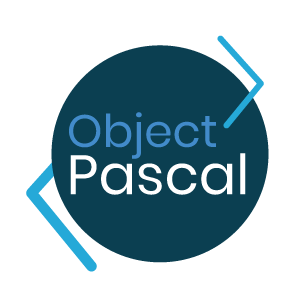
Руководство к выполнению лабораторных работ по ООП

**Язык программирования**

Предоставляется выбор из следующих языков:

Лабораторные работы можно делать, используя любой другой объектный язык, ***но*** на экзамене при ответе на билет разрешено приводить примеры только на языках из списка выше.

**Порядок сдачи**

Лабораторная работа засчитывается только при наличии ***рабочей*** программы вместе с ***исходным кодом*** и ***отчетом***. Отчет необходимо приносить в ***электронном виде***, образец отчета представлен ниже. Программы и отчет можно (и желательно) ***приносить на своем ноутбуке или ноутбуке вашего одногруппника***. В противном случае выполнение и сдача лабораторных производится на компьютере в аудитории.

Сдача работ происходит в порядке живой очереди с наивысшим приоритетом студентов группы, у которой лабораторная работа по расписанию. ***Сдавать лабораторные с другой группой можно***, но только после того, как сдадут все студенты, пришедшие по расписанию.

В связи с неблагоприятной эпидемической обстановкой студенты ***без масок*** до сдачи лабораторных ***не допускаются***.

**Задания на лабораторную работу.**

**Лабораторная работа №1. Создание графического оконного интерфейса**

**Задание 1.**

На выбранном языке программирования с помощью соответствующей среды разработки написать простое консольное приложение для демонстрации разработанного вами класса, описывающего вектор  в ***n+1***-мерном пространстве, где ***n*** – ваш номер в списке группы. Разработанный класс содержит:

1) Закрытое поле/поля, хранящие координаты вектора;

2) Один конструктор класса;

3) Методы доступа к полям;

4) Методы, реализующие вычисление норм



**Задание 2.**

Спроектировать оконный интерфейс приложений для последующих лабораторных работ. Суть работ: с помощью тех или иных механизмов создаются наборы классов, отвечающих за отрисовку простых геометрических фигур.

***Пользователь должен***:

- Создавать несколько объектов разных классов с помощью разных конструкторов (создать случайно, создать с заданными параметрами). При нажатии на кнопку создания выводить сообщение.

- Уметь перемещать все объекты одного класса разом ***и*** работать с отдельным объектом (например, объект можно выбрать их списка и затем его переместить).

- Уметь удалять выбранный объект ***и*** объекты одного класса.

- Уметь вызывать дополнительные функции выбранного объекта ***и*** объектов одного класса.

Наборы фигур, их цвета и дополнительный функционал представлены в таблице ниже.

|  |  |  |
| --- | --- | --- |
| Номер в списке группы | Фигуры | Дополнительный функционал |
| 1, 7, 13, 19, 25 | круг, линия, квадрат | изменение стороны квадрата |
| 2, 8, 14, 20, 26 | круг, линия, квадрат | изменение длины линии |
| 3, 9, 15, 21, 27 | круг, линия, квадрат | изменение радиуса круга |
| 4, 10, 16, 22, 28 | круг, квадрат, прямоугольник | изменение сторон прямоугольника |
| 5, 11, 17, 23, 29 | круг, квадрат, прямоугольник | увеличение сторон квадрата в два раза |
| 6, 12, 18, 24, 30 | круг, линия, квадрат | уменьшение радиуса круга в три раза |

**Лабораторная работа №2. Создание и использование отдельных классов для графических примитивов.[[1]](#footnote-1)**

Требуется создать небольшой набор ***не связанных*** друг с другом классов, описывающих основные графические примитивы из работы 1. Каждый класс должен содержать необходимые параметры-свойства (координаты базовой точки, размеры, признак видимости), не менее двух конструкторов, метод отображения примитива Show, метод для перемещения объекта MoveTo. Кроме того, классы содержат уникальный метод для реализации дополнительного функционала. Каждый класс оформляется в виде отдельного модуля, который подключается к основной программе для демонстрации возможностей.

