**Лабораторная работа №6**

**Основы программирования на Java: модификаторы доступа и класса**

**Цель работы:** Освоить основные способы создания Java-программ либо с помощью обычного редактора, либо с помощью среды разработки. Приобрести навыки работы с модификаторами доступа в Java.

**Теоретические сведения**



**Модификаторы** — [ключевые слова](http://proglang.su/java/syntax#klyuchevye-slova-v-java), которые Вы добавляете при инициализации для изменения значений. Язык Java имеет широкий спектр модификаторов, основные из них:

* модификаторы доступа;
* модификаторы класса, метода, переменной и потока, используемые не для доступа.

Чтобы использовать модификатор в Java, нужно включить его ключевое слово в определение класса, метода или переменной. Модификатор должен быть впереди остальной части оператора, как показано в следующих примерах:

public class className {

// ...

}

private boolean myFlag;

static final double weeks = 9.5;

protected static final int BOXWIDTH = 42;

public static void main(String[] arguments) {

// тело метода

}

**Модификаторы доступа**

Java предоставляет ряд модификаторов доступа, чтобы задать уровни доступа для классов, переменных, методов и конструкторов. Существует четыре доступа:

* Видимый в пакете (стоит по умолчанию и модификатор не требуются).
* Видимый только для класса (private).
* Видимый для всех (public).
* Видимый для пакета и всех подклассов (protected).

**Модификатор доступа по умолчанию — без ключевого слова**

**Модификатор доступа по умолчанию** — означает, что мы явно не объявляем модификатор доступа в Java для класса, поля, метода и т.д.

Переменная или метод, объявленные без модификатора контроля доступа доступны для любого другого класса в том же пакете. Поля в интерфейсе неявно являются public, static, final, а методы в интерфейсе по умолчанию являются public.

**Пример**

Переменные и методы могут быть объявлены в Java без каких-либо модификаторов, как показано в следующем примере:

String version = "1.5.1";

boolean processOrder() {

return true;

}

**Модификатор доступа private**

**Модификатор private** — методы, переменные и конструкторы, которые объявлены как private в Java могут быть доступны только в пределах самого объявленного класса.

Модификатор доступа private является наиболее ограничивающим уровенем доступа. Класс и интерфейсы не могут быть private.

Переменные, объявленные как private, могут быть доступны вне класса, если получающие их открытые (public) методы присутствуют в классе (ниже смотрите пример и пояснения).

Использование модификатора private в Java является основным способом, чтобы скрыть данные.

**Пример**

Следующий класс использует контроль доступа private:

public class Logger {

private String format;

public String getFormat() {

return this.format;

}

public void setFormat(String format) {

this.format = format;

}

}

Здесь переменная *format* класса *Logger* является private, так что нет никакого способа для других классов, чтобы получить и установить её значение напрямую.

Таким образом, чтобы эта переменная была доступна для всего, мы определили два открытых (public) метода: *getFormat()*, который возвращает значение *format*, и *setFormat(String)*, который устанавливает её значение.

**Модификатор доступа public**

**Модификатор public** — класс, метод, конструктор, интерфейс и т.д. объявленные как public могут быть доступны из любого другого класса. Поэтому поля, методы, блоки, объявленные внутри public класса могут быть доступны из любого класса, принадлежащего к "вселенной" Java.

Тем не менее, если к public классу в другом пакете мы пытаемся получить доступ, то public класс приходится импортировать.

Благодаря наследованию классов, в Java все публичные (public) методы и переменные класса наследуются его подклассами.

**Пример**

Следующая функция использует контроль доступа public:

public static void main(String[] arguments) {

// ...

}

Метод *main()* должен быть публичным (public). В противном случае, он не может быть вызван с помощью java-интерпретатора, чтобы запустить класс.

**Модификатор доступа protected**

**Модификатор protected** — переменные, методы и конструкторы, которые объявляются как protected в суперклассе, могут быть доступны только для подклассов в другом пакете или для любого класса в пакете класса protected.

