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 №2

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. ContainerNode -абстрактные класс - узел компонент, который может содержать потомков
2. Leaf - конечный узел.

В ходе работы было реализовано 3 вида узлов - контейнеров, унаследованных от ContainerNode: ClassNode - может являться корнем, классом или подклассом графа, IndividualNode узел-индивид, AttributeNode- содержит название одного атрибута. Так же был реализован один конечный узел ValueNode - содержит одно значение атрибута.

Для связи между узлами созданы 2 вида связей: DataProperty (связи: IndividNode → AttributeNode, AttributeNode → ValueNode) и ObjectProperty (ClassNode → ClassNode, ClassNode → IndividNode).

Для создания графовой структуры реализован паттерн Строитель. От основного интерфейса Builder, унаследован GraphBuilder, который реализует методы необходимые для создания графа. Для ввода ограничений и упорядочивания шагов создания графа, над строителем реализован класс Director.

Классы Director и GraphBuilder являются бинами. Это позволяет создать экземпляры данных классов в момент сборки приложения. Класс GraphBuilder внедряется в главный класс App при помощи DI (Depedency Injection), что позволяет обеспечить слабосвязанность приложения и сделать данные классы Singleton'ами без изменения кода.

Использование классов GraphBuilder и Director позволяет использовать Spring AOP над их методами, без прямого взаимодействия с библиотекой AspectJ. Применение аспектов для данных классов используется для логирования приложения при помощи библиотеки Logf4.

Для вывода графа в консоль в удобном для прочтения виде, создан интерфейс Printer и унаследованный от него класс GraphPrinter, который так же является бином и singleton'ом.

……………………………………………………………………………

На рис 1. приведена диаграмма классов.