Что должно быть:

1) ***Классы для примитивов (фигур) из работы 1***. Все классы для примитивов располагать в отдельной библиотеке или пакете.

2) Каждый из классов-примитивов имеет ***несколько закрытых полей данных***, причем поля в виде ***ссылок на объектные переменные*** допускаются ***в самом крайнем случае***. Взаимодействие с полями только через ***методы Get/Set***.

3) Каждый из классов-примитивов имеет минимум ***два различных конструктора***: один инициализирует объект по заданным пользователем параметрам, а второй – случайно.[[2]](#footnote-2)

4) Каждый из классов-примитивов имеет ***как минимум два метода***: метод отображения примитива ***Show*** и метод перемещения объекта ***MoveTo***. Метод Show допускает любые необходимые параметры, необходимые для отрисовки. Метод MoveTo принимает в качестве параметров ***смещение по координатам***.

5) Для работы с группами объектов одного класса использовать массив, стандартные коллекции использовать нельзя!

6) Демонстрационный модуль должен показать работу с примитивами.

***Примеры:***

1. Поля классов

public class Circle

{

private int x,y,r;

1. Реализация метода Show

C#

public void show(Graphics gc, Color color){

Pen pen = new Pen(color, 5);

gc.DrawEllipse(pen,this.x - r, this.y - r, r\*2,r\*2);

}

Вызов метода

circle.show(canvas.CreateGraphics(),Color.Blue);

Java

public void draw(GraphicsContext gc){

gc.setFill(Color.RED);

gc.fillOval(this.getX() - this.getR(), this.getY() - this.getR(), this.getR() \* 2, this.getR() \* 2);

}

Пакет javafx.scene.canvas.GraphicsContext

Эта и последующие лабораторные работы основаны на работе №2, НО программа должна соответствовать заданию и ***не содержать лишних компонент*** из других заданий.

**Лабораторная работа №3. Реализация взаимодействия классов на основе агрегации.[[3]](#footnote-3)**

Разработанные в работе 2 отдельные классы для графических примитивов необходимо модифицировать с целью реализации возможности их агрегационного взаимодействия. Для этого требуется ввести класс точек и заменить в классах для графических примитивов непосредственное использование свойств-координат объектами-точками.

Создать отдельный проект на основе предыдущего.

Что должно быть:

1) ***Базовый класс точка***, имеющий ***поля координат*** и ***методы доступа***, а также хотя бы ***один конструктор***, выводящий в консоль или в виде диалогового окна ***сообщение о создании точки***.

2) ***Изменить три класса из предыдущей работы*** так, чтобы они имели в качестве поля ссылку на объект-точку, и пользовались методами объекта-точки для работы с «центром» фигуры.

3) Добавить ***класс кольцо***, состоящий из двух объектов класса круг.

4) ***Придумать собственный уникальный геометрический объект***, состоящий из более простых, и разработать класс такого объекта на основе агрегации.

5) Во всех классах графических объектов, кроме точки, реализовать ***три конструктора*** – ***по заданным параметрам*** (например, Circle( int x, int y, int r)), ***с помощью уже созданного объекта*** (Circle (Point center, int r)), ***создающий объект со случайными параметрами*** вместе со всеми входящими в него объектами.

6) ***Внести изменения в демонстрационный модуль***, чтобы пользователь мог работать и с новыми объектами.

**Лабораторная работа №4. Создание и использование библиотеки классов для графических примитивов на основе принципа наследования.[[4]](#footnote-4)**

Требуется создать небольшую иерархию классов, описывающих основные графические примитивы из работы 1. Корнем всей иерархии является класс TFigure, определяющий общие свойства и поведение всех объектов-примитивов: координаты базовой точки примитива, конструктор, методы доступа, абстрактные методы прорисовки Show и перемещения MoveTo. В каждом классе необходимо реализовать свой конструктор и методы прорисовки и перемещения. Кроме того, классы могут содержать методы, уникальные только для соответствующего поддерева (смотри работу 1). Вся библиотека оформляется в виде отдельного модуля, который подключается к основной программе для демонстрации возможностей этой библиотеки.