Модификатор доступа protected в Java не может быть применен к классу и интерфейсам. Методы и поля могут быть объявлены как protected, однако методы и поля в интерфейсе не могут быть объявлены как protected.

Доступ protected дает подклассу возможность использовать вспомогательный метод или переменную, предотвращая неродственный класс от попыток использовать их.

**Пример**

Следующий родительский класс использует контроля доступа protected, чтобы его дочерний класс переопределил метод *openSpeaker()*:

class AudioPlayer {

protected boolean openSpeaker(Speaker sp) {

// детали реализации

}

}

class StreamingAudioPlayer {

boolean openSpeaker(Speaker sp) {

// детали реализации

}

}

При этом, если мы определим метод *openSpeaker()* как protected, то он не будет доступен из любого другого класса, кроме AudioPlayer. Если мы определим его как public, то он станет доступным всем. Но наше намерение состоит в том, чтобы раскрыть этот метод только подклассу, вот почему мы использовали модификатор protected.

**Правила контроля доступа и наследования**

Следующие правила в Java применяются для унаследованных методов:

* Методы, объявленные как public в суперклассе, также должны быть public во всех подклассах.
* Методы, объявленные как protected в суперклассе, должны либо быть либо protected, либо public в подклассах; они не могут быть private.
* Методы, объявленные как private для всех не наследуются, так что нет никакого правила для них.

**Модификаторы класса, метода, переменной и потока, используемые не для доступа**

Java предоставляет ряд модификаторов не для доступа, а для реализации многих других функциональных возможностей:

* модификатор *static* применяется для создания методов и переменных класса;
* модификатор *final* используется для завершения реализации классов, методов и переменных;
* модификатор *abstract* необходим для создания абстрактных классов и методов;
* модификаторы *synchronized* и *volatile* используются в Java для потоков.

**Модификатор static**

**Модификатор static** — применяется для создания методов и переменных класса.

**Переменные static**

Ключевое слово static используется для создания переменных, которые будут существовать независимо от каких-либо экземпляров, созданных для класса. Только одна копия переменной static в Java существует вне зависимости от количества экземпляров класса.

Статические переменные также известны как переменные класса. В Java локальные переменные не могут быть объявлены статическими (static).

**Методы static**

Ключевое слово static используется для создания методов, которые будут существовать независимо от каких-либо экземпляров, созданных для класса.

В Java статические методы или методы static не используют какие-либо переменные экземпляра любого объекта класса, они определены. Методы static принимают все данные из параметров и что-то из этих параметров вычисляется без ссылки на переменные.

Переменные и методы класса могут быть доступны с использованием имени класса, за которым следует точка и имя переменной или метода.

**Пример**

Модификатор static в Java используется для создания методов классов и переменных, как показано в следующем примере:

public class InstanceCounter {

private static int numInstances = 0;

protected static int getCount() {

return numInstances;

}

private static void addInstance() {

numInstances++;

}

InstanceCounter() {

InstanceCounter.addInstance();

}

public static void main(String[] arguments) {

System.out.println("Начиная с " +

InstanceCounter.getCount() + " экземпляра");

for (int i = 0; i < 500; ++i){

new InstanceCounter();

}

System.out.println("Создано " +

InstanceCounter.getCount() + " экземпляров");

}

}

Будет получен следующий результат:

Начиная с 0 экземпляра

Создано 500 экземпляров

**Модификатор final**

**Модификатор final** — используется для завершения реализации классов, методов и переменных.

**Переменные final**

Переменная final может быть инициализирована только один раз. Ссылочная переменная, объявленная как final, никогда не может быть назначен для обозначения другого объекта.

Однако данные внутри объекта могут быть изменены. Таким образом, состояние объекта может быть изменено, но не ссылки.

С переменными в Java модификатор final часто используется со static, чтобы сделать константой переменную класса.

