев
--------------------------------------------------------------------------------

1. - Создать узел типа Атрибут(AttributeNode)
q. - Завершить ввод на данном уровне

Ввод: 1

Введите имя атрибута: возраст
Введите значение атрибута: 5 лет
1. - Создать узел типа Атрибут(AttributeNode)
q. - Завершить ввод на данном уровне

Ввод: 1

Введите имя атрибута: Длина
Введите значение атрибута: 2 метра
1. - Создать узел типа Атрибут(AttributeNode)
q. - Завершить ввод на данном уровне

Ввод: q

--------------------------------------------------------------------------------
                            Создание нового узла
--------------------------------------------------------------------------------
Родитель: Медведь
--------------------------------------------------------------------------------

1. - Создать узел типа Атрибут(AttributeNode)
q. - Завершить ввод на данном уровне

Ввод: 1

Введите имя атрибута: окрас
Введите значение атрибута: бурый
1. - Создать узел типа Атрибут(AttributeNode)
q. - Завершить ввод на данном уровне
```

```
Ввод: q

--------------------------------------------------------------------------------
                            Создание нового узла
--------------------------------------------------------------------------------
Родитель: Коза
--------------------------------------------------------------------------------

1. - Создать узел типа Атрибут(AttributeNode)
q. - Завершить ввод на данном уровне
```

```
Ввод: q

------------------------------------------------------------------------------------------------------
                                    Создание нового узла
------------------------------------------------------------------------------------------------------
Родитель: Зебра
------------------------------------------------------------------------------------------------------

1. - Создать узел типа Атрибут(AttributeNode)
q. - Завершить ввод на данном уровне

Ввод: 1
```

```
Введите имя атрибута: окрас
Введите значение атрибута: белая в чёрную полоску
1. - Создать узел типа Атрибут(AttributeNode)
q. - Завершить ввод на данном уровне

Ввод: q

        Представление графовой структуры в виде списков смежностей:

Обозначения: узел дерева, подкласс, индивид.

Звери: Хищники, Травоядные, Крот
Хищники: Лев, Медведь
Травоядные: Коза, Зебра
Крот: ID: 1
Лев: ID: 1, возраст: 5 лет, Длина : 2 метра
Медведь: ID: 2, окрас: бурый
Коза: ID: 1
Зебра: ID: 2, окрас: белая в чёрную полоску

                Конец работы...

Process finished with exit code 0
```