МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОУ НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Оглавление

[Введение 2](#_Toc124245941)

[1. Постановка задачи (Вариант 4) 3](#_Toc124245942)

[2. Анализ предметной области 3](#_Toc124245943)

[А) Текстовое описание сущностей предметной области с указанием их свойств и функций. 3](#_Toc124245944)

[Б) Текстовое описание декомпозиции или интеграции сущностей, делегирование функций одними сущностями другим. 4](#_Toc124245945)

[В) Текстовое описание ассоциаций (агрегаций, композиций) и обобщений. 5](#_Toc124245946)

[3. Разработка объектно-ориентированной модели предметной области. 5](#_Toc124245947)

[3.1 Описание структуры классов (POJO, Java Bean или другая). 5](#_Toc124245948)

[3.2 Описание подсистемы управления созданием объектов, описание применения порождающих шаблонов проектирования. 6](#_Toc124245949)

[3.3 Разработка объектно-ориентированной модели предметной области (с построением диаграммы классов). Обоснование слабой межмодульной связанности и расширяемости системы. 6](#_Toc124245950)

[4. Реализация объектно-ориентированной библиотеки и приложения: 7](#_Toc124245951)

[4.1 Аналитический обзор языков программирования (привести краткую характеристику 2-3 варианта). Описать преимущества и недостатки каждого инструмента, обосновать выбор языка программирования для решения задачи по варианту. 7](#_Toc124245952)

[4.2 Описание алгоритмов, реализуемых программно (привести блок-схемы хотя бы двух алгоритмов). Описание реализации методов. 11](#_Toc124245953)

[4.3 Описание тестового примера: входные, выходные и промежуточные данные для каждого теста, привести скриншоты тестов, из которых очевидна корректность работы программы (при необходимости добавьте пояснения к скриншотам). 14](#_Toc124245954)

[Заключение 16](#_Toc124245955)

[Список литературы 17](#_Toc124245956)

[Приложение 1. Программный код 17](#_Toc124245957)

[Приложение 2. Руководство пользователя консольного приложения 17](#_Toc124245958)

# Введение

Программные проекты зачастую достаточно сложны, и их декомпозиция - основная стратегия борьбы со сложностью. Она состоит в разбиении проблемы на мелкие управляемые элементы. ООП — это методология программирования, которая основана на представлении программы в виде совокупности объектов, каждый из которых является реализацией определенного класса (типа особого вида), а классы образуют иерархию на принципах наследуемости». Основная идея объектно-ориентированного анализа и проектирования информационных систем состоит в рассмотрении предметной области и логического решения задачи с точки зрения объектов (понятий или сущностей). В процессе объектно-ориентированного анализа основное внимание уделяется определению и описанию объектов (или понятий) в терминах предметной области. В процессе объектно-ориентированного проектирования определяются логические программные объекты, которые будут реализованы средствами объектно-ориентированного языка программирования. Эти программные объекты включают в себя атрибуты и методы. Шаблоны проектирования — это один из инструментов, который помогает сэкономить время и сделать более качественное решение. Благодаря паттернам разработчику легко не только понимать чужой код, но и расширять его своими решениями, а также добавлять новые.

Необходимо реализовать консольную утилиту, позволяющую создавать графовые структуры, добавлять узлы и связи, причем каждый узел определяется именем и описанием, а каждая связь – именем узла источника, именем узла приемника, описанием отношения. Полученный граф необходимо распечатать в консоли, в форматах матриц смежности и инцидентности. Необходимо обеспечить возможность расширения по алгоритмам обработки графов. Таким образом нам необходимо создать множество объектов для чего я воспользуюсь паттерном **строитель –** это порождающий паттерн проектирования, который позволяет создавать сложные объекты пошагово. Строитель даёт возможность использовать один и тот же код строительства для получения разных представлений объектов.

Цель курсовой работы. – разработать требуемую модель документа, научиться работать с ООП и паттернами программирования, а также с языком программирования Java.

# Постановка задачи (Вариант 4)

Предложите систему команд и реализуйте консольную утилиту, позволяющую создавать графовые структуры, добавлять узлы и связи, причем каждый узел определяется именем и описанием, а каждая связь — именем узла источника, именем узла приемника, описанием отношения. Полученный граф необходимо распечатать в консоли, в форматах матриц смежности и инцидентности. Необходимо обеспечить возможность расширения по алгоритмам обработки графов.

# Анализ предметной области

## А) Текстовое описание сущностей предметной области с указанием их свойств и функций.

Программа должна уметь:

* Создавать графовые структуры
* Хранить различные описания
* Печатать граф в виде различных матриц
* Так же необходимо обеспечить возможность расширения по алгоритмам обработки графов

Создание графовой структуры я оформлю простым вводом через консоль, с последующим хранением имен узлов в одномерном массиве, связей в двумерном, где первый индекс обозначает узел источник, второй – узел приемник, а элемент массива содержит саму связь.

Ввод и хранения описаний будет осуществляться точно по такому же принципу.

Чтобы вывести граф в виде матриц смежности и инцидентности, нужно эти матрицы создать. Создание матриц (двумерных массивов) я предоставлю специальным классам строителям, которые создадут матрицы (трансформируют граф).

Возможность расширения по алгоритмам обработки графов будет достигнута, благодаря удобному и понятному хранению графа. Массивы графа можно будет легко скопировать в хэш-таблицы (например, для использования алгоритма Дейкстры – поиск кротчайшего пути) или в связанный список.

Паттерн строитель обеспечит понятную структуру приложения, которую можно будет легко изменить или расширить.

**Классы:**

Graph, Description, TheAdjacencyMatrix, TheincidenceMatrix – это классы продукты, которые будут хранить поля (массивы графа, описаний, матриц). Доступ к полям будет осуществляться с помощью специальных методов сеттеров и геттеров. Тем самым мы повысим контроль за данными и их безопасность. (инкапсуляция)

**Интерфейсы**:

MaxtrixBuilder, GraphBuilder объявляют шаги конструирования продуктов, общие для всех видов конкретных строителей. Шаги, в моем случае - это также методы для взаимодействия с объектами. Шагами паттерна **строителя** они оформлены для удобства.

**Конкретные классы строители:**

ConcreteGraphBuilder, ConcreteDescriptionBuilder, ConstructАnAdjacencyMatrix, ConstructАnIncidenceMatrix - это классы реализующие интерфейс строителя и содержащие непосредственно код.

Вишенкой на торте послужит класс User, содержащий ссылки на методы конкретных строителей (шаги постройки объекта). Это своего рода директор из паттерна **строителя**. Однако тут мы выступает в роли директора и решаем порядок вызова шагов непосредственно во время работы программы, а не по заранее написанному коду. Некоторые шаги будут недоступны пока не выполнен какой-нибудь определенный. Например, изменение имени узла недоступно, до ввода графа.

## Б) Текстовое описание декомпозиции или интеграции сущностей, делегирование функций одними сущностями другим.

Граф и описания создаются пользователем после чего записываются в классы- продукты. Далее пользователю предоставляется выбор операций, которые класс user вызывает из конкретных строителей.

Класс - продукт графа содержит поля, которые передаются в остальные классы-продукты для дальнейшего использования другими конкретными строителями.

Конкретные классы строителя графа и описаний делегирует некоторую работу утилитарному классу «полезные функции». Для простоты и большей внятности кода.

## В) Текстовое описание ассоциаций (агрегаций, композиций) и обобщений.

Агрегация (агрегирование по ссылке) — отношение «часть-целое» между двумя равноправными объектами, когда один объект (контейнер) имеет ссылку на другой объект. Оба объекта могут существовать независимо: если контейнер будет уничтожен, то его содержимое — нет.

Композиция (агрегирование по значению) — более строгий вариант агрегирования, когда включаемый объект может существовать только как часть контейнера. Если контейнер будет уничтожен, то и включённый объект тоже будет уничтожен.

Обобщение и наследование позволяют выявить аналогии между различными классами объектов, определяют многоуровневую классификацию объектов.

Описание связей в своем приложении считаю логичным начать «сверху вниз»

Класс User пользуется функцианалом конкретных классов. Ассоциация кратности {1;\*}

Конкретные строители реализуют интерфейсы. Реализация.

Конкретные строители берут или записывают некоторые поля в свои классы-продукты. Двойная ассоциация кратности {\*,1}. Любое количество объектов строителей, один объект продукта.

Так же классы-продукты описаний, матриц инцидентности и смежности содержат в себе поля, которые берутся из класса-продукта. Таким образом 3 продукта пользуются графом. Ассоциация кратности {1,1}.

# Разработка объектно-ориентированной модели предметной области.

## Описание структуры классов (POJO, Java Bean или другая).

JavaBean — это объект Java, который удовлетворяет определенным соглашениям программирования:

* класс JavaBean должен реализовывать либо Serializable, либо Externalizable;
* класс JavaBean должен иметь общедоступный конструктор без аргументов;
* все свойства JavaBean должны иметь общедоступные методы установки и получения (при необходимости);
* все переменные экземпляра JavaBean должны быть закрытыми.

Класс POJO - это обычный класс без каких-либо специальностей, класс, полностью слабо связанный с технологией/фреймворком. Класс не реализуется из технологии/фреймворка и не расширяется из API-интерфейса технологии/фреймворка.

Многие классы в будущем приложении расширяются из интерфейсов, а также удовлетворяют соглашениям JavaBean, следовательно, структура классов – JavaBean.