**Пример**

public class Test{

final int value = 10;

// Ниже приведены примеры объявления констант:

public static final int BOXWIDTH = 6;

static final String TITLE = "Менеджер";

public void changeValue(){

value = 12; //будет получена ошибка

}

}

**Методы final**

Метод final не может быть переопределен любым подклассом. Как упоминалось ранее, в Java модификатор final предотвращает метод от изменений в подклассе.

Главным намерение сделать метод final будет то, что содержание метода не должно быть изменено стороне.

**Пример**

Объявление метода, использующего модификатор final в объявление класса, показано в следующем примере:

public class Test{

public final void changeName(){

// тело метода

}

}

**Класс final**

Основная цель в Java использования класса объявленного в качестве final заключается в предотвращении класс от быть подклассом. Если класс помечается как final, то ни один класс не может наследовать любую функцию из класса final.

**Пример**

public final class Test {

// тело класса

}

**Модификатор abstract**

**Модификатор abstract** — используется для создания абстрактных классов и методов.

**Класс abstract**

Класс abstract не может создать экземпляр. Если класс объявлен как abstract, то единственная цель для него быть расширенным.

Класс не может быть одновременно abstract и final, так как класс final не может быть расширенным. Если класс содержит абстрактные методы, то он должен быть объявлен как abstract. В противном случае будет сгенерирована ошибка компиляции.

Класс abstract может содержать как абстрактные методы, а также и обычные.

**Пример**

abstract class Caravan{

private double price;

private String model;

private String year;

public abstract void goFast(); //абстрактный метод

public abstract void changeColor();

}

**Метод abstract**

Метод abstract является методом, объявленным с любой реализацией. Тело метода (реализация) обеспечивается подклассом. Методы abstract никогда не могут быть final или strict.

Любой класс, который расширяет абстрактный класс должен реализовать все абстрактные методы суперкласса, если подкласс не является абстрактным классом.

Если класс в Java содержит один или несколько абстрактных методов, то класс должен быть объявлен как abstract. Абстрактный класс не обязан содержать абстрактные методы.

Абстрактный метод заканчивается точкой с запятой. Пример: public abstract sample();

**Пример**

public abstract class SuperClass{

abstract void m(); //абстрактный метод

}

class SubClass extends SuperClass{

// реализует абстрактный метод

void m(){

.........

}

}

**Модификатор synchronized**

**Модификатор synchronized** — используются в Java для потоков.

Ключевое слово synchronized используется для указания того, что метод может быть доступен только одним потоком одновременно. В Java модификатор synchronized может быть применен с любым из четырех модификаторов уровня доступа.

**Пример**

public synchronized void showDetails(){

.......

}

**Модификатор transient**

Переменная экземпляра отмеченная как transient указывает виртуальной машине Java (JVM), чтобы пропустить определённую переменную при сериализации объекта, содержащего её.

Этот модификатор включён в оператор, что создает переменную, предшествующего класса или типа данных переменной.

**Пример**

public transient int limit = 55; // не будет сохраняться

public int b; // будет сохраняться

**Модификатор volatile**

**Модификатор volatile** — используются в Java для потоков.

В Java модификатор volatile используется, чтобы позволить знать JVM, что поток доступа к переменной всегда должен объединять свою собственную копию переменной с главной копией в памяти.

Доступ к переменной volatile синхронизирует все кэшированные скопированные переменные в оперативной памяти. Volatile может быть применен только к переменным экземпляра, которые имеют тип объект или private. Ссылка на объект volatile может быть null.