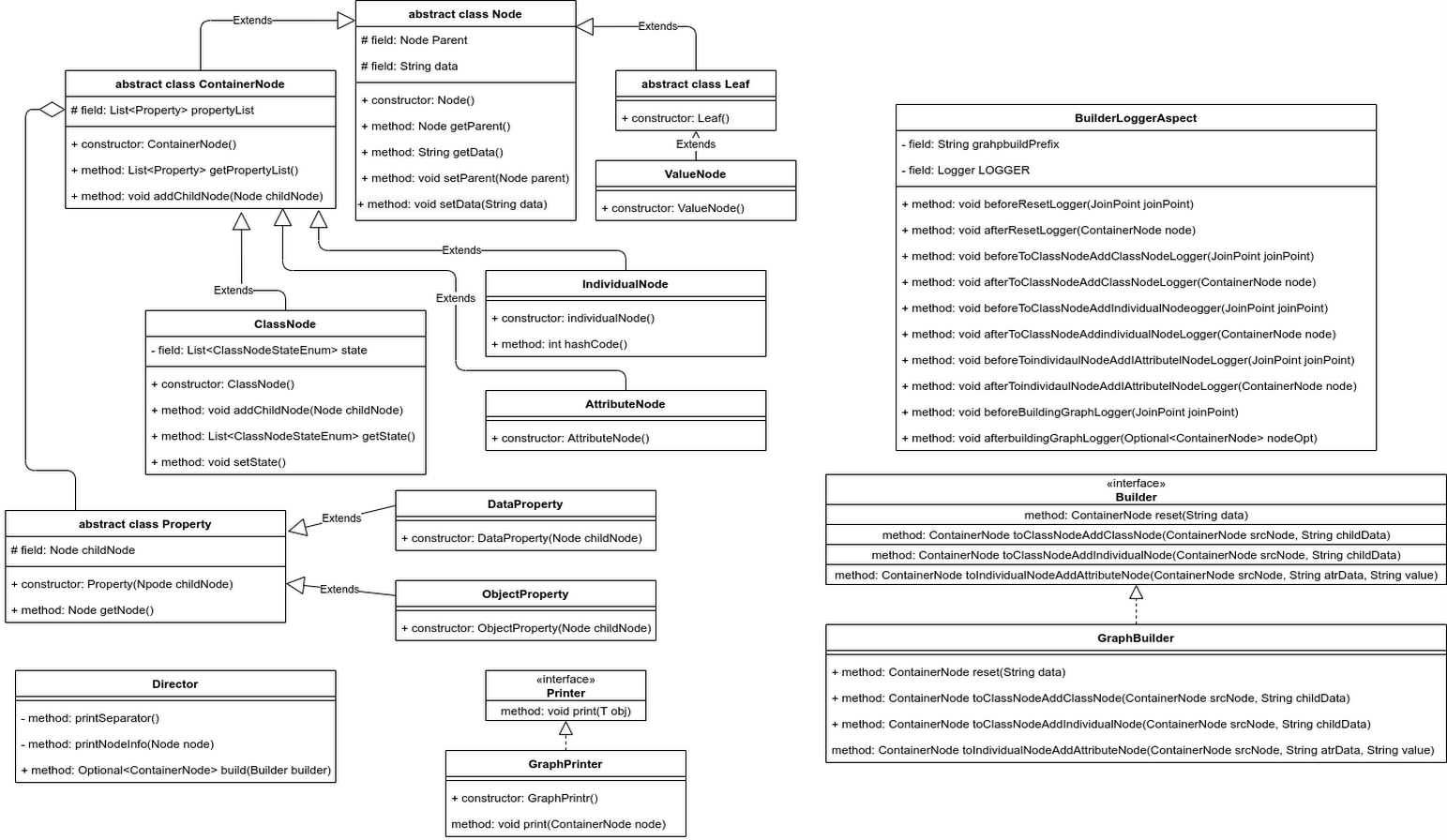


Рис. 1 – Диаграмма классов

# Вывод

Использование Spring позволяет облегчить разработку приложений на Java. В приложении, основанном на данном фреймворке объекты являются слабосвязанными за счёт использования Depedency Injection. Так же использование Spring позволяет не беспокоится программисту о связывании объектов, потому что это производится автоматически.

Кроме того, при использовании Spring'a возможно программирование в декларативном стиле с помощью аннотаций, что значительно уменьшает объем кода.

# Приложение 1

**Программный код**

**Director.java**

**package** com.ngtu.sdp.laboratory\_work2.builder;

**import** com.ngtu.sdp.laboratory\_work2.nodes.ClassNode;

**import** com.ngtu.sdp.laboratory\_work2.nodes.ContainerNode;

**import** com.ngtu.sdp.laboratory\_work2.nodes.IndividualNode;

**import** com.ngtu.sdp.laboratory\_work2.nodes.Node;

**import** org.springframework.context.annotation.Scope;

**import** org.springframework.stereotype.Component;

**import** java.util.\*;

/\*\*

\* Класс, управляющий Builder'ом

\*

\* **@see** Builder

\* **@see** GraphBuilder

\* \*/

@Component("director")

@Scope("singleton")

**public** **class** Director {

//Константы для хранения последовательностей для

//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";

/\*\*

\* Конструктор по умолчанию.

\* \*/

**public** Director() {

}

/\*\*

\* 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 " + ***GREEN*** + "Создание нового узла" + ***RESET***);

System.***out***.println("----------------------------------------------------");

/\*\*

\* Метод выводящий информацию о переданном узле.

\* Скрыт, т.к. не использутеся напрямую

\*

\* **@param** node - узел, ин-ию о котором необходимо вывести.

\* \*/

**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).getStateAsString());

}

System.***out***.println();

System.***out***.println("------------------------------------------------------");

}

/\*\*

\* Метод сборки графа

\*

\* **@return** оболочку Optional с родительским узлом

\* **@param** builder - экземпляр билдера при помощи которого будет производится

\* создание структуры.

\* \*/

**public** Optional<ContainerNode> build(Builder builder)

{

//Создаем очередь для хранения узлов

Queue<ContainerNode> nodeQueue = **null**;

ContainerNode outputNode;

ContainerNode inputNode;

ContainerNode root = **null**;

//Открытие потока ввода

Scanner scanner = **new** Scanner(System.***in***);

String input; //Временные ссылки для хранения

String data; //введенных строк

//Создание 1ого узла - корня графовой структуры

//Механизм do while предалагает пользователю повторный ввод

//при неверных введенных данных

**do** {

*printSeparator*();

System.***out***.println("1. - Создать корень дерева");

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"): {

nodeQueue = **new** ArrayDeque<>();

data = "";

System.***out***.print("Введите имя узла: ");

**while** (data.isBlank())

{

data = scanner.nextLine();

}

root = builder.reset(data);

nodeQueue.offer(root);

//Когда закончили ввод, то устанавливает флаг - выход

input = "q";