## Описание подсистемы управления созданием объектов, описание применения порождающих шаблонов проектирования.

Программа создает граф, описания, матрицы. Их создания, изменения (граф, описания) тоже являются шагом конечного результата, даже если пользователь не будет ни разу изменять граф и остальные объекты, возможность ему предоставляется. Для создания объектов подойдет паттерн строитель. С его помощью удасться выстроить четкую и понятную структуру программы.

## Разработка объектно-ориентированной модели предметной области (с построением диаграммы классов). Обоснование слабой межмодульной связанности и расширяемости системы.

Программный код четко разбит на модули: продукты, строители, «полезные функции», интерфейсы и главный класс user.

Продукты отвечают за хранение, прием и отдаву полей. Это дает четкое понимание откуда нужно взять поля для содания нового объекта, анализа и т.д.

Строители содержат львиную долю кода, а благодаря паттерну строитель эти классы разбиты на более мелкие шаг, благодаря чему улучшается контроль за кодом. По личнному опыту скажу, что отслеживание изменения полей и справление ошибок внутри классов строителей не представляет проблем.

# Реализация объектно-ориентированной библиотеки и приложения:

## 4.1 Аналитический обзор языков программирования (привести краткую характеристику 2-3 варианта). Описать преимущества и недостатки каждого инструмента, обосновать выбор языка программирования для решения задачи по варианту.

В качестве методологии программирования было выбрано объектно-ориентированное программирование. Самыми популярными и востребованными языками ООП являются: Java, Python, C++, C#, JavaScript, Ruby.

**Java** — язык программирования общего назначения. Относится к объектно-ориентированным языкам программирования, к языкам с сильной типизацией. Создатели реализовали принцип WORA: write once, run anywhere или «пиши один раз, запускай везде». Это значит, что написанное на Java приложение можно запустить на любой платформе, если на ней установлена среда исполнения Java (JRE, Java Runtime Environment).

Плюсы:

1) Концепция ООП (объектно-ориентированного программирования). Это значит, что программист сам определяет тип данных, его структуру и набор применяемых к нему функций.

2) Это язык высокого уровня, т.е., он больше похож на человеческую речь, а не на машинный код. Следовательно, у него сравнительно простой синтаксис, что делает его быстрым для освоения и удобным для написания кода, его чтения и обслуживания. Есть и более простые варианты (например, [Python](https://www.cischool.ru/product-category/python-2/)), однако у человека с базовым пониманием основ программирования здесь не должно возникнуть сложностей;

3) Безопасность. У Java есть несколько функций, которые ликвидируют часто встречающиеся уязвимости. В частности, это Security Manager – создаваемая для каждого приложения политика безопасности, в которой можно прописать правила доступа;

4) Удобство для распределённого программирования. Этот язык изначально создавался для совместной работы (в том числе удалённой), поэтому он позволяет совместно использовать данные и программы несколькими компьютерами одновременно;

5) Принцип «написать один раз и использовать везде» — написанное на Java приложение можно запустить на любой поддерживающей его платформе;

6) Стабильное и постоянно развивающееся сообщество. По многочисленности и активности с ним мало кто может соперничать. В Сети есть масса ресурсов, где на любой вопрос по этой теме либо уже есть ответ, либо найдётся кто-нибудь, кто его подскажет, равно как и сотни курсов, семинаров и обучающих программ, как платных, так и бесплатных.

Минусы:

1) Низкая скорость. Все высокоуровневые языки приходится компилировать с помощью виртуальной машины, что плохо сказывается на их производительности. Java – не исключение, кроме того, у него есть и некоторые собственные особенности, вызывающие дополнительные проблемы с производительностью;

2) Многословие (verbosity). Сходство с естественными языками делает Java проще для изучения и понимания, но также ведёт и к тому, что он содержит много лишней информации и довольно громоздок;

**Python** — универсален и используется во многих местах. Однако Python прочно обосновался в машинном обучении и науке о данных. Это один из предпочтительных языков для этой новой и постоянно растущей области. Он имеет четко структурированное семантическое ядро и достаточно простой синтаксис. Все, что пишется на этом языке, всегда легко читаемо. В случае необходимости передать аргументы язык использует функцию call-by-sharing. Набор операторов в языке вполне стандартен. Удобная особенность синтаксиса – это форматирование текста кода при помощи разбивки их на блоки с помощью отступов, которые создают нажатием клавиш «Space» и «Tab». В синтаксисе отсутствуют фигурные или операторные скобки, обозначающие начало и конец блока. Такое решение заметно сокращает количество строк тела программы и приучает программиста соблюдать хороший стиль и аккуратность при написании кода.

Плюсы Python:

1) Понятность кода. Синтаксическая особенность Python — выделение блоков кода отступами, что значительно упрощает зрительное восприятие программ, написанных на этом языке.

2) Интерпретируемость. Программы, написанные на языке программирования Python, не переводятся в машинный код, а сразу выполняются программой-интерпретатором. Это позволяет запускать код на любой платформе с установленным заранее интерпретатором.

3) Объектно ориентированность. Python — это язык, созданный согласно парадигме объектно ориентированного программирования (ООП). В ней основными являются понятия объекта и класса. Классы — это специальные типы данных, объекты — экземпляры классов. То есть любое значение является объектом конкретного класса. В Python вы можете не только использовать уже существующие классы, но и создавать свои собственные.

4) Динамическая типизация. В отличие от C-подобных языков программирования, в Python переменные связываются с типом в момент присваивания в них конкретных значений.

Минусы:

1) Python является одним из самых медленных языков программирования.

2) Python не подходит для задач, которые требуют большого объема памяти.

**C ++** — обладает скоростью C с функциональностью классов и объектно-ориентированной парадигмой. Это скомпилированный, надежный и мощный язык. Фактически, он даже используется для создания компиляторов и интерпретаторов для других языков. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником — языком C, — наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования.

Плюсы

1) Поддержка объектно-ориентированного программирования (ООП). ООП помогает сделать код проще, и его быстрее писать. Большой цикл статей про ООП.

2) Высокая скорость.

3) Возможности для работы с данными на низком уровне — то есть на уровне, близком к аппаратному. Благодаря этому на С++ можно писать драйвера, микроконтроллеры.

4) Популярность:

Для С++ создано много библиотек и компиляторов.

С++ используется практически везде (несколько примеров мы уже привели выше).

5) Синтаксис С++ похож на синтаксис C, С# и Java, так что переключаться между этими языками достаточно легко.

6) Совместимость с C благодаря тому, что С++ создавался на его основе.

Минусы

1) Небезопасность: С++ даёт большую свободу действий, но и не удержит вас от ошибок. А лёгкий доступ к памяти делает его уязвимым не только во время хакерских атак, но и при неосторожной работе.

2) Зависимость от платформы: написать на С++ портативный код (такой, который бы работал на разных платформах) очень сложно.

3) Синтаксис строгий и «многословный»: код читается хуже, чем в некоторых других языках

4) Сложность: у С++ сложный синтаксис и маленькая стандартная библиотека, а ещё надо разбираться в указателях и работе с памятью, поэтому учить его нелегко, особенно с нуля.

**Сравнение Python и Java** (Python — это интерпретируемый язык с динамической типизацией. Java же — компилируемый язык со статической типизацией. Эти различия делают Python и Java полными противоположностями друг друга в плане скорости запуска и выполнения программ. Код, написанный на Python, быстрее запускается и дольше выполняется. В то время как программы на Java медленнее запускаются, но гораздо быстрее выполняются. С помощью Java можно разрабатывать кроссплатформенные приложения, но и Python также совместим со многими операционными системами. Также с помощью этих языков программирования разработчики могут создавать сетевые приложения. Если говорить о сложности этих двух языков, то Java безусловно уступает Python в простоте изучения.)

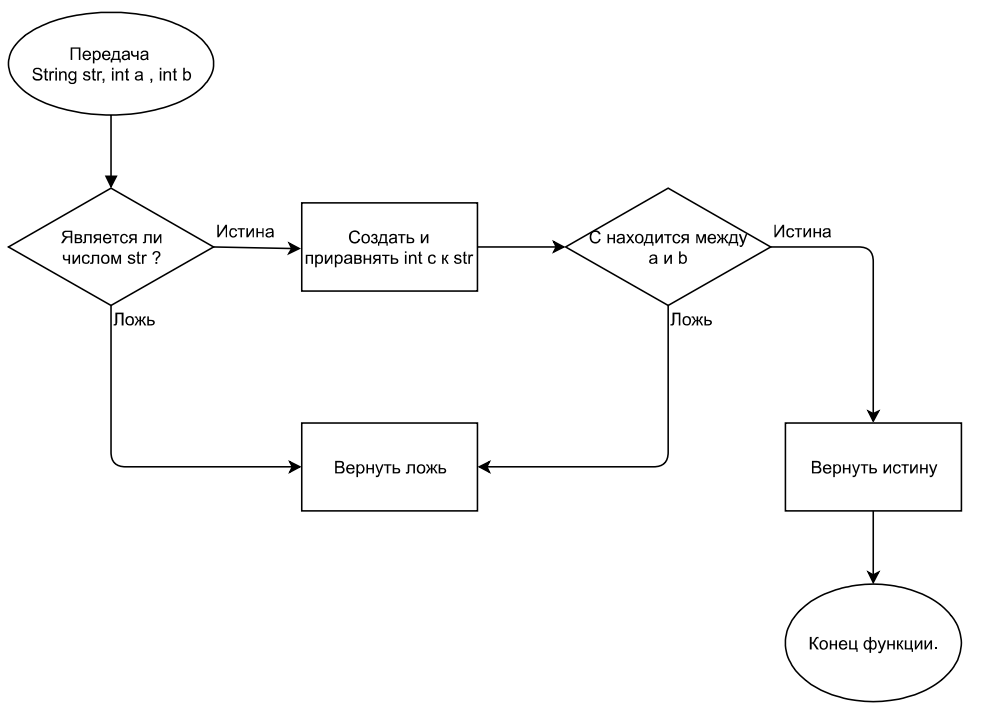
#### Сравнение С++ и Java (Зависимость от платформы: при компиляции программа на Java сначала переводится в особый байт-код с помощью интерпретатора. Благодаря этому код в Java можно запустить на любой платформе. С++ при компиляции переводится в машинный код, поэтому это платформо-зависимый язык. Указатели: огромная часть работы в С++ — работа с указателями, а в Java их нельзя использовать. В Java нет многих комплексных функций, которые есть в С++, — например структур, указателей и объединений. Многопоточность поддерживается в Java, но не в С++. Уровень языка: и Java, и С++ — высокоуровневые языки, но на Java работать на низком уровне не получится, в отличие от С++.)