Что должно быть:

1) ***Абстрактный класс***, содержащий ***свойства-координаты***, ***хотя бы один конструктор*** и ***два абстрактных метода*** – отрисовка фигуры ***Show*** и перемещение ***MoveTo***.

2) ***Три класса*** простых фигур, ***наследуемых*** непосредственно от ***абстрактного класса*** пункта 1. Требования к методам этих и следующих классов такие же, как и для лабораторным 1 и 2, но с поправкой на переопределение абстрактных методов.

3) ***Создать класс***, отвечающий за ***эллипс***, на основе класса окружности, т.е. с добавлением нового свойства-полуоси помимо основного (радиус) у окружности. В этом классе ***дополнительно реализовать метод поворота на 90 градусов*** (поменять местами полуоси).

4) Для класса квадрат сделать ***минимум два из трех подклассов***: ***ромб, трапеция, прямоугольник***.

5) Внести соответствующие изменения в демонстрационный модуль. ***Добавить*** возможность вызова ***специального метода у эллипса***.

**Лабораторная работа №5. Создание библиотеки классов для графических примитивов с использованием виртуальных методов.[[5]](#footnote-5)**

Требуется изменить созданную ранее (в работе 4) иерархию классов для основных графических примитивов с целью использования виртуальных методов. Для этого в иерархию классов необходимо внести некоторые изменения.

Что должно быть:

1) Внести следующие ***изменения в абстрактный класс***: ***метод отрисовки*** сделать ***виртуальным*** (переопределяемым), ***метод перемещения*** – ***реальным[[6]](#footnote-6), с вызовом метода отрисовки***.

2) ***В наследуемых классах*** ***убрать метод перемещения***.

3) Демонстрационный модуль оставить ***без изменения***.

В следующих лабораторных работах ***запрещено*** использовать стандартные библиотеки, реализующие контейнеры объектов, например, списки, и какие-либо их модификации (касается только работ с фигурами).

**Лабораторная работа №6. Использование полиморфных объектных указателей.[[7]](#footnote-7)**

На основе созданной в предыдущем задании библиотеки графических фигур необходимо реализовать приложение для демонстрации основных возможностей полиморфных указателей. Для этого требуется создать простейший контейнер с возможностью занесения в него любых объектов-фигур. Простейший контейнер (иногда называемый коллекцией) можно создать на основе массива полиморфных указателей. Должны быть предусмотрены следующие операции:

• заполнение массива указателями на случайно выбранные фигуры;

• показ всех фигур в массиве;

• стирание всех фигур в массиве;

• перемещение сразу всех фигур в новую базовую точку;

• выборочная обработка только фигур из некоторого поддерева (например, перемещение только окружностей и эллипсов) за счет динамического определения типа объектного указателя при прохождении по массиву;

• уничтожение массива фигур с освобождением памяти.

Что должно быть:

1) ***Классы*** примитивов ***из работы 4 или 5***.

2)В демонстрационном модуле ***объявлен массив*** со ссылками на объекты, наследуемые от абстрактного класса.

3) Более конкретно по функциям контейнера (массива) –показ и стирание фигур – сделать в классах примитивов, если не было ранее, функцию, обратную отрисовки, которая стирает с холста данную фигуру; перемещение в базовую точку можно сделать либо отдельным методом (в классах примитивов), либо через методы доступа к координатам центра; работа с отдельным поддеревом подразумевает использование is или instanceof, поэтому обработка, к примеру, сразу окружностей и эллипсов допускается (можно сделать и только обработку каждого отдельного класса).

4) Обязательно использовать в программе операторы типа ***is или instanceof*** при работе с подиерархией.

5) Демонстрационный модуль должен показать работу с коллекцией (массивом). ***Пользователь должен***:

- создать массив из ***«10+номер по списку»*** случайных фигур;

- вызывать все функции по работе с массивом (показ всех фигур, стирание всех фигур, перемещение фигур в новую базовую точку, выборочная обработка, уничтожение массива);

- дополнительно у эллипсов (если они есть) вызывать уникальный метод, в противном случае вызвать уникальный метод другой фигуры;

- по нажатию стрелок на клавиатуре (или нажатию в угол экрана) перемещать на ***«10+номер по списку»*** пикселей все фигуры в соответствующем направлении.

