ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Санкт – Петербургский государственный университет телекоммуникаций

им. проф. М.А. Бонч-Бруевича»

КАФЕДРА ИНФОРМАЦИОННЫХ УПРАВЛЯЮЩИХ СИСТЕМ

**Лабораторно-практическая работа № 01**

по дисциплине

**«СИСТЕМЫ ХРАНЕНИЯ ДАННЫХ»**

Студент гр. 831 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Пономарев Е.И.

(подпись)

Проверил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Параничев А.В.

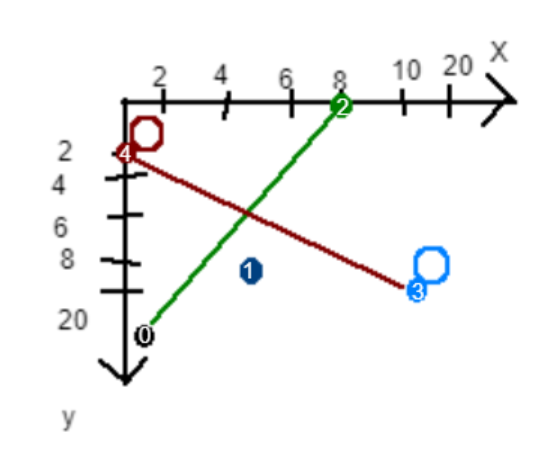
(оценка и подпись)

Санкт-Петербург

2021 год

**Лабораторно-практическая работа № 02: Представление сети в виде набора «фабрик» с помощью шабонов Фабричный Метод и Прототип**

Выполним представление сети в виде набора «фабрик», используя онлайн-компилятор <https://www.onlinegdb.com/> для следующего кода на языке Java для варианта 83114 (рис. 1)



*Рис.1. Представление графа сети в результате выполнения задания Task\_3*

Для этого составим исходный код:

import java.sql.\*;

import java.util.\*;

import java.lang.\*;

public class Main

{

public static final int SINGLE\_SWITCH\_FABRIC = 1;

public static final int CASCADE\_FABRIC = 2;

public static void main(String[] args)

{

System.out.println("1. Создание сети \"Фабричным методом\":");

// создаем "Фабрику" для "произодства" объектов

DataFactory factory = new DataFactory();

List<aData> list\_items = new ArrayList<aData> ();

aData item;

item = factory.getByParams(SINGLE\_SWITCH\_FABRIC, 1, "blue");

list\_items.add(item);

item = factory.getByParams(CASCADE\_FABRIC, 4, "red", 3, "red");

list\_items.add(item);

item = factory.getByParams(CASCADE\_FABRIC, 0, "black", 2, "green");

list\_items.add(item);

for(int i = 0; i < list\_items.size(); i++)

{

list\_items.get(i).showEdges();

}

System.out.println("2. Создание сети с помощью шаблона Прототип:");

// создаем Менеджер прототипов с прототипами (можно добавить в run-time)

DataPrototypes prototypes = new DataPrototypes();

list\_items.clear();

// пример добавления и отображения 2 сетей с помощью Прототипов

String[] strs\_1 = {"1", "blue"};

item = prototypes.get\_SingleSwitchNetwork\_NoLoop(strs\_1);

list\_items.add(item);

String[] strs\_2 = {"4", "3", "red"};

item = prototypes.get\_CascadeNetwork\_Double(strs\_2);

list\_items.add(item);

String[] strs\_3 = {"2", "0", "green"};

item = prototypes.get\_CascadeNetwork\_Double(strs\_3);

list\_items.add(item);

for(int i = 0; i < list\_items.size(); i++)

{

list\_items.get(i).showEdges();

}

}

}

abstract class aData // абстрактный класс (интерфейс) с данными для отображения

{

protected String \_str\_name = "noname"; // перекрываемые (override) данные

public abstract void showEdges(); // перекрываемый метод извлечения всех ветвей

}

class DataFactory // класс "Фабрики"

{

public aData getByParams(Integer index\_fabric,

Integer node\_num\_1, String node\_color\_1

)

{ // для числа параметров, равного 3, остальные default

return getByParams(index\_fabric, node\_num\_1, node\_color\_1,

-1, "no\_color");