**Пример**

public class MyRunnable implements Runnable{

private volatile boolean active;

public void run(){

active = true;

while (active){ // линия 1

// здесь какой-нибудь код

}

}

public void stop(){

active = false; // линия 2

}

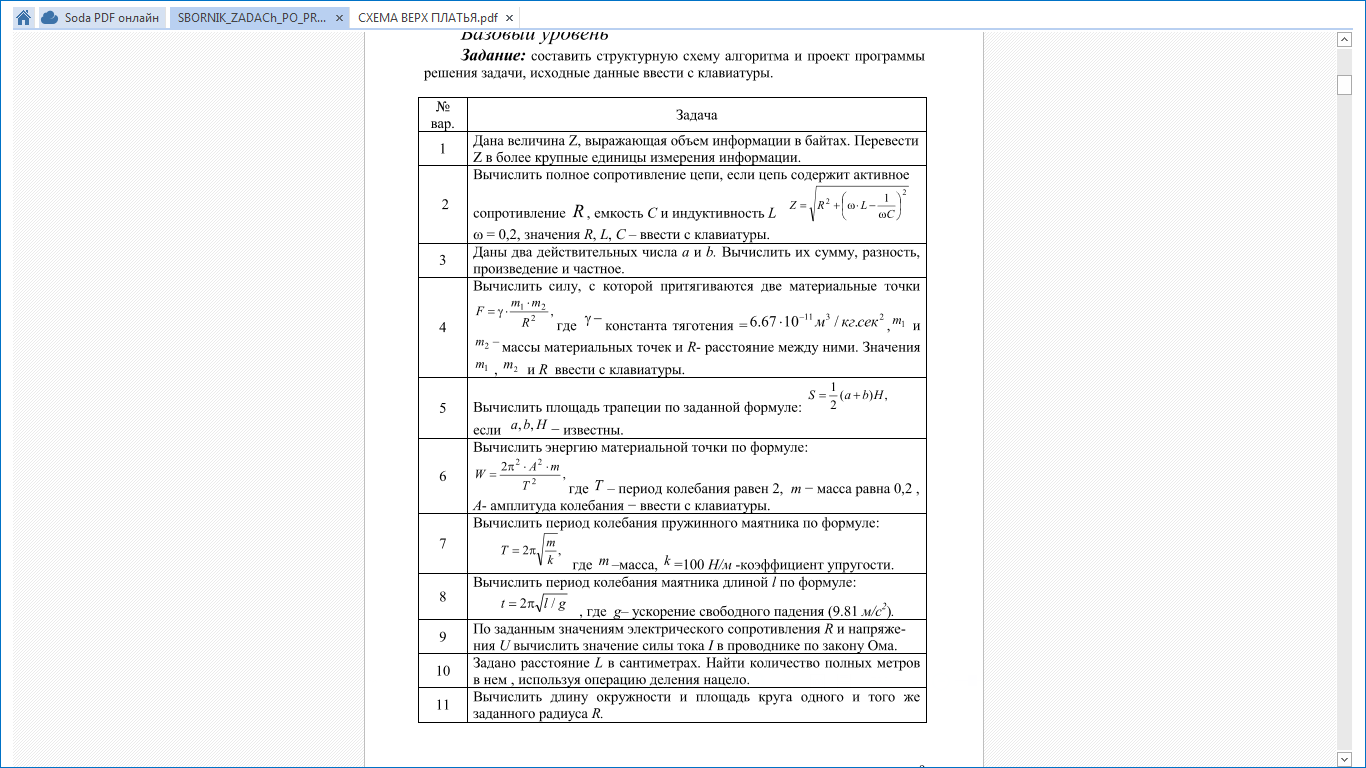
}