**break**;

}

**case** ("q"):

{

//Если мы вышли на данном шаге, то graphRoot = null

//поэтому сразу возращаем null

**return** Optional.*empty*();

}

**default**:

{

System.***out***.println(***RED*** + "Ошибка ввода..." + ***RESET***);

**break**;

}

}

}

**while** (!input.equals("q"));

System.***out***.println();

//Цикл do while

//пока очередь не опустеет

//делаем "шаги автомата"

**do**

{

*printSeparator*();

outputNode = nodeQueue.poll(); //"Вытаскиваем" узел из очереди

**assert** outputNode != **null**; //Проверка на null

*printNodeInfo*(outputNode); //Вывод информации о "вытащенном" узле

//Если "вытащенный" узел имеет тип ClassNode

//то предлагается создать дочерний подкласс или индивид

**if** (outputNode **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 (IndividNode)");

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"):

{

data = "";

System.***out***.print("Введите имя узла: ");

**while** (data.isBlank())

{

data = scanner.nextLine();

}

inputNode = builder.toClassNodeAddClassNode(

outputNode, data);

nodeQueue.offer(inputNode);

**break**;

}

//Создание нового индивида

**case** ("2"):

{

System.***out***.print("Введите имя индивида: ");

data = scanner.nextLine();

inputNode = builder.toClassNodeAddIndividualNode

(outputNode, data);

//Добавляем новый созданный узел в очередь

nodeQueue.offer(inputNode);

**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** (outputNode **instanceof** IndividualNode)

{

//Механизм do while предалагает пользователю повторный ввод

//при неверных введенных данных

**do** {

System.***out***.println("1.-Создать узел - 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"):

{

data = "";

System.***out***.print("Введите имя атрибута: ");

String name = scanner.nextLine();

System.***out***.print("Введите значение атрибута: ");

**while** (data.isBlank())

{

data = scanner.nextLine();

}

builder.toIndividualNodeAddAttributeNode(

outputNode, 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());

**assert** root != **null**;

**return** Optional.*of*(root);

}

}

GraphBuilder.java

**package** com.ngtu.sdp.laboratory\_work2.builder;

**import** com.ngtu.sdp.laboratory\_work2.nodes.\*;

**import** org.springframework.context.annotation.Scope;

**import** org.springframework.stereotype.Component;

/\*\*

\* Частная релизация bilder'a

\* Bilder для графа.

\*

\* **@see** Builder

\* **@see** Director

\* \*/

@Component("graphBuilder")

@Scope("prototype")

**public** **class** GraphBuilder **implements** Builder

{

/\*\*

\* Сброс строителя, посторение нового графа.

\*

\* **@param** data - данные корня графа.

\* **@return** полученный узел

\* \*/

@Override

**public** ContainerNode reset(String data)

{

//Задаем корню состояние "Класс"

**return** **new** ClassNode(data, ClassNodeStateEnum.CLASS);

}

/\*\*

\* Метод, для добавления к узлу ClassNode потомка ClassNode

\*

\* **@param** childNodeData - данные дочернего узла

\* **@param** srcNode - узел родитель, к которому необходимо добавить потомка.

\* **@return** полученный узел

\* \*/

@Override

**public** ContainerNode toClassNodeAddClassNode(ContainerNode srcNode,

String childNodeData)

{

//Создание нового дочернего узла

ClassNode childNode = **new** ClassNode(srcNode, childNodeData,

ClassNodeStateEnum.SUBCLASS);

//К род. узлу добавляем новый узел

srcNode.addChildNode(childNode);

//Если родиетльский узел еще не имеет состояния "ИМЕЕТ ПОДКЛАСС", то задаем ему это состояние

**if** (!((ClassNode)srcNode).getState().contains(ClassNodeStateEnum.HAVE\_SUBCLASS))

{

((ClassNode)srcNode).addState(ClassNodeStateEnum.HAVE\_SUBCLASS);

}

**return** childNode;

}

/\*\*

\* Метод, для добавления к узлу ClassNode потомка IndividualNode

\*

\* **@param** childNodeData - данные дочернего узла

\* **@param** srcNode - узел родитель, к которому необходимо добавить потомка.