}

public aData getByParams(Integer index\_fabric,

Integer node\_num\_1, String node\_color\_1,

Integer node\_num\_2, String node\_color\_2

)

{ // для числа параметров, равного 5, остальные default

return getByParams(index\_fabric, node\_num\_1, node\_color\_1,

node\_num\_2, node\_color\_2, -1, "no\_color");

}

public aData getByParams(Integer index\_fabric,

Integer node\_num\_1, String node\_color\_1,

Integer node\_num\_2, String node\_color\_2,

Integer node\_num\_3, String node\_color\_3

)

{ // для числа параметров, равного 7, остальные default

return getByParams(index\_fabric, node\_num\_1, node\_color\_1,

node\_num\_2, node\_color\_2, node\_num\_3, node\_color\_3, -1, "no\_color");

}

public aData getByParams(Integer index\_fabric,

Integer node\_num\_1, String node\_color\_1,

Integer node\_num\_2, String node\_color\_2,

Integer node\_num\_3, String node\_color\_3,

Integer node\_num\_4, String node\_color\_4

)

{ // для всех 9 параметров (т.к. в данной сети не более 4 узлов)

aData retData;

// "switch-case" для инициализации одного из объектов

if (index\_fabric == Main.SINGLE\_SWITCH\_FABRIC)

{

retData = new Single\_Switch\_SimpleNetwork(node\_num\_1, node\_color\_1,

node\_num\_2, node\_color\_2);

}

else if (index\_fabric == Main.CASCADE\_FABRIC)

{

retData = new Cascade\_SimpleNetwork(node\_num\_1, node\_color\_1,

node\_num\_2, node\_color\_2, node\_num\_3, node\_color\_3,

node\_num\_4, node\_color\_4);

}

else // заглушка при неправильных данных

{

retData = new Empty("Unknown network, not recognized!(");

}

return retData;

}

}

class Empty extends aData // класс заглушки, управляемый "Фабрикой"

{

Empty(String str)

{

\_str\_name = str;

}

@Override

public void showEdges()

{

System.out.println("EmptyNetwork class: " + \_str\_name);

}

}

class Single\_Switch\_SimpleNetwork extends aData

{ // класс отображения фабрик из одного узла, управляемый "Фабрикой"

Single\_Switch\_SimpleNetwork(int node\_num\_1, String node\_color\_1)

{

this(node\_num\_1, node\_color\_1, -1, "no\_color");

}

Single\_Switch\_SimpleNetwork(int node\_num\_1, String node\_color\_1,

int node\_num\_2, String node\_color\_2)

{

if(node\_num\_2 == -1)

{

\_str\_name = "Ветвь без петли " +

node\_num\_1 + "-" + node\_num\_1 + ", цвет: " + node\_color\_1;

}

else

{

\_str\_name = "Ветвь в виде двойной петли " +

node\_num\_1 + "-" + node\_num\_1 + ", цвет: " + node\_color\_1 +

" и " + node\_color\_2;

}

}

@Override

public void showEdges()

{

System.out.println("\n###SingleSwitchNetwork class:\n" + \_str\_name);

}

}

class Cascade\_SimpleNetwork extends aData

{// класс отображения каскадных фабрик из 2,3 или 4 узлов; управляется "Фабрикой"

Cascade\_SimpleNetwork(int node\_num\_1, String node\_color\_1,

int node\_num\_2, String node\_color\_2)

{

this(node\_num\_1, node\_color\_1, node\_num\_2, node\_color\_2, -1, "no\_color");

}

Cascade\_SimpleNetwork(int node\_num\_1, String node\_color\_1,

int node\_num\_2, String node\_color\_2,

int node\_num\_3, String node\_color\_3)

{

this(node\_num\_1, node\_color\_1, node\_num\_2, node\_color\_2,

node\_num\_3, node\_color\_3, -1, "no\_color");

}

Cascade\_SimpleNetwork(int node\_num\_1, String node\_color\_1,

int node\_num\_2, String node\_color\_2,

int node\_num\_3, String node\_color\_3,

int node\_num\_4, String node\_color\_4)

{

\_str\_name = "Ветви:\n" + node\_num\_1 + "-" + node\_num\_2 + ", цвет: " +

node\_color\_2 + "\n";

if(node\_num\_3 != -1)