**Сравнение Python и C++** (Одно из главных достоинств Python — простой и понятный синтаксис. Программистам с C++ он будет понятен почти сразу, пусть изначально может и не хватать скобок и точек с запятой. У Python огромная стандартная библиотека с ридерами/райтерами для CSV, ZIP и других форматов, XML-парсеры, инструменты для работы с сетью и так далее. Язык подходит для создания веб-приложений. Лучше всего подходит для машинного обучения. Главное преимущество C++ — производительность. Его скорость работы намного выше в сравнении с Python. C++ подходит почти для всех платформ, а также для встроенных систем, в то время как Python работает только на отдельных платформах, поддерживающих высокоуровневые языки. C++ более предсказуем благодаря статической типизации. Это же влияет и на производительность. При работе с C++ можно изучать низкоуровневое программирование, ведь язык близок к железу. В случае с Python это не сработает.)

По итогу основным языком разработки был выбран язык Java. Он является универсальным языком программирования. Код на этом языке компилируется не в машинный код, а в специальный байт-код, который исполняется виртуальной машиной Java, а версии языка обладают обратной совместимостью, поэтому код скомпилированный однажды может быть запущен везде, где есть JVM. Для решения задачи по варианту он подходит больше, т.к. имеет ряд преимуществ. Он позволяет многократно использовать одни и те же объекты в разных программах, более удобно для организации структуры программ (особенно с большим объёмом кода), помогает избегать ошибок и упрощает поддержку и модернизацию старого кода

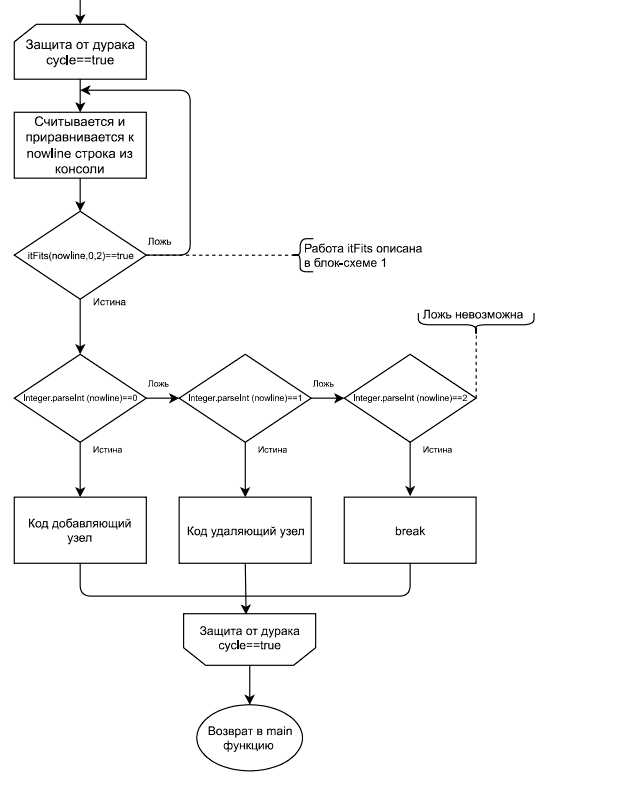
## 4.2 Описание алгоритмов, реализуемых программно (привести блок-схемы хотя бы двух алгоритмов). Описание реализации методов.

Блок-схема метода itFits из класса (Блок-схема 1). Данный алгоритм определяет можно ли сделать явное преобразование стоокого значения str в int и, если преобразование возможно, находится ли str в диапазоне между аргументами a и b? Данный алгоритм (метод) возвращается Boolean значение.



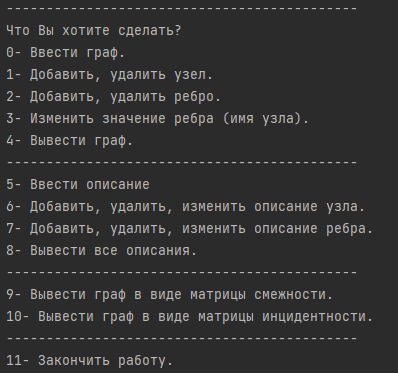
Блок-схема 1

Блок схема метода AddDeleteNote из класса ConcreteGraphBuilder (блок-схема 2 ). Данный алгортим представляет из себя цикл, проверки введеной строки. Конечно, можно было сравнивать введеную строку со String «0», «1», «2», но заранее созданные «полезные функции» позволии мне немного оптимизировать код и использовать утилитарные функции и в других местах. (Блок-схема 1 раскрывает вызываемую функцию itFits)

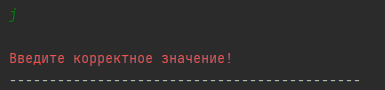


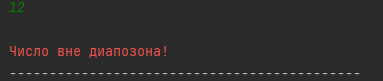
Блок-схема 2

## 4.3 Описание тестового примера: входные, выходные и промежуточные данные для каждого теста, привести скриншоты тестов, из которых очевидна корректность работы программы (при необходимости добавьте пояснения к скриншотам).

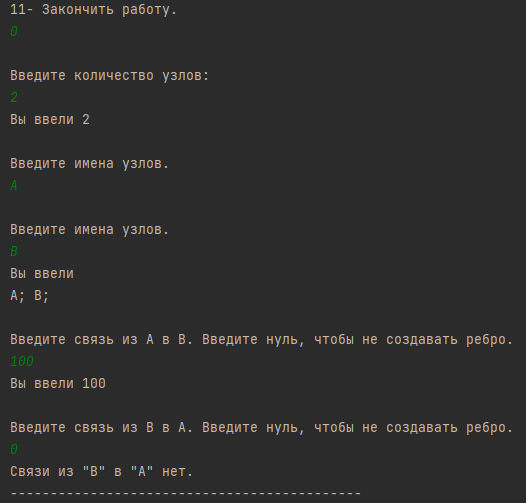


После запуска вам препоставится выбор.

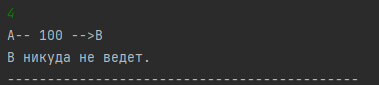




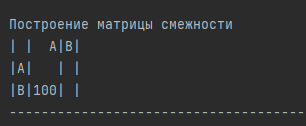
Подобная защита от ошибок реализована во всей программе.



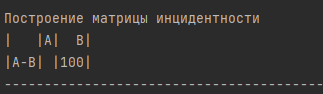
Чтобы программа разрешила вам выбирать пункты 1-11 нужно ввести граф.(Выбор варианта 0)



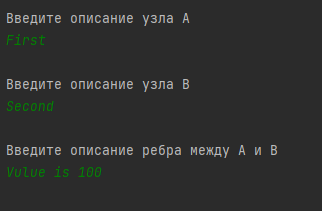
Печать графа.



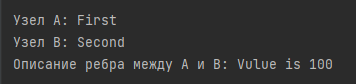
Выбор варианта 9



Выбор варианта 10



Выбор варианта 5



Выбор варианта 8.

# Заключение

Разработано консольное приложение способное создавать графы, хранить информацию о его элементах и печатать граф в виде двух матриц. Данное приложение имеет некоторый функционал, который позволяет работать с графами и не бояться допустить ошибок при вводе.

Преимуществами программы является простота использования, скрытность и защищенность полей, высокая четкость структуры, позволяющая расширить ее без особых проблем.

Недостатками являются большое количество кода в конкретных строителях (граф, описание), отсутствие пометок (Mark). Нет классов или интерфейсов, которые бы улучшили структуру взаимодействия классов, осуществляющих предостовление услуг и непосредлственно исполняющих эти услуги.

В перспективах: добавить большее количество функций, таких как добавить узел в определенное место, найти кратчайший путь от одного узла к другим и подобные алгоритмы. Также планирую существующие и после добавленные алгоритмы для работы с графами, например возможность давать ребра цифровые значения для различных алгоритмов, печатать граф в графически, производить операции над матрицами (кружочки, стрелочки) (и остальные «услуги») реализовать в виде паттерна команды. Таким образом создать связку паттерна команда, строитель и еще одного паттерна для взаимодействия с пользователем (или, безусловно, графический интерфейс). Помимо добавления основного функционала, постараюсь добавить второстепенный, по типу сохранения изменений или возможноси отката изменений.