\* **@return** полученный узел

\* \*/

@Override

**public** ContainerNode toClassNodeAddIndividualNode(ContainerNode srcNode,

String childNodeData)

{

//Создание нового дочернего узла

IndividualNode childNode = **new** IndividualNode(srcNode, childNodeData);

//К род. узлу добавляем новый узел

srcNode.addChildNode(childNode);

//Если родительский узел еще не имеет состояния "ИМЕЕТ ИНДИВИДА", то задаем ему это состояние

**if** (!((ClassNode)srcNode).getState().contains(ClassNodeStateEnum.HAVE\_INDIVIDUAL))

{

((ClassNode)srcNode).addState(ClassNodeStateEnum.HAVE\_INDIVIDUAL);

}

//Автоматическое задание Атрибута ID

childNode.setID();

**return** childNode;

}

/\*\*

\* Метод, для добавления к узлу IndividualNode потомка AttributeNode с параметром

\*

\* **@param** value - значение атрибута

\* **@param** atrNodeData - данные дочернего узла

\* **@param** srcNode - узел родитель, к которому необходимо добавить потомка.

\* **@return** полученный узел

\* \*/

@Override

**public** ContainerNode toIndividualNodeAddAttributeNode(ContainerNode srcNode,

String atrNodeData, String value)

{

//Создание нового узла значения

ValueNode valueNode = **new** ValueNode(value);

//Создание нового узла атрибута

AttributeNode atrNode = **new** AttributeNode(srcNode, atrNodeData);

//К атрибуту прибавляем значение

atrNode.addChildNode(valueNode);

//Значению записываем родителя атрибута

valueNode.setParent(atrNode);

//К род. узлу добавляем новый узел

srcNode.addChildNode(atrNode);

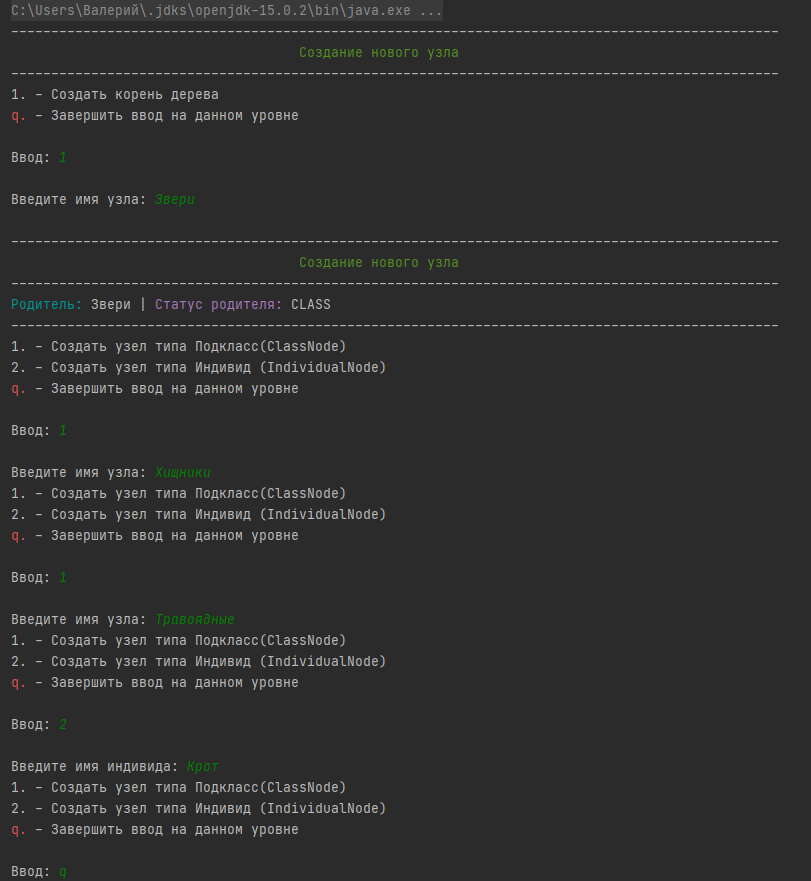
**return** atrNode;