{

\_str\_name += node\_num\_1 + "-" + node\_num\_3 + ", цвет: " +

node\_color\_3 + "\n";

}

if(node\_num\_4 != -1)

{

\_str\_name += node\_num\_1 + "-" + node\_num\_4 + ", цвет: " +

node\_color\_4 + "\n";

}

}

@Override

public void showEdges()

{

System.out.println("\n###CascadeNetwork class:\n" + \_str\_name);

}

}

class Prototype extends aData // класс "типовой" (обобщенной) сети

{

public Prototype(String str\_name)

{

\_str\_name = str\_name;

}

@Override

public void showEdges()

{

System.out.println("\n###Prototype data: " + \_str\_name);

}

}

class DataPrototypes // класс, хранящий набор прототипов (и дополняемый прототипами)

{

public aData get\_SingleSwitchNetwork\_NoLoop(String[] list){

String str\_data = "SingleSwitchNetwork\_NoLoop: Ветвь без петли " +

list[0] + ", цвет: " + list[1];

return new Prototype(str\_data);

}

public aData get\_CascadeNetwork\_Double(String[] list){

String str\_data = "CascadeNetwork\_Double: Ветви:\n" +

list[0] + "-" + list[1] + ", цвет: " + list[2] + "; ";

return new Prototype(str\_data);

}

}

Результаты вывода на экран (https://onlinegdb.com/ByRaESRMu):

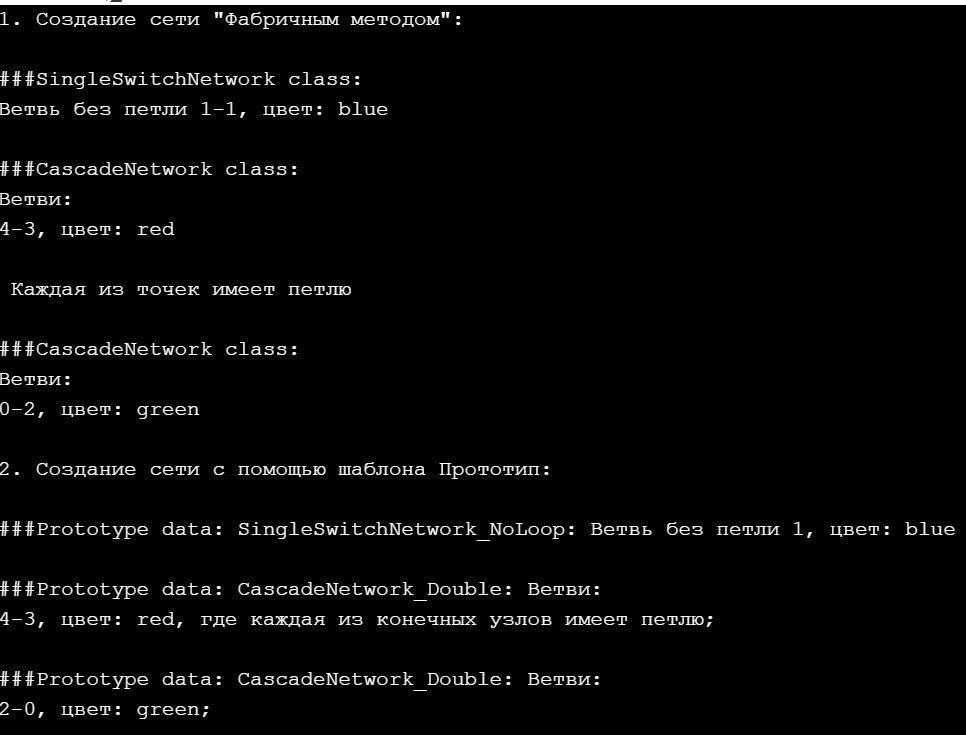


Рис. 2. Результаты вывода на экран

В данной лабораторной работе сразу можно сравнить реализации паттерна «Фабрика» и «Прототип». С помощью «Прототипа» мы решили нашу задачу, затратив минимум времени на написание кода(общий код написанного кода около 38 строк), что совершенно несоизмеримо с потраченным временем на написание «Фабрики». Именно акцент на итоговой реализации объекта является решающим фактором в выборе паттерна специалистами.