Доработанное консольное приложение можно использовать для создания графов и произведения операциями над ними и строящимися на их основе матрицами.

# Список литературы

1. Погружение в паттерны проектирования / Александр Швец изд. 2018

2. JavaBeans / <https://ru.wikipedia.org/wiki/JavaBeans>

3. Языки объектно-ориентированного программирования / <https://bestprogrammer.ru/programmirovanie-i-razrabotka/yazyki-obektno-orientirovannogo-programmirovaniya>

4. Python краткий обзор языка / <https://techrocks.ru/2019/01/21/about-python-briefly/>

5. Различия Python и C++ / <https://pythonru.com/baza-znanij/python-ili-c>

6.Что нужно знать о С++ / <https://skillbox.ru/media/code/vybiraem_yazyk_programmirovaniya_chto_nuzhno_znat_o_s_>

7. Плюсы и минусы Java <https://www.cischool.ru/plyusy-i-minusy-java/>

8. Плюсы и минусы Python https://skysmart.ru/articles/programming/preimushestva-i-nedostatki-python

9. <https://codelab.ru/cat/patterns>

10. <http://book.uml3.ru/sec_3_3>

11. https://github.com/blinky-z/OOP-Tutorial

# Приложение 1. Программный код

**package** course.paper;

**import** **static** course.paper.UsefulFunctions.*itFits*;

**import** **static** course.paper.UsefulFunctions.*notEmpty*;

**import** java.util.Scanner;

**public** **class** User {

**private** **static** **boolean** *cycle*=**true**;

**public** **static** **void** main(String[] args) {

ConcreteGraphBuilder Graph = **new** ConcreteGraphBuilder();

ConcreteDescriptionBuilder Description = **new** ConcreteDescriptionBuilder();

ConstructАnAdjacencyMatrix Adjacency = **new** ConstructАnAdjacencyMatrix();

ConstructАnIncidenceMatrix Incidence = **new** ConstructАnIncidenceMatrix();

String nowLine;

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

**int** a;

**while**(*cycle*==**true**){

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

System.***out***.println("Что Вы хотите сделать?");

System.***out***.println("0- Ввести граф.");

System.***out***.println("1- Добавить, удалить узел.");

System.***out***.println("2- Добавить, удалить ребро.");

System.***out***.println("3- Изменить значение ребра (имя узла).");

System.***out***.println("4- Вывести граф.");

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

System.***out***.println("5- Ввести описание");

System.***out***.println("6- Добавить, удалить, изменить описание узла.");

System.***out***.println("7- Добавить, удалить, изменить описание ребра.");

System.***out***.println("8- Вывести все описания.");

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

System.***out***.println("9- Вывести граф в виде матрицы смежности.");

System.***out***.println("10- Вывести граф в виде матрицы инцидентности.");

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

System.***out***.println("11- Закончить работу.");

nowLine=in.nextLine();

**if** (*itFits*(nowLine,0,11)==**true**){

**if** (*notEmpty*(nowLine)==**true**){

a=Integer.*parseInt*(nowLine);

**switch**(a){

**case**(0) -> Graph.Enter();

**case**(1) -> Graph.AddDeleteNote();

**case**(2) -> Graph.AddDeleteEdge();

**case**(3) -> Graph.Change();

**case**(4) -> Graph.Print();

**case**(5) -> Description.Enter();

**case**(6) -> Description.AddDeleteNote();

**case**(7) -> Description.AddDeleteEdge();

**case**(8) -> Description.Print();

**case**(9) -> Adjacency.Print\_Matrix();

**case**(10) -> Incidence.Print\_Matrix();

**case**(11) -> *cycle*=**false**;

}

}

}

}

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**package** course.paper;

**import** **static** course.paper.UsefulFunctions.*itFits*;

**import** **static** course.paper.UsefulFunctions.*notEmpty*;

**import** java.util.Scanner;

**public** **class** User {

**private** **static** **boolean** *cycle*=**true**;

**public** **static** **void** main(String[] args) {

ConcreteGraphBuilder Graph = **new** ConcreteGraphBuilder();

ConcreteDescriptionBuilder Description = **new** ConcreteDescriptionBuilder();

ConstructАnAdjacencyMatrix Adjacency = **new** ConstructАnAdjacencyMatrix();

ConstructАnIncidenceMatrix Incidence = **new** ConstructАnIncidenceMatrix();

String nowLine;

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

**int** a;

**while**(*cycle*==**true**){

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

System.***out***.println("Что Вы хотите сделать?");

System.***out***.println("0- Ввести граф.");

System.***out***.println("1- Добавить, удалить узел.");

System.***out***.println("2- Добавить, удалить ребро.");

System.***out***.println("3- Изменить значение ребра (имя узла).");

System.***out***.println("4- Вывести граф.");

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

System.***out***.println("5- Ввести описание");

System.***out***.println("6- Добавить, удалить, изменить описание узла.");

System.***out***.println("7- Добавить, удалить, изменить описание ребра.");

System.***out***.println("8- Вывести все описания.");

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

System.***out***.println("9- Вывести граф в виде матрицы смежности.");

System.***out***.println("10- Вывести граф в виде матрицы инцидентности.");

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

System.***out***.println("11- Закончить работу.");

nowLine=in.nextLine();

**if** (*itFits*(nowLine,0,11)==**true**){

**if** (*notEmpty*(nowLine)==**true**){

a=Integer.*parseInt*(nowLine);

**switch**(a){

**case**(0) -> Graph.Enter();

**case**(1) -> Graph.AddDeleteNote();

**case**(2) -> Graph.AddDeleteEdge();

**case**(3) -> Graph.Change();

**case**(4) -> Graph.Print();

**case**(5) -> Description.Enter();

**case**(6) -> Description.AddDeleteNote();

**case**(7) -> Description.AddDeleteEdge();

**case**(8) -> Description.Print();

**case**(9) -> Adjacency.Print\_Matrix();

**case**(10) -> Incidence.Print\_Matrix();

**case**(11) -> *cycle*=**false**;

}

}

}

}

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**package** course.paper;

**public** **interface** MatrixBuilder {

**public** **void** Fill\_in\_matrix();

**public** **void** Counting\_lengths();

**public** **void** Creating\_gaps();

**public** **void** Print\_Matrix();

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**package** course.paper;

**public** **interface** GraphBuilder {

**public** **void** Enter ();

**public** **void** AddDeleteNote ();

**public** **void** AddDeleteEdge ();

**public** **void** Change ();

**public** **void** Print ();

}

package course.paper;

import static products.Graph.get\_Number\_of\_nodes;

import static products.Graph.get\_associations;

import static products.Graph.get\_graph;

import static products.Graph.get\_namesOfNodes;

import static products.TheincidenceMatrix.\*;

public class ConstructАnIncidenceMatrix implements MatrixBuilder{

//Поля графа.

private static int Number\_of\_nodes;

private static String associations[][];

private static String namesOfNodes[];

//Поля матрицы инцидентности.

private static int Connection; //Кол-во ребер.

private static String ProbInc[][]; //Пробелы в виде String для печати перед каждый элементом м.инц.

private static String Inc[][]; //Матрица инцидентности.

private static int maxLongInc[]; //Максимальная длина столбца м.инц.

private static int IndInc[][]; //Кол-во необходимых пробелом перед каждым элементом.

private static int justLongInc[][]; //Длина каждого элемента м.инц.

//Подсчет Connection

@Override

public void Fill\_in\_matrix(){

Number\_of\_nodes=get\_Number\_of\_nodes();

namesOfNodes=get\_namesOfNodes();

associations=get\_associations();

Connection=0;

for (int i=0;i<Number\_of\_nodes;i++){

for (int j=0;j<Number\_of\_nodes;j++){

if (i!=j){

if (!"".equals(associations[i][j])) {

Connection++;

}

}

}

}

Inc=new String [Number\_of\_nodes+1][Connection+1];

set\_long\_Inc(Number\_of\_nodes+1);

Inc[0][0]="";

for (int i=1;i<Number\_of\_nodes+1;i++){

Inc[i][0]=namesOfNodes[i-1];

}

int count=1;

for (int i=0;i<Number\_of\_nodes;i++){

for (int k=0;k<Number\_of\_nodes;k++){

if (i!=k){

if (!"".equals(associations[i][k])){

Inc[0][count]=namesOfNodes[i]+"-"+namesOfNodes[k];

Inc[k+1][count]=associations[i][k];

count++;

}

}

}

}

for (int i=0;i<Number\_of\_nodes+1;i++){

for (int j=0;j<Connection+1;j++){

if (Inc[i][j]==null){

Inc[i][j]="";

}

}

}

}

//Подсчет maxLongInc и justLongInc

@Override

public void Counting\_lengths(){

maxLongInc=new int[Number\_of\_nodes+1];

justLongInc= new int[Number\_of\_nodes+1][Connection+1];

set\_long\_maxLongInc(Number\_of\_nodes+1);

set\_long\_justLongInc(Number\_of\_nodes+1);

int max;

for (int i=0;i<Number\_of\_nodes+1;i++){

max=0;

for (int k=0;k<Connection+1;k++){

if (max<Inc[i][k].length()){

max=Inc[i][k].length();

maxLongInc[i]=max;

}

}

}

for (int i=0;i<Number\_of\_nodes+1;i++){

for (int k=0;k<Connection+1;k++){

justLongInc[i][k]=Inc[i][k].length();

}

}

}

//Подсчет, создание IndInc и ProbInc

@Override

public void Creating\_gaps(){

IndInc= new int[Number\_of\_nodes+1][Connection+1];

ProbInc= new String[Number\_of\_nodes+1][Connection+1];

set\_long\_IndInc(Number\_of\_nodes+1);

set\_long\_ProbInc(Number\_of\_nodes+1);

for (int i=0;i<Number\_of\_nodes+1;i++){

for (int k=0;k<Connection+1;k++){

ProbInc[i][k]=" ";

IndInc[i][k]=/\*1\*/+(maxLongInc[i]-justLongInc[i][k]);

ProbInc[i][k]= ProbInc[i][k].repeat(IndInc[i][k]);

}

}

set\_maxLongInc(maxLongInc);

set\_Inc(Inc);

set\_justLongInc(justLongInc);

set\_IndInc(IndInc);

set\_ProbInc(ProbInc);

set\_Connection(Connection);

}

//Вывод матрицы.