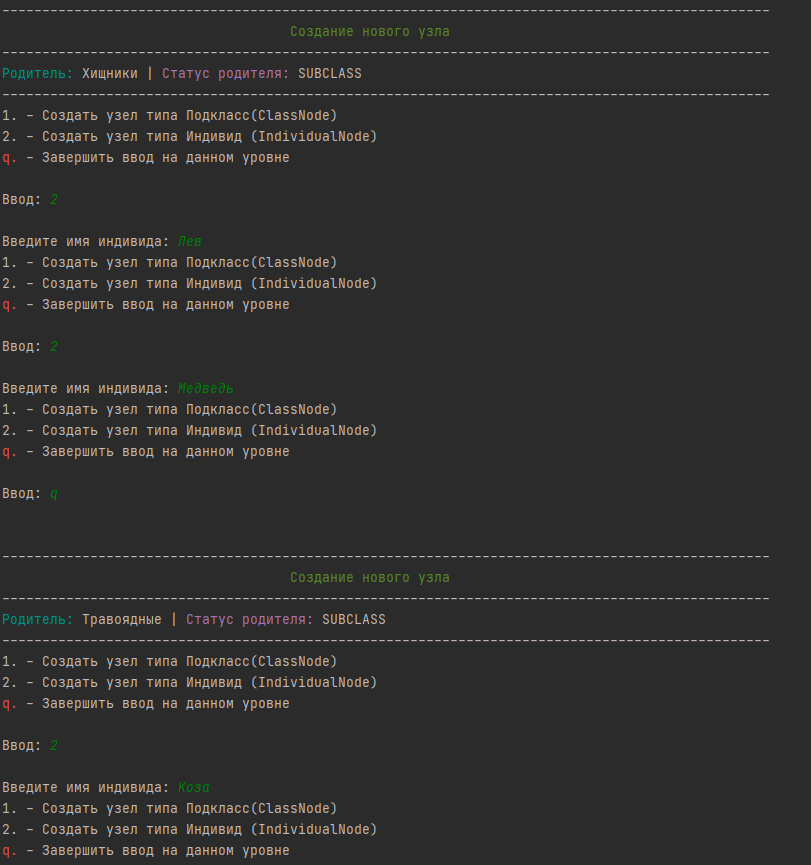
}

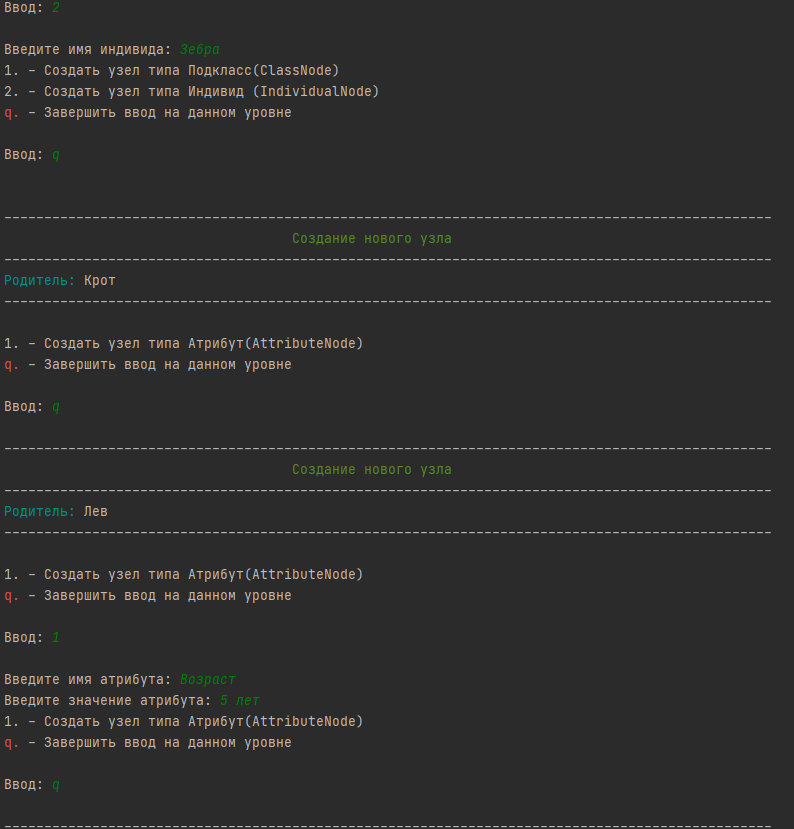
}

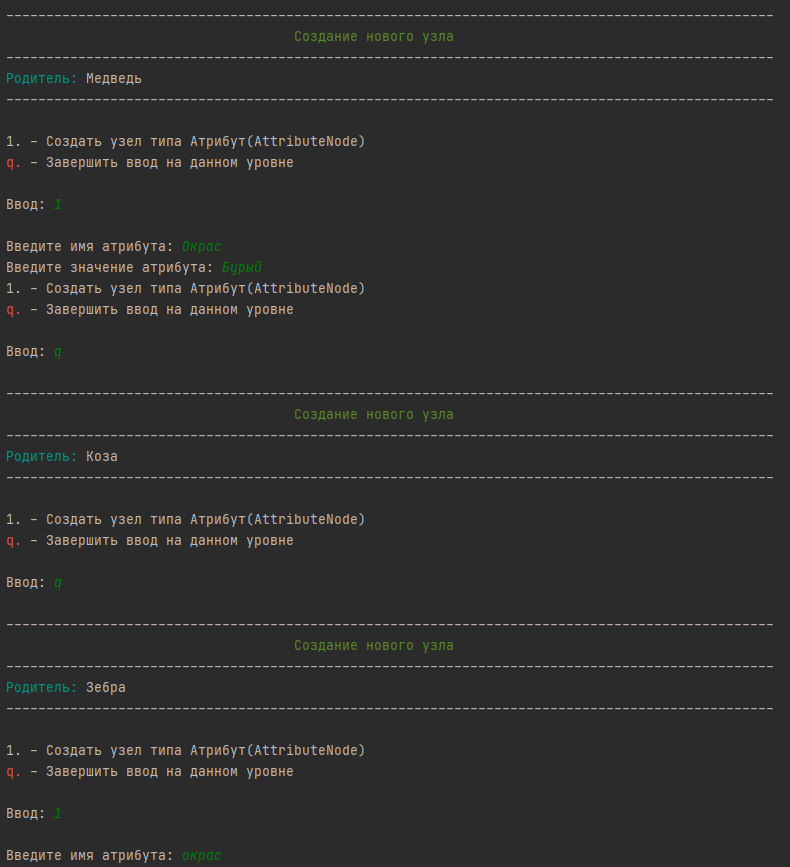
# Приложение 2

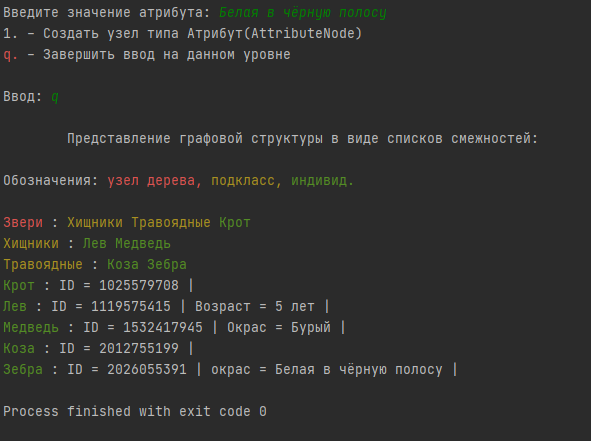
**Результаты тестирования**











# Приложение 3

**Файл логирования приложения**

2021-05-05 19:27:46.273 [main] DEBUG *org.springframework.context.support.ClassPathXmlApplicationContext* - Refreshing *org.springframework.context.support.ClassPathXmlApplicationContext*@43dac38f  
2021-05-05 19:27:46.438 [main] DEBUG *org.springframework.context.annotation.ClassPathBeanDefinitionScanner* - Identified candidate component class: file [C:\Users\Валерий\Desktop\Учёба\Software-design-patterns-*2*-course-*2*-semestr\Laboratory\_work2\target\classes\com\ngtu\sdp\laboratory\_work2\App.class]  
2021-05-05 19:27:46.442 [main] DEBUG *org.springframework.context.annotation.ClassPathBeanDefinitionScanner* - Identified candidate component class: file [C:\Users\Валерий\Desktop\Учёба\Software-design-patterns-*2*-course-*2*-semestr\Laboratory\_work2\target\classes\com\ngtu\sdp\laboratory\_work2\aspects\BuilderLoggerAspect.class]  
2021-05-05 19:27:46.444 [main] DEBUG *org.springframework.context.annotation.ClassPathBeanDefinitionScanner* - Identified candidate component class: file [C:\Users\Валерий\Desktop\Учёба\Software-design-patterns-*2*-course-*2*-semestr\Laboratory\_work2\target\classes\com\ngtu\sdp\laboratory\_work2\builder\Director.class]  
2021-05-05 19:27:46.445 [main] DEBUG *org.springframework.context.annotation.ClassPathBeanDefinitionScanner* - Identified candidate component class: file [C:\Users\Валерий\Desktop\Учёба\Software-design-patterns-*2*-course-*2*-semestr\Laboratory\_work2\target\classes\com\ngtu\sdp\laboratory\_work2\builder\GraphBuilder.class]  
2021-05-05 19:27:46.447 [main] DEBUG *org.springframework.context.annotation.ClassPathBeanDefinitionScanner* - Identified candidate component class: file [C:\Users\Валерий\Desktop\Учёба\Software-design-patterns-*2*-course-*2*-semestr\Laboratory\_work2\target\classes\com\ngtu\sdp\laboratory\_work2\printers\GraphPrinter.class]  
2021-05-05 19:27:46.470 [main] DEBUG *org.springframework.beans.factory.xml.XmlBeanDefinitionReader* - Loaded *11* bean definitions from class path resource [*applicationContext.xml*]  
2021-05-05 19:27:46.486 [main] DEBUG *org.springframework.beans.factory.support.DefaultListableBeanFactory* - Creating shared instance of singleton bean 'org.springframework.context.annotation.internalConfigurationAnnotationProcessor'  