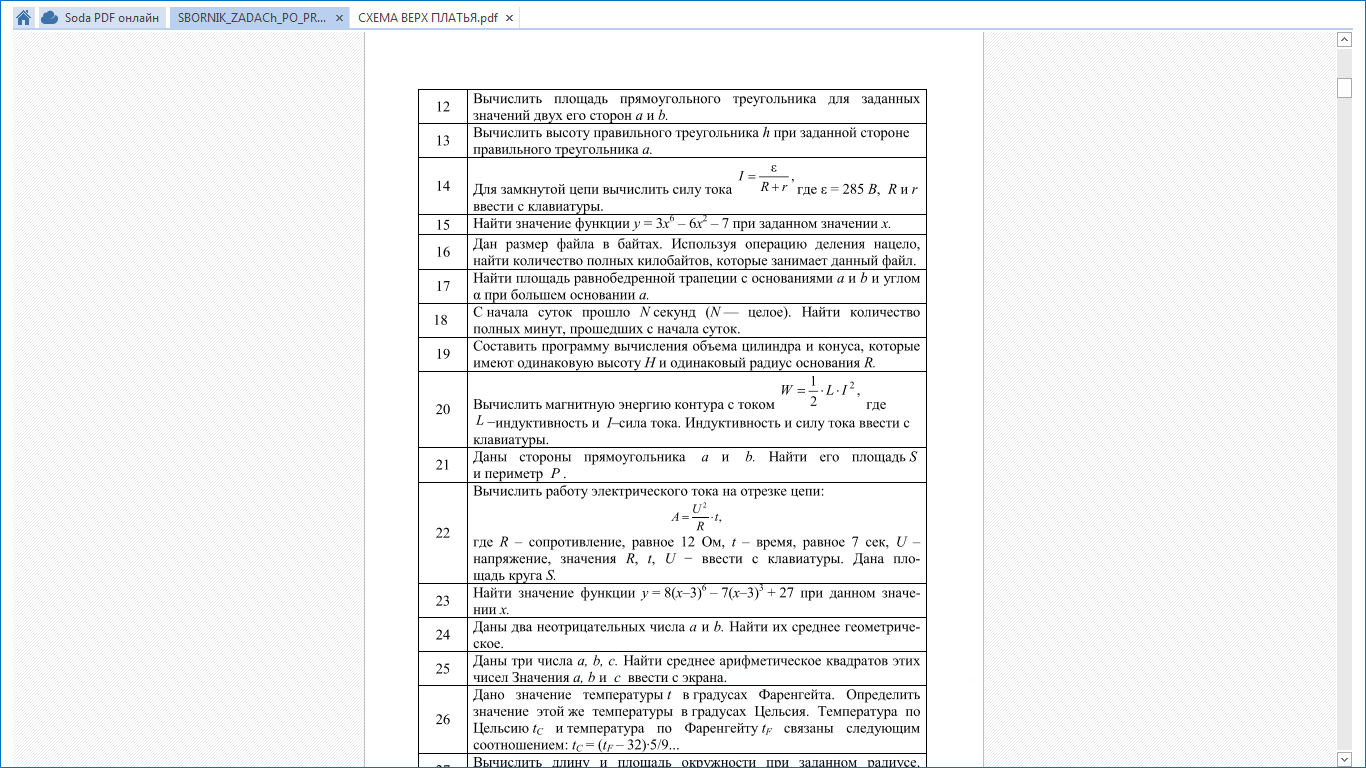
Как правило, run() вызывается в одном потоке (впервые начинаете использовать Runnable в Java), а stop() вызывается из другого потока. Если в линии 1 используется кэшированное значение active, то цикл не может остановиться, пока Вы не установите active false в линии 2.