@Override

public void Print\_Matrix(){

if (get\_graph()==false){

System.out.println("");

System.out.println((char) 27 + "[31mСперва введите граф! " + (char)27 + "[0m");

}

else{

Fill\_in\_matrix();

Counting\_lengths();

Creating\_gaps();

System.out.println("");

System.out.println("Построение матрицы инцидентности");

for (int y=0;y<Connection+1;y++){

for (int x=0;x<Number\_of\_nodes+1;x++){

System.out.print("|"+ProbInc[x][y]+Inc[x][y]);

}

System.out.print("|\n");

}

}

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**package** course.paper;

**import** **static** products.Graph.\*;

**import** **static** products.TheAdjacencyMatrix.*set\_Adj*;

**import** **static** products.TheAdjacencyMatrix.*set\_IndAdj*;

**import** **static** products.TheAdjacencyMatrix.*set\_ProbAdj*;

**import** **static** products.TheAdjacencyMatrix.*set\_justLongAdj*;

**import** **static** products.TheAdjacencyMatrix.*set\_long\_Adj*;

**import** **static** products.TheAdjacencyMatrix.*set\_long\_IndAdj*;

**import** **static** products.TheAdjacencyMatrix.*set\_long\_ProbAdj*;

**import** **static** products.TheAdjacencyMatrix.*set\_long\_justLongAdj*;

**import** **static** products.TheAdjacencyMatrix.*set\_long\_maxLongAdj*;

**import** **static** products.TheAdjacencyMatrix.*set\_maxLongAdj*;

**public** **class** ConstructАnAdjacencyMatrix **implements** MatrixBuilder{

//Поля графа.

**private** **static** **int** *Number\_of\_nodes*;

**private** **static** String *namesOfNodes*[];

**private** **static** String *associations*[][];

//Поля матрицы смежности.

**private** **static** **int** *maxLongAdj*[]; //Максимальная длина столбца м.смеж.

**private** **static** String *Adj*[][]; //Матрица смежности.

**private** **static** **int** *justLongAdj*[][]; //Длина каждого элемента м.смеж.

**private** **static** **int** *IndAdj*[][]; //Необходимое кол-во пробелов перед каждым элементом м.смеж.

**private** **static** String *ProbAdj*[][]; //Пробелы в виде строки, чтобы печатать перед каждым эдементов м.смеж.

//Заполнение Adj

@Override

**public** **void** Fill\_in\_matrix() {

*Number\_of\_nodes*=*get\_Number\_of\_nodes*();

*namesOfNodes*=*get\_namesOfNodes*();

*associations*=*get\_associations*();

*Adj*=**new** String [*Number\_of\_nodes*+1][*Number\_of\_nodes*+1];

*set\_long\_Adj*(*Number\_of\_nodes*+1);

**for** (**int** i=1;i<*Number\_of\_nodes*+1;i++) //Первая строка и первый столбец заполняются именами узлом. Самый верхний левый элемент остается пустым.

{

*Adj*[i][0]=*namesOfNodes*[i-1];

*Adj*[0][i]=*namesOfNodes*[i-1];

}

**for** (**int** i=1;i<*Number\_of\_nodes*+1;i++) //Ассоциациями заполняются элементы помимо первых столбца и строки.

{

**for** (**int** k=1;k<*Number\_of\_nodes*+1;k++)

{

*Adj*[i][k]=*associations*[i-1][k-1];

}

}

**for** (**int** i=0;i<*Number\_of\_nodes*+1;i++) //Прогоняется весь массив и если элемента нет, вставляется строковая пустота. Вместо null, "".

{

**for** (**int** j=0;j<*Number\_of\_nodes*+1;j++)

{

**if** (*Adj*[i][j]==**null**)

{

*Adj*[i][j]="";

}

}

}

}

//Подсчет maxLongAdj и justLongAdj

@Override

**public** **void** Counting\_lengths() {

*maxLongAdj*=**new** **int**[*Number\_of\_nodes*+1];

*justLongAdj*= **new** **int**[*Number\_of\_nodes*+1][*Number\_of\_nodes*+1];

*set\_long\_maxLongAdj*(*Number\_of\_nodes*+1);

*set\_long\_justLongAdj*(*Number\_of\_nodes*+1);

**int** max;

**for** (**int** i=0;i<*Number\_of\_nodes*+1;i++) //Выбирается максимальная длина по столбцам.

{

max=0;

**for** (**int** k=0;k<*Number\_of\_nodes*+1;k++)

{

**if** (max<*Adj*[i][k].length())

{

max=*Adj*[i][k].length();

*maxLongAdj*[i]=max;

}

}

}

**for** (**int** i=0;i<*Number\_of\_nodes*+1;i++) //Заполняется массив длин длиной элементов.

{

**for** (**int** k=0;k<*Number\_of\_nodes*+1;k++)

{

*justLongAdj*[i][k]=*Adj*[i][k].length();

}

}

}

//Подсчет, заполнение IndAdj и ProbAdj

@Override

**public** **void** Creating\_gaps() {

*IndAdj*= **new** **int**[*Number\_of\_nodes*+1][*Number\_of\_nodes*+1];

*ProbAdj*= **new** String[*Number\_of\_nodes*+1][*Number\_of\_nodes*+1];

*set\_long\_IndAdj*(*Number\_of\_nodes*+1);

*set\_long\_ProbAdj*(*Number\_of\_nodes*+1);

**for** (**int** i=0;i<*Number\_of\_nodes*+1;i++)

{

**for** (**int** k=0;k<*Number\_of\_nodes*+1;k++)

{

*ProbAdj*[i][k]=" ";

*IndAdj*[i][k]=(*maxLongAdj*[i]-*justLongAdj*[i][k]); //Тут я цбрал +1

*ProbAdj*[i][k]= *ProbAdj*[i][k].repeat(*IndAdj*[i][k]);

}

}

*set\_maxLongAdj*(*maxLongAdj*);

*set\_Adj*(*Adj*);

*set\_justLongAdj*(*justLongAdj*);

*set\_IndAdj*(*IndAdj*);

*set\_ProbAdj*(*ProbAdj*);

}

//Печать матрицы смежности

@Override

**public** **void** Print\_Matrix() {

**if** (*get\_graph*()==**false**){

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

System.***out***.println((**char**) 27 + "[31mСперва введите граф! " + (**char**)27 + "[0m");

}

**else**{

Fill\_in\_matrix();

Counting\_lengths();

Creating\_gaps();

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

System.***out***.println("Построение матрицы смежности");

**for** (**int** y=0;y<*Number\_of\_nodes*+1;y++)

{

**for** (**int** x=0;x<*Number\_of\_nodes*+1;x++)

{

System.***out***.print("|"+*ProbAdj*[x][y]+*Adj*[x][y]);

}

System.***out***.print("|\n");

}

}

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

package course.paper;

import static course.paper.UsefulFunctions.\*;

import java.util.Scanner;

import static products.Description.\*;

import static products.Graph.\*;

public class ConcreteGraphBuilder implements GraphBuilder {

private static int Number\_of\_nodes;

private static String namesOfNodes[];

private static String associations[][];

private static String descriptionNotes[];

private static String descriptionAssociations[][];

private static int countOfNew;

private static int deletingNode;

@Override

public void Enter() {

set\_graph(true);

String nowLine;

boolean Cycle=true;

Scanner in = new Scanner(System.in);

while (Cycle==true){

System.out.println("");

System.out.println("Введите количество узлов:");

nowLine=in.nextLine();

if (isNumeric(nowLine)==true){

Number\_of\_nodes=(Integer.parseInt (nowLine));

Cycle=false;

}

}

System.out.println("Вы ввели "+ Number\_of\_nodes);

namesOfNodes = new String[Number\_of\_nodes];

associations=new String[Number\_of\_nodes][Number\_of\_nodes];

descriptionNotes=new String [Number\_of\_nodes];

descriptionAssociations=new String [Number\_of\_nodes][Number\_of\_nodes];

set\_long\_namesOfNodes(Number\_of\_nodes);

set\_long\_associations(Number\_of\_nodes);

set\_long\_descriptionAssociations(Number\_of\_nodes);

set\_long\_descriptionNotes(Number\_of\_nodes);

for(int i=0; i<Number\_of\_nodes; i++){

if(i==0){

namesOfNodes[i]=enteringString("\nВведите имена узлов.");

}

else {

namesOfNodes[i]=enteringString("\nВведите имена узлов.");