***Функционал из прошлых работ***, т.е. возможность работы с отдельными фигурами, ***убрать***. В отчете – убрать пункт «описание» классов, а в листинге указать с комментариями те места кода, в которых происходила работа с массивом.

**Лабораторная работа №7. Создание и использование контейнерных классов.[[8]](#footnote-8)**

Требуется создать класс-контейнер для хранения любых графических примитивов из ранее созданной библиотеки. Контейнер должен быть реализован ***в двух вариантах*** – на базе динамического массива (т.е. если массив закончился – расширяем, например, на 10% от исходного объема) и на базе линейного динамического списка. Каждый контейнерный класс реализуется в отдельном модуле. Демонстрационная программа должна показывать использование созданных контейнеров.

Что должно быть:

1) ***Классы*** примитивов ***из работы 4 или 5***.

2)Всего ***должно быть*** разработано ***не менее 3х классов[[9]](#footnote-9)*** – один для контейнера-массива, два для контейнера-списка (один непосредственно контейнер, а другой – элемент списка). Принцип инкапсуляции соблюдается и для этих классов. Допускается использование интерфейсов (например, IEnumerable), но только в крайних случаях.

3) Контейнер на базе массива должен изначально ***создаваться*** на ***«10+номер по списку»*** элементов (сделать конструктор). При нехватке места массив увеличивается на 10 или более процентов от исходного объема.

4) Для реализации списка необходимо помимо основного создать отдельный класс, хранящий указатель, например, на следующий за ним объект того же класса и указатель на абстрактную фигуру.

5) Создать ***два конструктора*** в каждом контейнерном классе – один создаёт пустой контейнер (для массива – с указанным размером), а другой после создания заполняет контейнер на ***«номер по списку»*** случайных фигур.

6) Добавление в список можно сделать как в начало, так и в конец списка.

7) Оба контейнерных класса содержат ***метод-итератор***, который организует проход по массиву или списку и с каждым объектом-примитивом выполняет заданное действие – ***отрисовка, стирание, перемещение***.

8) Контейнеры реализуют следующие функции:

- добавление случайной фигуры со случайными параметрами;

- обработка всех фигур (см. пункт 7);

- удаление всех фигур и самого контейнера.

9) Демонстрационный модуль должен показать весь функционал созданных контейнеров. ***Пользователь должен:***

- задать тип создаваемого контейнера;

- создать пустой контейнер или заполненный на «***10+номер по списку***» фигур;

- добавить одну случайную фигуру;

- нарисовать все фигуры (изначально они не показаны);

- переместить все фигуры по заданному смещению;

- стереть все фигуры;

- уничтожить контейнер.

При этом либо пользователь может одновременно работать со списком и массивом в рамках одного модуля, ***либо*** для каждой коллекции создаётся отдельный демонстрационный модуль.

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное бюджетное образовательное учреждение высшего**

**профессионального образования**

**«Казанский национальный исследовательский технический университет**

**им. А.Н. Туполева-КАИ»**

Институт компьютерных технологий и защиты информации

Кафедра Прикладной математики и Информатики имени Ю.В. Кожевникова

Лабораторная работа № X

по дисциплине «Объектно-ориентированное программирование»

**Выполнил**

студент группы 42XX

Username

Казань

20XX

Тема[[10]](#footnote-10)

Численные методы нахождения эпсилон-постоянной Липшица и их приложения.

Цель[[11]](#footnote-11)

Ставится задача поиска минимальной оценки ε-постоянной Липшица непрерывной на параллелепипеде n-мерного евклидова пространства функции.

Обобщить алгоритм Стронгина на случай минимизации непрерывной на отрезке функции, доказана его сходимость. Для работы алгоритма не требуется априорного знания оценки ε-постоянной Липшица – эта оценка уточняется в ходе работы алгоритма.