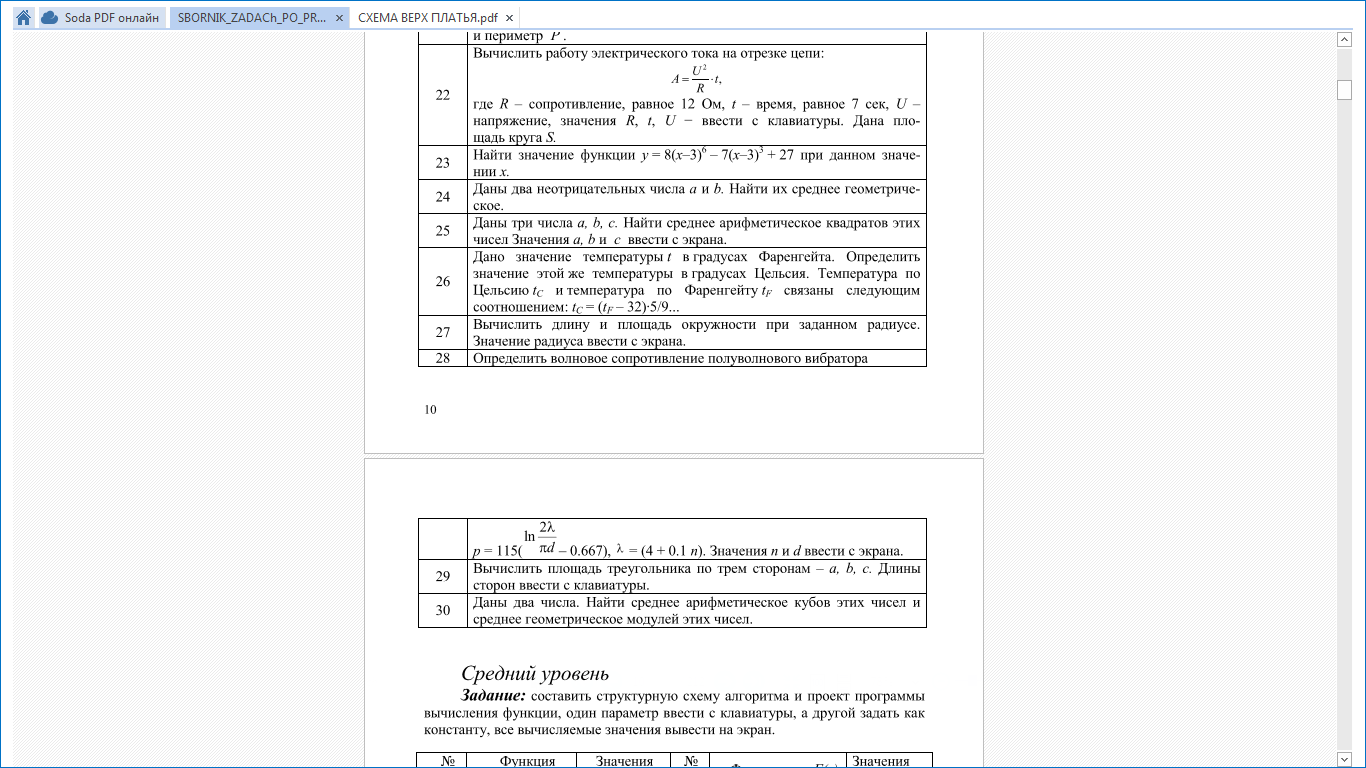