2021-05-05 19:27:46.515 [main] DEBUG *org.springframework.beans.factory.support.DefaultListableBeanFactory* - Creating shared instance of singleton bean 'org.springframework.context.event.internalEventListenerProcessor'  
2021-05-05 19:27:46.516 [main] DEBUG *org.springframework.beans.factory.support.DefaultListableBeanFactory* - Creating shared instance of singleton bean 'org.springframework.context.event.internalEventListenerFactory'  
2021-05-05 19:27:46.518 [main] DEBUG *org.springframework.beans.factory.support.DefaultListableBeanFactory* - Creating shared instance of singleton bean 'org.springframework.context.annotation.internalAutowiredAnnotationProcessor'  
2021-05-05 19:27:46.518 [main] DEBUG *org.springframework.beans.factory.support.DefaultListableBeanFactory* - Creating shared instance of singleton bean 'org.springframework.context.annotation.internalCommonAnnotationProcessor'  
2021-05-05 19:27:46.522 [main] DEBUG *org.springframework.beans.factory.support.DefaultListableBeanFactory* - Creating shared instance of singleton bean 'org.springframework.aop.config.internalAutoProxyCreator'  
2021-05-05 19:27:46.576 [main] DEBUG *org.springframework.beans.factory.support.DefaultListableBeanFactory* - Creating shared instance of singleton bean 'app'  
2021-05-05 19:27:46.601 [main] DEBUG *org.springframework.aop.aspectj.annotation.ReflectiveAspectJAdvisorFactory* - Found AspectJ method: public void *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect.beforeBuildingGraphLogger*(*org.aspectj.lang.JoinPoint*)  
2021-05-05 19:27:46.603 [main] DEBUG *org.springframework.aop.aspectj.annotation.ReflectiveAspectJAdvisorFactory* - Found AspectJ method: public void *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect.beforeResetLogger*(*org.aspectj.lang.JoinPoint*)  
2021-05-05 19:27:46.603 [main] DEBUG *org.springframework.aop.aspectj.annotation.ReflectiveAspectJAdvisorFactory* - Found AspectJ method: public void *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect.beforeToClassNodeAddClassNodeLogger*(*org.aspectj.lang.JoinPoint*)  
2021-05-05 19:27:46.603 [main] DEBUG *org.springframework.aop.aspectj.annotation.ReflectiveAspectJAdvisorFactory* - Found AspectJ method: public void *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect.beforeToClassNodeAddIndividualNodeLogger*(*org.aspectj.lang.JoinPoint*)  
2021-05-05 19:27:46.604 [main] DEBUG *org.springframework.aop.aspectj.annotation.ReflectiveAspectJAdvisorFactory* - Found AspectJ method: public void *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect.beforeToIndividualNodeAddAttributeNodeLogger*(*org.aspectj.lang.JoinPoint*)  