Описание классов[[12]](#footnote-12)

**Класс ExStrongin**

Программная реализация (без реализации методов):

import java.util.LinkedList;

public class ExStrongin extends Method{

private double v, m, m0;

private double Lext;

private double le;

private LinkedList<ExStronginIteration> Iterations;

public ExStrongin(Function f, double e, double v, double m, double m0) throws InitializationException {}

private void Search(){}

private double LkeCount(List Xk, int smax, double vanEps){}

private int RkiCount(List Xk, double Lke, double vanEps){}

private double vkCount(List Xk, double Lke, int s){}

public double getV() {}

public double getM() {}

public double getM0() {}

public double getLext() {}

public double getLe() {}

public LinkedList<ExStronginIteration> getIterations() {}

}

**Описание свойств:**

1) Параметры метода

private double v, m, m0;

2) Точное значение оценки ε-постоянной Липшица и коэффициент уменьшения шага (возмущения)

private double Lext, alpha;

3) Вспомогательная переменная, хранящая текущее наибольшее значение l(ε)

private double le;

4) Лист итераций. Содержит в себе объекты класса ExStronginIteration, в котором имеются все необходимые поля для хранения промежуточных данных, тривиальный конструктор и методы доступа к полям

private LinkedList<ExStronginIteration> Iterations;

**Описание методов:**

1) Конструктор для создания метода, реализующего обобщенный метод Стронгина

public ExStrongin(Function f, double e, double v, double m, double m0) throws InitializationException {}

2) Реализация поиска глобального минимума непрерывной на отрезке функции обобщенным алгоритмом Стронгина. По окончании работы в поле Iterations будет храниться список с данными всех итераций.

private void Search()

3) Вычисляет значение Lk(ε) для заданного набора точек из List Xk. Поиск производится относительно вновь найденной поисковой точки с номером smax.

private double LkeCount(List Xk, int smax, double vanEps){}

4) Вычисляет параметр Rk(i,ε) для набора точек из List Xk

private int RkiCount(List Xk, double Lke, double vanEps){}

5) Поиск очередной поисковой точки для интервала с номером s

private double vkCount(List Xk, double Lke, int s){}

6) Методы доступа

public double getV() {}

public double getM() {}

public double getM0() {}

public double getLext() {}

public double getLe() {}

public LinkedList<ExStronginIteration> getIterations() {}

Описание структуры проекта

Структура папок проекта разработанного приложения имеет вид:

build

classes

Frame

Images

Logic

empty

generated-sources

ap-source-output

dist

lib

nbprojects

private

src

Frame

Images

Logic

build.xml

manifest.mf

В папке build хранятся скомпилированные файлы классов, имеющие расширение .class (бинарный код). В папке dist хранятся файлы, предназначенные для распространения как результат компиляции (модуль JAR приложения). Также в этой папке хранятся используемые библиотеки (папка lib). В папке nbproject находится служебная информация по проекту. В папке src находятся исходные коды классов (папки Frame и Logic) и изображения в папке Images. В той же папке src в XML-файлах с расширением .form хранится информация об экранных формах.

Файлы .properties нужны для хранения статических данных, необходимых проекту. JAR-файл — это Java-архив (сокращение от англ. Java ARchive), который представляет собой ZIP-архив, содержащий часть программы на языке Java. Чтобы JAR-файл был исполняемым, он должен содержать файл MANIFEST.MF в каталоге META-INF, в котором должен быть указан главный класс программы.

Руководство пользователя[[13]](#footnote-13)

Для запуска приложения необходимо, чтобы на операционной системе конечного пользователя была установлена среда выполнения для Java JRE версии не менее 1.7.0.

После запуска приложения в верхней горизонтальной строке меню (Рис.1) следует выбрать пункт «Функция». В появившемся диалоговом окне «функция» сначала необходимо выбрать вид функции из выпадающего списка (Рис. 2). Далее требуется указать ε в поле «epsilon» и указать параметры выбранной функции, заполнив соответствующие поля (разделителем между целой и дробной частью является точка).