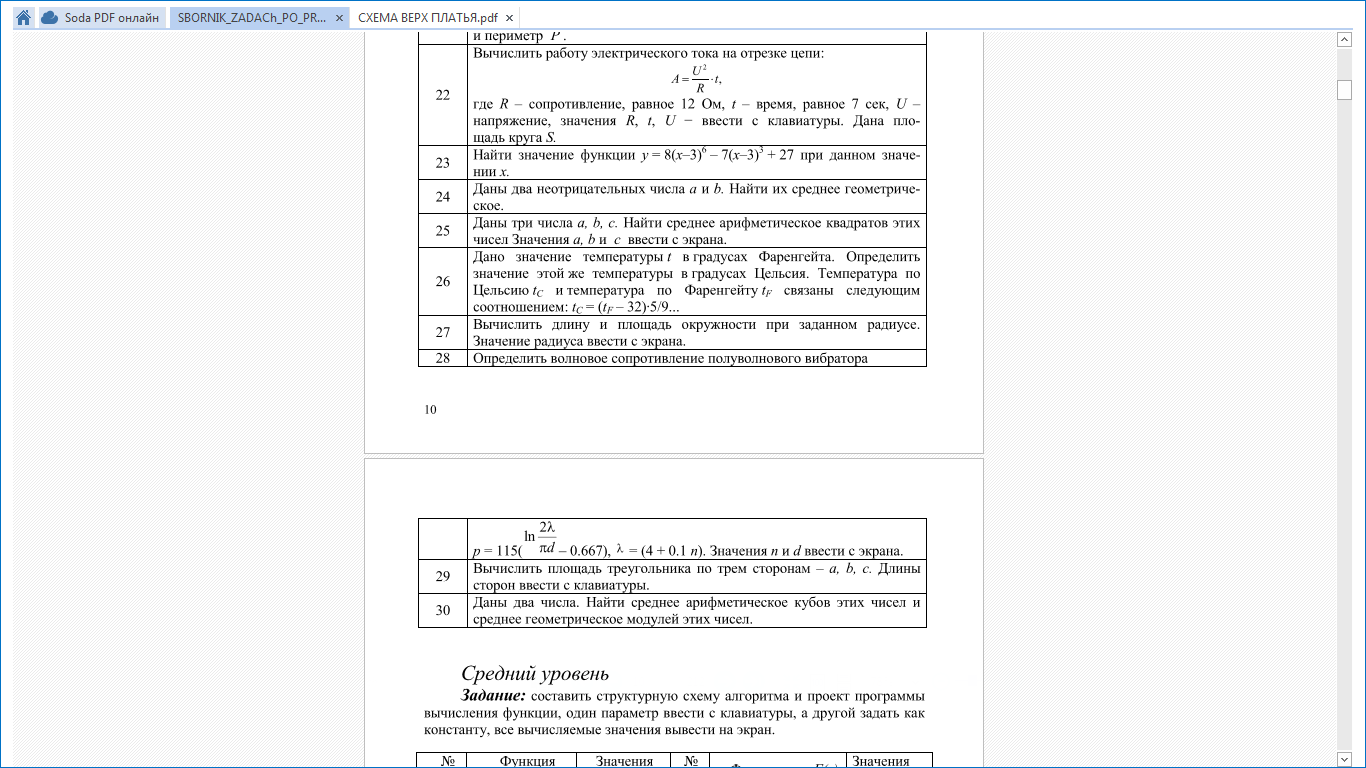
**Задание**