2021-05-05 19:27:46.604 [main] DEBUG *org.springframework.aop.aspectj.annotation.ReflectiveAspectJAdvisorFactory* - Found AspectJ method: public void *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect.afterReturningBuildingGraphLogger*(*java.util.Optional*)  
2021-05-05 19:27:46.612 [main] DEBUG *org.springframework.aop.aspectj.annotation.ReflectiveAspectJAdvisorFactory* - Found AspectJ method: public void *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect.afterReturningResetLogger*(*com.ngtu.sdp.laboratory\_work2.nodes.ContainerNode*)  
2021-05-05 19:27:46.613 [main] DEBUG *org.springframework.aop.aspectj.annotation.ReflectiveAspectJAdvisorFactory* - Found AspectJ method: public void *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect.afterReturningToClassNodeAddClassNodeLogger*(*com.ngtu.sdp.laboratory\_work2.nodes.ContainerNode*)  
2021-05-05 19:27:46.615 [main] DEBUG *org.springframework.aop.aspectj.annotation.ReflectiveAspectJAdvisorFactory* - Found AspectJ method: public void *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect.afterReturningToClassNodeAddIndividualNodeLogger*(*com.ngtu.sdp.laboratory\_work2.nodes.ContainerNode*)  
2021-05-05 19:27:46.616 [main] DEBUG *org.springframework.aop.aspectj.annotation.ReflectiveAspectJAdvisorFactory* - Found AspectJ method: public void *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect.afterReturningToIndividualNodeAddAttributeNodeLogger*(*com.ngtu.sdp.laboratory\_work2.nodes.ContainerNode*)  
2021-05-05 19:27:46.630 [main] DEBUG *org.springframework.beans.factory.support.DefaultListableBeanFactory* - Creating shared instance of singleton bean 'printer'  
2021-05-05 19:27:46.720 [main] DEBUG *org.springframework.beans.factory.support.DefaultListableBeanFactory* - Creating shared instance of singleton bean 'builderLoggerAspect'  
2021-05-05 19:27:46.731 [main] DEBUG *org.springframework.beans.factory.support.DefaultListableBeanFactory* - Creating shared instance of singleton bean 'director'  
2021-05-05 19:27:46.800 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: start graph building.  
2021-05-05 19:27:53.199 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: building the root of the graph (class) - Звери.  
2021-05-05 19:27:53.200 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: root of the graph (class) - Звери - built success!  
2021-05-05 19:27:58.816 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: building subclass - Хищники.  
2021-05-05 19:27:58.817 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: subclass - Хищники - built success!  