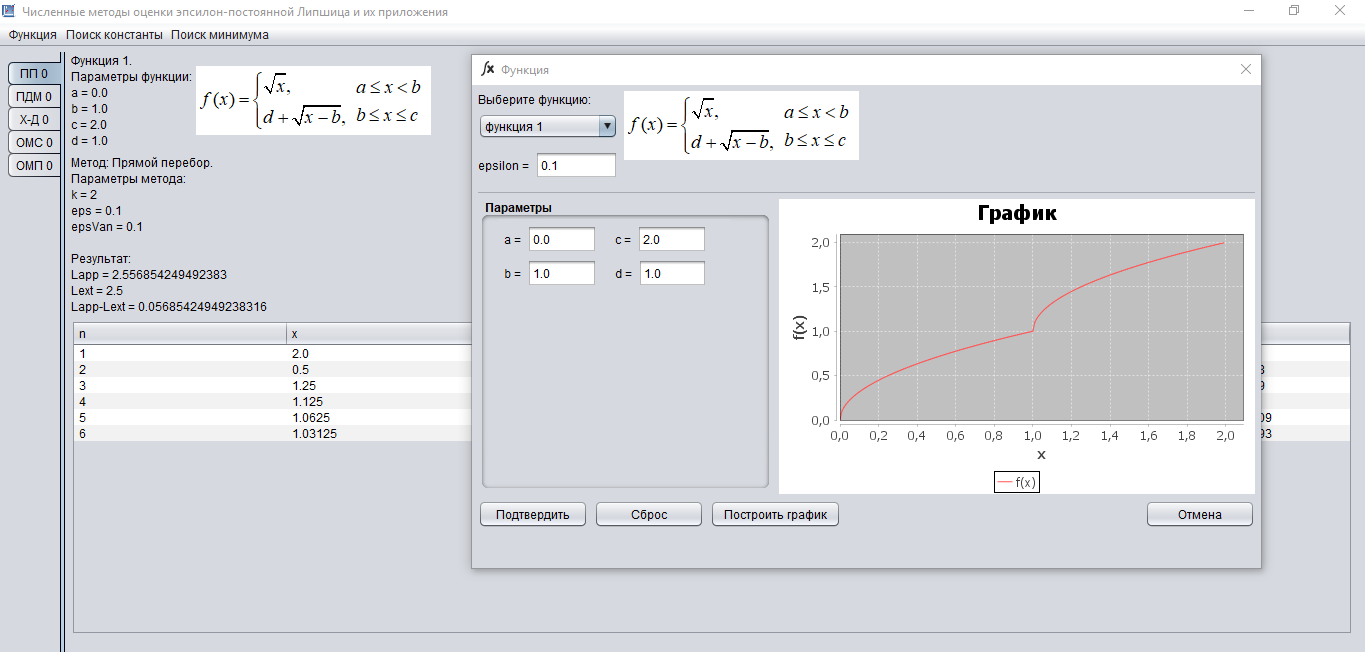


Рисунок 1. Главное окно программы

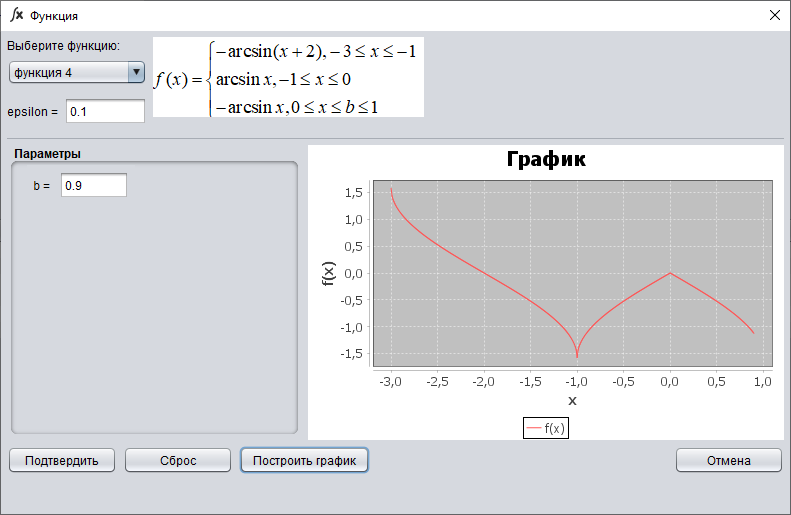


Рисунок 2. Диалоговое окно выбора функции

Листинг программы[[14]](#footnote-14)