}

}

System.out.println("Вы ввели ");

for (int i=0; i<Number\_of\_nodes; i++){

System.out.print(namesOfNodes[i]+"; ");

}

System.out.println("");

for (int i=0;i<Number\_of\_nodes;i++){

for (int k=0;k<Number\_of\_nodes;k++){

if (i!=k){

associations[i][k]=enteringString("\nВведите связь из "+namesOfNodes[i]+" в "+namesOfNodes[k]+". Введите нуль, чтобы не создавать ребро.");

if ("0".equals(associations[i][k])){

associations[i][k]="";

System.out.println("Связи из \""+namesOfNodes[i]+"\" в \""+namesOfNodes[k]+"\" нет.");}

else{

System.out.println("Вы ввели "+associations[i][k]);

}

}

}

}

set\_Number\_of\_nodes(Number\_of\_nodes);

set\_associations(associations);

set\_namesOfNodes(namesOfNodes);

for (int k=0;k<Number\_of\_nodes;k++)

{if(k<Number\_of\_nodes)

{

descriptionNotes[k]="";

}

}

for (int i=0;i<Number\_of\_nodes;i++)

{

for (int j=0;j<Number\_of\_nodes;j++)

{

if (i!=j){

if (!"".equals(associations[i][j])) {

descriptionAssociations[i][j]="";

}

}

}

}

set\_descriptionAssociations(descriptionAssociations);

set\_descriptionNotes(descriptionNotes);

}

@Override

public void AddDeleteNote() {

String newnamesOfNodes[];

String newassociations[][];

String newDESOfNodes[];

String newDESofAss[][];

Scanner in = new Scanner(System.in);

if (get\_graph()==false){

System.out.println("");

System.out.println((char) 27 + "[31mСперва введите граф! " + (char)27 + "[0m");

}

else{

boolean cycle=true;

String nowline;

while (cycle==true){

System.out.println("");

System.out.println("Что Вы хотите сделать? 0- добавить узел, 1- удалить узел, 2-выйти");

nowline=in.nextLine();

if (itFits(nowline,0,2)==true){

if (Integer.parseInt (nowline)==0){

////////////////////////////////////////////ДОБАВЛЕНИЕ УЗЛА//////////////////////////////////////////////

System.out.print("\n");

boolean Cycle1=true;

while (Cycle1==true){

System.out.println("");

System.out.println("Сколько добавить узлов?");

nowline=in.nextLine();

if (isNumeric(nowline)==true){

countOfNew=Integer.parseInt (nowline);

Cycle1=false;

}

}

newnamesOfNodes=new String[Number\_of\_nodes+countOfNew];

newassociations=new String [Number\_of\_nodes+countOfNew][Number\_of\_nodes+countOfNew];

newDESOfNodes=new String[Number\_of\_nodes+countOfNew];

newDESofAss=new String [Number\_of\_nodes+countOfNew][Number\_of\_nodes+countOfNew];

for (int i=0;i<Number\_of\_nodes;i++){

newnamesOfNodes[i]=namesOfNodes[i];

newDESOfNodes[i]=descriptionNotes[i];

}

for (int i=Number\_of\_nodes;i<Number\_of\_nodes+countOfNew;i++){

if(i==Number\_of\_nodes){

newnamesOfNodes[i]=enteringString("\nВведите имена узлов.");

newDESOfNodes[i]="";}

else{

newnamesOfNodes[i]=enteringString("\nВведите имена узлов.");

}

}

for (int i=0;i<Number\_of\_nodes;i++){

for (int k=0;k<Number\_of\_nodes;k++){

newassociations[i][k]=associations[i][k];

newDESofAss[i][k]=descriptionAssociations[i][k];

}

}

for (int i=0;i<Number\_of\_nodes+countOfNew;i++) {

for (int k=0;k<Number\_of\_nodes+countOfNew;k++){

if (i!=k){

if(newassociations[i][k]==null){

newassociations[i][k]=enteringString("\nВведите связь из "+newnamesOfNodes[i]+" в "+newnamesOfNodes[k]+", если связи нет введите нуль.");

newDESofAss[i][k]="";

if ("0".equals(newassociations[i][k])){

newassociations[i][k]="";

System.out.println("Вы ввели "+newassociations[i][k]);

}

}

}

}

}

namesOfNodes = new String[Number\_of\_nodes+countOfNew ];

associations=new String[Number\_of\_nodes+countOfNew][Number\_of\_nodes+countOfNew];

descriptionNotes=new String[Number\_of\_nodes+countOfNew ];

descriptionAssociations=new String[Number\_of\_nodes+countOfNew][Number\_of\_nodes+countOfNew];

set\_long\_namesOfNodes(Number\_of\_nodes+countOfNew );

set\_long\_associations(Number\_of\_nodes+countOfNew);

set\_long\_descriptionAssociations(Number\_of\_nodes+countOfNew);

set\_long\_descriptionNotes(Number\_of\_nodes+countOfNew);

Number\_of\_nodes=Number\_of\_nodes+countOfNew; //

associations=newassociations; //

namesOfNodes=newnamesOfNodes;

descriptionNotes=newDESOfNodes;

descriptionAssociations=newDESofAss;

set\_Number\_of\_nodes(Number\_of\_nodes);

set\_associations(associations);

set\_namesOfNodes(namesOfNodes);

set\_descriptionNotes(descriptionNotes);

set\_descriptionAssociations(descriptionAssociations);

//////////////////////////////////////УДАЛЕНИЕ УЗЛА////////////////////////////////////////////////////

}

else{

if (Integer.parseInt (nowline)==1){

String nowLine;

boolean Cycle=false;

while (Cycle==false){

System.out.println("");

System.out.println("Введите индекс узла, который Вы хотите удалить.");

nowLine=in.nextLine();

if(itFits(nowLine,0,Number\_of\_nodes-1)==true){

deletingNode=Integer.parseInt (nowLine);

Cycle=true;

}

}

countOfNew= Number\_of\_nodes-1; //Новое количество узлов

newnamesOfNodes=new String[countOfNew];

newassociations=new String [countOfNew][countOfNew];

newDESOfNodes=new String[countOfNew];

newDESofAss=new String [countOfNew][countOfNew];

for (int i=0;i<deletingNode;i++){

newnamesOfNodes[i]=namesOfNodes[i];

newDESOfNodes[i]=descriptionNotes[i];

}

for (int i=deletingNode;i<countOfNew;i++) {

newnamesOfNodes[i]=namesOfNodes[i+1];

newDESOfNodes[i]=descriptionNotes[i+1];

}

for (int x=0;x<deletingNode;x++){

for (int y=0;y<deletingNode;y++){

newassociations[x][y]=associations[x][y];

newDESofAss[x][y]=descriptionAssociations[x][y];

}

}

for (int x=0;x<Number\_of\_nodes;x++){

for (int y=deletingNode;y<Number\_of\_nodes-1;y++){

associations[x][y]=associations[x][y+1];

descriptionAssociations[x][y]=descriptionAssociations[x][y+1];

}

}

for (int y=0;y<Number\_of\_nodes;y++){

for (int x=deletingNode;x<Number\_of\_nodes-1;x++){

associations[x][y]=associations[x+1][y];

descriptionAssociations[x][y]=descriptionAssociations[x+1][y];

}

}

for (int x=0;x<Number\_of\_nodes-1;x++){

for (int y=0;y<Number\_of\_nodes-1;y++){

newassociations[x][y]=associations[x][y];

newDESofAss[x][y]=descriptionAssociations[x][y];

}

}

namesOfNodes = new String[Number\_of\_nodes-1];

associations=new String[Number\_of\_nodes-1][Number\_of\_nodes-1];

descriptionNotes=new String[Number\_of\_nodes-1 ];

descriptionAssociations=new String[Number\_of\_nodes-1][Number\_of\_nodes-1];

set\_long\_namesOfNodes(Number\_of\_nodes+countOfNew );

set\_long\_associations(Number\_of\_nodes+countOfNew);

set\_long\_descriptionAssociations(Number\_of\_nodes+countOfNew);

set\_long\_descriptionNotes(Number\_of\_nodes+countOfNew);

associations=newassociations;

namesOfNodes=newnamesOfNodes;

descriptionNotes=newDESOfNodes;

descriptionAssociations=newDESofAss;

Number\_of\_nodes=Number\_of\_nodes-1;

set\_Number\_of\_nodes(Number\_of\_nodes);

set\_associations(associations);

set\_namesOfNodes(namesOfNodes);

set\_descriptionNotes(descriptionNotes);

set\_descriptionAssociations(descriptionAssociations);

////////////////////////////////////////////////////////ВЫЙТИ////////////////////////////////////////////////////////////////////////////////////

}

else{

if (Integer.parseInt (nowline)==2){

break;

}

}

}

}

}

}

}

@Override

public void AddDeleteEdge() {

if (get\_graph()==false) {

System.out.println("");

System.out.println((char) 27 + "[31mСперва введите граф! " + (char)27 + "[0m");}

else {

Scanner in = new Scanner(System.in);

LoockInd();

boolean cycle1=true;

boolean cycle2=true;

while (cycle1==true){

System.out.println("\n"+Number\_of\_nodes+"- Показать индексы. \n"+(Number\_of\_nodes+1)+"- Выйти \nВведите индекс узла источника.");