2021-05-05 19:28:10.504 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: building subclass - Травоядные.  
2021-05-05 19:28:10.504 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: subclass - Травоядные - built success!  
2021-05-05 19:28:13.694 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: building individual - Крот.  
2021-05-05 19:28:13.697 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: individual - Крот - built success!  
2021-05-05 19:28:22.479 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: building individual - Лев.  
2021-05-05 19:28:22.479 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: individual - Лев - built success!  
2021-05-05 19:28:28.707 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: building individual - Медведь.  
2021-05-05 19:28:28.707 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: individual - Медведь - built success!  
2021-05-05 19:28:41.940 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: building individual - Коза.  
2021-05-05 19:28:41.940 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: individual - Коза - built success!  
2021-05-05 19:29:00.160 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: building individual - Зебра.  
2021-05-05 19:29:00.160 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: individual - Зебра - built success!  
2021-05-05 19:29:15.962 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: building attribute - Возраст with value = *5* лет.  
2021-05-05 19:29:15.962 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: attribute - Возраст with value = *5* лет - built success!  
2021-05-05 19:29:43.859 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: building attribute - Окрас with value = Бурый.  
2021-05-05 19:29:43.860 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: attribute - Окрас with value = Бурый - built success!  
2021-05-05 19:30:22.230 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: building attribute - окрас with value = Белая в чёрную полосу.  
2021-05-05 19:30:22.230 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: attribute - окрас with value = Белая в чёрную полосу - built success!  
2021-05-05 19:30:24.463 [main] DEBUG *com.ngtu.sdp.laboratory\_work2.aspects.BuilderLoggerAspect* - Building a graph structure: graph build success.  
2021-05-05 19:30:24.465 [main] DEBUG *org.springframework.context.support.ClassPathXmlApplicationContext* - Closing *org.springframework.context.support.ClassPathXmlApplicationContext*@43dac38f, started on Wed May *05* 19:27:46 MSK *2021*