import java.util.LinkedList;

import java.util.List;

public class ExStrongin extends Method{

private double v, m, m0;// параметры метода - минимальное значение l(e),

// увеличение константы, увеличение константы при

// не попадания новой точки в границы интервала

private double Lext;// точное значение l(e)

private double le;// вспомогательная переменная

private LinkedList<ExStronginIteration> Iterations;// лист итераций

public ExStrongin(Function f, double e, double v, double m, double m0)

throws InitializationException {

super(f, e);

if (v<0 || m<0 || m0<0)

throw new InitializationException("Неверные параметры метода! "

+ "(Все параметры должны быть положительные!)");

this.v=v;

this.m=m;

this.m0=m0;

Lext=f.L(f.vanEps);

Iterations=new LinkedList();

Search();

}

private void Search() { // поиск миниммума согласно обобщенному

// алгоритму Стронгина

List Xk = new LinkedList(); // текущий набор точек

double Lk,vk,vkprev; // оценка e-постонной, новая и старая поисковые точки

int s=0; // номер отрезка с наибольшей числовой характеристикой

int step=0; // число шагов

Xk.add(f.a);// добавление края отрезка

Xk.add(f.b);

Lk=LkeCount(Xk,s,f.vanEps); // вычисление e-постоянной

s=RkiCount(Xk,Lk,f.vanEps); // нахождение номер отрезка с наибольшей числовой характеристикой

vk=vkCount(Xk,Lk,s); // нахождение следующей точки

Xk.add(vk); // добавление найденной точки

Xk.sort(null); // упорядочивание точек по возрастанию

do{

System.out.println("Step "+step+": Lk="+Lk

+"; s="+s+"; vk="+vk+"; f(vk)="+f.f(vk));

vkprev=vk;

Lk=LkeCount(Xk,s,f.vanEps);

s=RkiCount(Xk,Lk,f.vanEps);

vk=vkCount(Xk,Lk,s);

Xk.add(vk);

Xk.sort(null);

step++;

Iterations.add(new ExStronginIteration(step,vk,f.f(vk),Lk,

Math.max(Math.abs(f.f(vk)-f.f(vkprev)),Math.abs(vk-vkprev))));

}while(Math.abs(f.f(vk)-f.f(vkprev))>e || Math.abs(vk-vkprev)>e );

1. Может быть полезен файл «пособие (практические задания)» пункт 1.5. Практические задания Задание 1 [↑](#footnote-ref-1)
2. Как делать неправильно – задавать в конструктор параметры, которые были случайно сгенерированы программой. Параметры задаются случайно самим конструктором! [↑](#footnote-ref-2)
3. Может быть полезен файл «пособие (практические задания)» пункт 2.4. Практические задания Задание 1 [↑](#footnote-ref-3)
4. Может быть полезен файл «пособие (практические задания)» пункт 2.4. Практические задания Задание 2 [↑](#footnote-ref-4)
5. Может быть полезен файл «пособие (практические задания)» пункт 3.4. Практические задания Задание 1 [↑](#footnote-ref-5)
6. В java – директивой final, в С# - как обычный метод [↑](#footnote-ref-6)
7. Может быть полезен файл «пособие (практические задания)» пункт 3.4. Практические задания Задание 2 [↑](#footnote-ref-7)
8. Может быть полезен файл «пособие (практические задания)» пункт 3.4. Практические задания Задание 3 [↑](#footnote-ref-8)
9. Их же и описывать в отчете [↑](#footnote-ref-9)
10. Указывается тема лабораторной работы. Данная тема взята из выпускной работы только в качестве примера! [↑](#footnote-ref-10)
11. Приводится постановка задачи на лабораторную работу [↑](#footnote-ref-11)
12. Краткое описание основных свойств и методов разработанных классов (классы, отвечающие за интерфейс приложения не описывать). [↑](#footnote-ref-12)
13. Кратко описать все основные возможности разработанного приложения [↑](#footnote-ref-13)
14. Исходный код только для классов, описанных в пункте «Описание классов» [↑](#footnote-ref-14)