Изучить теоретический материал и выполнить задачи по вариантам.

Составить блок-схему алгоритма и проект программы решения задачи, исходные данные ввести с клавиатуры.





**Форма отчета**

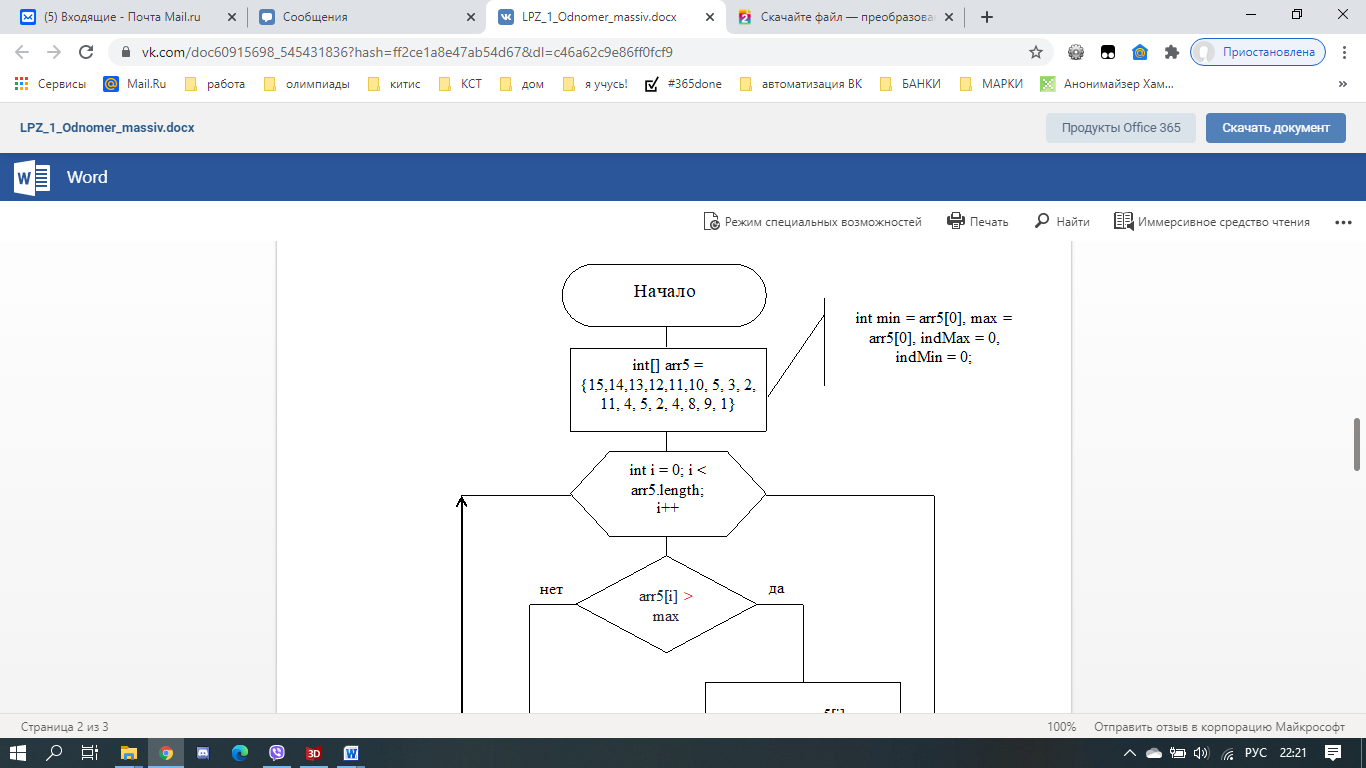
1. Тема, цель, индивидуальное задание

2. Таблица идентификаторов

|  |  |  |
| --- | --- | --- |
| Наименование переменной | Тип данных | Назначение |
|  |  |  |
|  |  |  |
|  |  |  |

3. Блок-схема алгоритма (см. Приложение)

Пример:



4. Код программы

5. Скриншот результата

**Контрольные вопросы для защиты**

1. Что такое модификатор?
2. Какие модификаторы есть в Java. Примеры.
3. Модификатор доступа private, public, protected.
4. Вспомогательные модификаторы в Java. Примеры.
5. Какие модификаторы вы использовали в работе?

**Приложение**

**Условные графические обозначения в схемах алгоритмов и программ, отображающие основные операции процесса обработки данных и программирования по ГОСТ 2.708 - 81**

Размер, а следует выбирать из ряда 10, 15, 20 мм. Допускается увеличение размера а на число, кратное 5. Размер b принима­ют равным 1,5а.

|  |  |
| --- | --- |
|  | Процесс. Выполнение операций (группы операций), в результате которых изменяется значение (форма представления, расположение) данных  Решение. Выбор направления алгоритма (програм­мы) в зависимости от некоторых переменных условий |
|  | Решение. Выбор направления алгоритма (програм­мы) в зависимости от некоторых переменных условий |
|  | Модификация. Выполнение операций, меняющих команды (группы команд), изменяющих программу |
|  | Предопределенный процесс. Использование ранее созданных и описанных отдельно алгоритмов (программ) |
|  | Ввод-вывод. Преобразование данных в форму, пригодную для обработки (ввод) или отображения результатов обработки (вывод) |
|  | Соединитель. Указание связи между прерванны­ми линиями потока (связывющие символы) |
|  | Пуск-останов. Начало, конец, прерывание про­цесса обработки данных или выполнения программы |
|  | Межстрочный соединитель |