String nowLine1=in.nextLine();

if ((itFits(nowLine1,0,(Number\_of\_nodes+1))&& (isNumeric(nowLine1)==true))){

if (Integer.parseInt (nowLine1)==Number\_of\_nodes) { LoockInd();}

if (Integer.parseInt (nowLine1)==Number\_of\_nodes+1) {cycle1=false;}

if ((Integer.parseInt (nowLine1)!=Number\_of\_nodes) && (Integer.parseInt (nowLine1)!=Number\_of\_nodes+1)) {

System.out.println("Первый индекс принят.");

while (cycle2==true) {

System.out.println("");

System.out.println("Введите индекс узла приемника.");

String nowLine2=in.nextLine();

if (itFits(nowLine2,0,Number\_of\_nodes-1)==true){

if (Integer.parseInt (nowLine2)!=Integer.parseInt (nowLine1)) {

String nowLine3=enteringString("\nВведите связь, если ввести 0, связи не будет.");

if ("0".equals(nowLine3)) {

associations[Integer.parseInt (nowLine1)][Integer.parseInt (nowLine2)]="";

cycle1=false;

cycle2=false;

}

else {associations[Integer.parseInt (nowLine1)][Integer.parseInt (nowLine2)]=nowLine3;cycle1=false;

cycle2=false;}

}

else

{

System.out.println("");

System.out.println((char) 27 + "[31mИндексы не должны совпадать! " + (char)27 + "[0m");

}

}

}

}

}

}

set\_associations(associations);

}

}

@Override

public void Change() {

if (get\_graph()==false) {

System.out.println("");

System.out.println((char) 27 + "[31mСперва введите граф! " + (char)27 + "[0m");}

else {

Scanner in = new Scanner(System.in);

System.out.print("\n");

int Z;

boolean CH=false;

String nowLine;

LoockInd();

while (CH==false)

{

System.out.println("");

System.out.println("Что Вы хотите изменить? 0-узел 1-ребро\n2- Если вы хотите увидеть индексы. \n3- Выйти. ");

nowLine=in.nextLine();

if (itFits(nowLine,0,3)){

Z=Integer.parseInt (nowLine);

switch(Z)

{

case (0):

boolean Cycle=false;

while (Cycle==false){

System.out.println("");

System.out.println("Введите индекс, того узла, чье имя вы хотите изменить.");

nowLine=in.nextLine();

if (itFits(nowLine, 0, (Number\_of\_nodes-1))==true){

namesOfNodes[Integer.parseInt (nowLine)]=enteringString("\nВведите новое имя узла.");

Cycle=true;

}

}

set\_namesOfNodes(namesOfNodes);

break;

case (1):

String nowLine1;

boolean Cycle1=false;

String nowLine2;

while (Cycle1==false){

System.out.println("Введите индекс узла источника.");

nowLine1=in.nextLine();

if (HasPath(nowLine1,0,Number\_of\_nodes-1)==true){

System.out.println("Первый индекс принят.");

while (Cycle1==false){

System.out.println("Введите индекс узла приемника.");

nowLine2=in.nextLine();

if (itFits(nowLine2,0,(Number\_of\_nodes-1))==true){

if (Integer.parseInt (nowLine2)!=Integer.parseInt (nowLine1)){

if (!"".equals(associations[Integer.parseInt (nowLine1)][Integer.parseInt (nowLine2)])){

associations[Integer.parseInt (nowLine1)][Integer.parseInt (nowLine2)]=enteringString("\nВведите новое значение связи.");

Cycle1=true;

}

}

else {

System.out.println("");

System.out.println((char) 27 + "[31mИндексы не должны совпадать! " + (char)27 + "[0m");

}

}

}

}

}

set\_associations(associations);

break;

case (2):

LoockInd();

break;

case (3):

CH=true;

break;

}

}

}

}

}

@Override

public void Print() {

if (get\_graph()==false) {

System.out.println("");

System.out.println((char) 27 + "[31mСперва введите граф! " + (char)27 + "[0m");}

else {

boolean ways[]=new boolean[Number\_of\_nodes];

for (int x=0;x<Number\_of\_nodes;x++){

for (int y=0;y<Number\_of\_nodes;y++){

if (x!=y){

if (!"".equals(associations[x][y])){

ways[x]=true;

}

}

}

}

for (int x=0;x<Number\_of\_nodes;x++){

if (ways[x]==true){

for (int y=0;y<Number\_of\_nodes;y++){

if (x!=y){

if (!"".equals(associations[x][y])){

System.out.println(namesOfNodes[x]+"-- "+associations[x][y]+" -->"+namesOfNodes[y]);

}

}

}

}

else {

System.out.println(namesOfNodes[x]+" никуда не ведет.");

}

}

}

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

package course.paper;

import static course.paper.UsefulFunctions.\*;

import java.util.Scanner;

import static products.Description.\*;

import static products.Graph.\*;

public class ConcreteDescriptionBuilder implements GraphBuilder{

private static int Number\_of\_nodes;

private static String namesOfNodes[];

private static String associations[][];

private static String descriptionNotes[];

private static String descriptionAssociations[][];

@Override

public void Enter() {

if(get\_graph()==false){

System.out.println("");

System.out.println((char) 27 + "[31mСперва введите граф! " + (char)27 + "[0m");

}

else {

set\_description(true);

Number\_of\_nodes=get\_Number\_of\_nodes();

namesOfNodes=get\_namesOfNodes();

associations=get\_associations();

descriptionNotes=get\_descriptionNotes();

descriptionAssociations=get\_descriptionAssociations();

Scanner in = new Scanner(System.in);

for (int k=0;k<Number\_of\_nodes;k++)

{if(k<Number\_of\_nodes){

descriptionNotes[k]=enteringString("\nВведите описание узла "+namesOfNodes[k]);

}

}

for (int i=0;i<Number\_of\_nodes;i++)

{

for (int j=0;j<Number\_of\_nodes;j++)

{

if (i!=j){

if (!"".equals(associations[i][j])) {

descriptionAssociations[i][j]=enteringString("\nВведите описание ребра между "+namesOfNodes[i]+" и "+namesOfNodes[j]+" ");

}

}

}

}

System.out.println("");

set\_descriptionNotes(descriptionNotes);

set\_descriptionAssociations(descriptionAssociations);

}

}

@Override

public void AddDeleteNote() {

if (get\_graph()==false) {

System.out.println("");

System.out.println((char) 27 + "[31mСперва введите граф! " + (char)27 + "[0m");

}

else{

Number\_of\_nodes=get\_Number\_of\_nodes();

namesOfNodes=get\_namesOfNodes();

associations=get\_associations();

descriptionNotes=get\_descriptionNotes();

descriptionAssociations=get\_descriptionAssociations();

Scanner in = new Scanner(System.in);

boolean cycle1=true;

String nowline;

LoockInd();

while(cycle1==true){

System.out.println("");

System.out.println("Введите индекс узла, чье описание вы хотите изменить или удалить."+Number\_of\_nodes+"- показать индексы, "+(Number\_of\_nodes+1)+"- выйти.");

nowline=in.nextLine();

if (itFits(nowline,0,(Number\_of\_nodes+1))==true){

if (Integer.parseInt(nowline)==Number\_of\_nodes){

LoockInd();

}

if (Integer.parseInt(nowline)==Number\_of\_nodes+1){

break;

}

if ((Integer.parseInt(nowline)>=0) && (Integer.parseInt(nowline)<Number\_of\_nodes)){

boolean cycle2=true;

while (cycle2==true){

String nowline1=enteringString("\nВведите новое описание узла "+namesOfNodes[Integer.parseInt(nowline)]+", если хотите удалить описание введите нуль.");

if (notEmpty(nowline1)==true){

if ("0".equals(nowline1)){

descriptionNotes[Integer.parseInt(nowline)]="";

cycle2=false;

cycle1=false;

}

else {

descriptionNotes[Integer.parseInt(nowline)]=nowline1;

cycle2=false;

cycle1=false;

set\_description(true);

}

}

}

}

}

}

set\_descriptionNotes(descriptionNotes);

}

}

@Override

public void AddDeleteEdge() {

if (get\_graph()==false) {

System.out.println("");

System.out.println((char) 27 + "[31mСперва введите граф! " + (char)27 + "[0m");

}

else{

Number\_of\_nodes=get\_Number\_of\_nodes();

namesOfNodes=get\_namesOfNodes();

associations=get\_associations();

descriptionNotes=get\_descriptionNotes();

descriptionAssociations=get\_descriptionAssociations();

Scanner in = new Scanner(System.in);

LoockInd();

boolean cycle1=true;

boolean cycle2=true;

while (cycle1==true){

System.out.println("\n"+Number\_of\_nodes+"- Показать индексы. \n"+(Number\_of\_nodes+1)+"- Выйти \nВведите индекс узла источника.");

String nowLine1=in.nextLine();

if ((itFits(nowLine1,0,(Number\_of\_nodes+1)))){

if (Integer.parseInt (nowLine1)==Number\_of\_nodes) { LoockInd();}

if (Integer.parseInt (nowLine1)==Number\_of\_nodes+1) {cycle1=false;}

if ((Integer.parseInt (nowLine1)!=Number\_of\_nodes) && (Integer.parseInt (nowLine1)!=Number\_of\_nodes+1)&& (HasPath(nowLine1,0,Number\_of\_nodes-1))) {

System.out.println("Первый индекс принят.");

while (cycle2==true) {

System.out.println("");

System.out.println("Введите индекс узла приемника.");

String nowLine2=in.nextLine();

if (itFits(nowLine2,0,Number\_of\_nodes-1)==true){

if (Integer.parseInt (nowLine2)!=Integer.parseInt(nowLine1)) {

String nowLine3=enteringString("\nВведите описание связи, если ввести 0, описания не будет не будет.");

if ("0".equals(nowLine3)) {

descriptionAssociations[Integer.parseInt (nowLine1)][Integer.parseInt (nowLine2)]="";

cycle1=false;

cycle2=false;

}

else {

descriptionAssociations[Integer.parseInt (nowLine1)][Integer.parseInt (nowLine2)]=nowLine3;cycle1=false;

cycle2=false;

set\_description(true);

}

}

else

{

System.out.println("");

System.out.println((char) 27 + "[31mИндексы не должны совпадать! " + (char)27 + "[0m");

}

}

}

}

set\_descriptionAssociations(descriptionAssociations);

}

}

}

}

@Override

public void Change() {

}

@Override

public void Print() {

if ((get\_description()==false) || (get\_graph()==false)){

if (get\_graph()==false){

System.out.println("");

System.out.println((char) 27 + "[31mСперва введите граф! " + (char)27 + "[0m");

}

if ((get\_graph()==true)&&(get\_description()==false)){

System.out.println("");

System.out.println((char) 27 + "[31mУ вас нет ни одного описания! " + (char)27 + "[0m");

}

}

else{

Number\_of\_nodes=get\_Number\_of\_nodes();

namesOfNodes=get\_namesOfNodes();

associations=get\_associations();

descriptionNotes=get\_descriptionNotes();

descriptionAssociations=get\_descriptionAssociations();

System.out.print("\n");

for (int k=0;k<Number\_of\_nodes;k++)

{

if(k<Number\_of\_nodes)

{

if ("".equals(descriptionNotes[k]))

{

System.out.println("Узел "+namesOfNodes[k]+": нет информации");

}

else

{

System.out.println("Узел "+namesOfNodes[k]+": "+descriptionNotes[k]);

}

}

}

for (int i=0;i<Number\_of\_nodes;i++)

{

for (int j=0;j<Number\_of\_nodes;j++)

{

if (i!=j){

if (!"".equals(associations[i][j])) {

if (!"".equals(descriptionAssociations[i][j]))

{

System.out.println("Описание ребра между "+namesOfNodes[i]+" и "+namesOfNodes[j]+": "+descriptionAssociations[i][j]);

}

else

{

System.out.println("Описание ребра между "+namesOfNodes[i]+" и "+namesOfNodes[j]+": нет информации");

}

}

}

}

}

}

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**package** products;

**public** **class** Description {

//Поля описаний.

**private** **static** **boolean** *description*=**false**;

**private** **static** String *descriptionNotes*[];

**private** **static** String *descriptionAssociations*[][];

//Сеттеры и геттеры для доступа к полям этого класса.

**public** **static** **boolean** get\_description(){

**return** *description*;

}

**public** **static** **void** set\_description(**boolean** a){

*description*=a;

}

**public** **static** **void** set\_long\_descriptionAssociations (**int** a){

*descriptionAssociations*=**new** String [a][a];

}

**public** **static** **void** set\_descriptionAssociations (String[][] a){

*descriptionAssociations*=a;

}

**public** **static** String[][] get\_descriptionAssociations(){

**return** *descriptionAssociations*;

}

**public** **static** **void** set\_one\_descriptionAssociations(String a,**int** b, **int** c){

*descriptionAssociations*[b][c]=a;

}

**public** **static** String get\_one\_descriptionAssociations(**int** a, **int** b){

**return** *descriptionAssociations*[a][b];

}

**public** **static** **void** set\_long\_descriptionNotes(**int** a){

*descriptionNotes*=**new** String [a];

}

**public** **static** **void** set\_descriptionNotes(String[] a){

*descriptionNotes*=a;

}

**public** **static** String[] get\_descriptionNotes(){

**return** *descriptionNotes*;

}

**public** **static** **void** set\_one\_descriptionNotes (String a,**int** b){

*descriptionNotes*[b]=a;

}

**public** **static** String get\_one\_descriptionNotes(**int** a){

**return** *descriptionNotes*[a];

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**package** products;

**public** **class** Graph {

//Поля графа.

**private** **static** **int** *Number\_of\_nodes*;

**private** **static** String *associations*[][];

**private** **static** String *namesOfNodes*[];

**private** **static** **boolean** *graph*=**false**;

//Сеттеры и геттеры для доступа к полям этого класса.

**public** **static** **boolean** get\_graph(){

**return** *graph*;

}

**public** **static** **void** set\_graph(**boolean** a){

*graph*=a;

}

**public** **static** **void** set\_Number\_of\_nodes (**int** a){

*Number\_of\_nodes*=a;

}

**public** **static** **int** get\_Number\_of\_nodes (){

**return** *Number\_of\_nodes*;

}

**public** **static** **void** set\_long\_associations (**int** a){

*associations*=**new** String [a][a];

}

**public** **static** **void** set\_associations (String[][] a){

*associations*=a;

}

**public** **static** String[][] get\_associations(){

**return** *associations*;

}

**public** **static** **void** set\_long\_namesOfNodes(**int** a){

*namesOfNodes*=**new** String [a];

}

**public** **static** **void** set\_namesOfNodes(String[] a){

*namesOfNodes*=a;

}

**public** **static** String[] get\_namesOfNodes(){

**return** *namesOfNodes*;

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**package** products;

**public** **class** TheAdjacencyMatrix {

//Поля матрицы смежности.

**private** **static** **int** *maxLongAdj*[];

**private** **static** String *Adj*[][];

**private** **static** **int** *justLongAdj*[][];

**private** **static** **int** *IndAdj*[][];

**private** **static** String *ProbAdj*[][];

//Сеттеры и геттеры для доступа к полям этого класса.

**public** **static** **void** set\_long\_maxLongAdj (**int** a){

*maxLongAdj*=**new** **int** [a];

}

**public** **static** **void** set\_maxLongAdj (**int**[] a){

*maxLongAdj*=a;

}

**public** **static** **int**[] get\_maxLongAdj(){

**return** *maxLongAdj*;

}

**public** **static** **void** set\_long\_Adj(**int** a){

*Adj*=**new** String [a][a];

}

**public** **static** **void** set\_Adj(String[][] a){

*Adj*=a;

}

**public** **static** String[][] get\_namesOfNodes(){

**return** *Adj*;

}

**public** **static** **void** set\_long\_justLongAdj(**int** a){

*justLongAdj*=**new** **int** [a][a];

}

**public** **static** **void** set\_justLongAdj(**int**[][] a){

*justLongAdj*=a;

}

**public** **static** **int**[][] get\_justLongAdj(){

**return** *justLongAdj*;

}

**public** **static** **void** set\_long\_IndAdj(**int** a){

*IndAdj*=**new** **int** [a][a];

}

**public** **static** **void** set\_IndAdj(**int**[][] a){

*IndAdj*=a;

}

**public** **static** **int**[][] get\_IndAdj(){

**return** *IndAdj*;

}

**public** **static** **void** set\_long\_ProbAdj(**int** a){

*ProbAdj*=**new** String [a][a];

}

**public** **static** **void** set\_ProbAdj(String[][] a){

*ProbAdj*=a;

}

**public** **static** String[][] get\_ProbAdj(){

**return** *ProbAdj*;

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**package** products;

**public** **class** TheincidenceMatrix {

//Поля матрицы инцидентности.

**private** **static** **int** *Connection*;

**private** **static** **int** *maxLongInc*[];

**private** **static** String *Inc*[][];

**private** **static** **int** *justLongInc*[][];

**private** **static** **int** *IndInc*[][];

**private** **static** String *ProbInc*[][];

//Сеттеры и геттеры для доступа к полям этого класса.

**public** **static** **void** set\_Connection(**int** a){

*Connection*=a;

}

**public** **static** **int** get\_Connection(){

**return** *Connection*;

}

**public** **static** **void** set\_long\_maxLongInc (**int** a){

*maxLongInc*=**new** **int** [a];

}

**public** **static** **void** set\_maxLongInc (**int**[] a){

*maxLongInc*=a;

}

**public** **static** **int**[] get\_maxLongInc(){

**return** *maxLongInc*;

}

**public** **static** **void** set\_long\_Inc(**int** a){

*Inc*=**new** String [a][a];

}

**public** **static** **void** set\_Inc(String[][] a){

*Inc*=a;

}

**public** **static** String[][] get\_Inc(){

**return** *Inc*;

}

**public** **static** **void** set\_long\_justLongInc(**int** a){

*justLongInc*=**new** **int** [a][a];

}

**public** **static** **void** set\_justLongInc(**int**[][] a){

*justLongInc*=a;

}

**public** **static** **int**[][] get\_justLongInc(){

**return** *justLongInc*;

}

**public** **static** **void** set\_long\_IndInc(**int** a){

*IndInc*=**new** **int** [a][a];

}

**public** **static** **void** set\_IndInc(**int**[][] a){

*IndInc*=a;

}

**public** **static** **int**[][] get\_IndInc(){

**return** *IndInc*;

}

**public** **static** **void** set\_long\_ProbInc(**int** a){

*ProbInc*=**new** String [a][a];

}

**public** **static** **void** set\_ProbInc(String[][] a){

*ProbInc*=a;

}

**public** **static** String[][] get\_ProbInc(){

**return** *ProbInc*;

}

}

# Приложение 2. Руководство пользователя консольного приложения