|  |  |
| --- | --- |
| Федеральное государственное бюджетное образовательное учреждение высшего образования | |
| Югорский государственный университет (ЮГУ) | |
| Институт (НОЦ) систем управления и информационных технологий (ИСУиИТ) | |
| Кафедра компьютерного моделирования и информационных технологий (КМИТ) | |
|  | |
|  |  |
| Реализация базовых алгоритмов и структур данных на языке Java. сортировка шелла. ЭЛЕКТРОННЫЙ УЧЕТ ПОСЕЩАЕМОСТИ СТУДЕНТОВ. | |
| Курсовой проект по дисциплине «Объектно-ориентированное программирование» | |
|  | |
| Студент гр. 1541б \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Чернавский А.Ю./  (подпись) | |
| Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Бурлуцкий В.В./  (подпись) | |
|  | |
|  | |
| Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | |
| Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | |
|  | |

Ханты-Мансийск, 2016 г.

**Содержание**

1 Введение.……………………………………………………………………….......3

2 Основная часть[……………………………………………………………………..](#_2_Основная_часть)6

2.1 Постановка задачи[……………………………………………………………….](#_2.1_Постановка_задачи)6

2.2 Технические и программные средства[…………………………………………](#_2.2_Технические_и)7

2.2.1 Программные средства[………………………………………………………..](#_2.2_Технические_и)7

2.2.2 Технические средства[………………………………………………………..](#_2.2_Технические_и)..9

2.3 Математические методы и модели[……………………………………………](#_2.3._Математические_методы)..9

2.3.1 Виды сортировок[……………………………………………………………..](#_2.3.1_Виды_сортировок)..9

2.3.2 Алгоритм сортировки вставками[……………………………………………](#_2.3.1_Виды_сортировок)10

2.3.3 Алгоритм сортировки Шелла[………………………………………………..](#_2.3.1_Виды_сортировок)11

2.4 Алгоритмы работы и организация данных[……………………………….......](#_2.4_Алгоритмы_работы)13

2.4.1 Работа программы[……………………………………………………………](#_Во_время_запуска)17

2.5 Результаты работы программы и их оценка[………………………………….](#_2.5_Результаты_работы)20

2.5.1 Тестирование[………………………………………………………………….](#_2.5.1_Тестирование)20

2.5.2 Отладка[………………………………………………………………………..](#_2.5.2._Отладка)22

3 Заключение[………………………………………………………………………..](#_3_Заключение)24

Список использованных источников[………………………………………….......](#_Список_использованных_источников)25

Приложение А (обязательное) Исходный программный код[……………….......](#_Список_использованных_источников)26

Приложение Б (справочное) Текст справки[………………………………………](#_Список_использованных_источников)35

**1 Введение**

Цели курсового проекта:

1. Изучить парадигму ООП, основные понятия и принципы работы с объектами, классами и интерфейсами.
2. Получить навыки разработки приложений на объектно-ориентированном языке программирования Java.
3. Разработать программный проект, реализующий базовый алгоритм сортировки структуры данных, используя сортировку Шелла.

Задачи курсового проекта:

1. Спроектировать программный продукт.
2. Написать программный код продукта в IDE Intellij Idea.
3. Разработать интерфейс продукта в IDE Intellij Idea.
4. Протестировать программный продукт.
5. Отладить программный продукт.

Область применения разрабатываемого проекта:

Данный разрабатываемый продукт может использоваться в учебных целях, поскольку наглядно реализует базовый алгоритм сортировки структуры данных.

Таблица 1 – Календарный план

|  |  |  |
| --- | --- | --- |
| Наименование этапа работы | Плановый срок выполнения этапа работы | Результат выполнения |
| Изучение литературы о базовых структуре данных и алгоритме | 01.02.16 – 14.02.16 | Раздел КП постановка задачи |
| Изучение технической документации по языку. Проектирование программного модуля | 15.02.16 –  28.02.16 | Архитектурное решение, раздел КП |
| Формирование требований | 29.02.16 –  12.03.16 | ТЗ, раздел КП, прототипы интерфейсов. |
| Кодирование | 13.03.16 –  26.03.16 | Первый прототип, раздел КП. |
| Промежуточный контроль результатов работы | 30. 03. 16 |  |
| Тестирование и отладка | 04.04.16 –  25.04.16 | Второй прототип модуля, разделы КП, отчет о тестировании. |

Продолжение таблицы 1 – Календарный план

|  |  |  |
| --- | --- | --- |
| Наименование этапа работы | Плановый срок выполнения этапа работы | Результат выполнения |
| Оформление и эксплуатационная документация | 26.04.16 –  09.05.16 | Оформление текста пояснительной записки к КП. |
| Защита КП | 30.05.16 | Доклад. |

# 

# 2 Основная часть

# 2.1 Постановка задачи

Поставлена задача: разработать программную систему с функцией сортировки заданной структуры данных при помощи базового алгоритма сортировки Шелла.

Дополнительной и отдельной задачей проекта является работа с файлами Excel и составления отчета посещаемости студентов по данным документам. Для совмещения этих задач было принято решение сортировать структуру данных, включающую в себя студентов, при помощи сортировки Шелла.

Программная система должна реализовать следующие функциональные требования:

a) Создание новой структуры, элементами которой являются строки с данными о студентах, хранящиеся во внешнем файле.

b) Упорядочивание структуры сортировкой Шелла.

c) Изменение значения элемента в структуре.

d) Удаление элемента из структуры.

e) Визуализация сортировки.

f) Обработка исключительных ситуаций при работе с файлами.

g) Вывод справочной математической информации о структуре данных.

h) Вывод информации о посещаемости студентов.

Программная система должна быть построена в соответствии с объектно-ориентированной парадигмой программирования. Время отклика программы при выполнении каждой из функций не должно превышать 10 секунд. Дополнительных требований к защите и производительности не предъявляется. Программная система разрабатывалась по техническому заданию (ТЗ). Оно приложено к данному курсовому проекту. По данной программной системе создано приложение, работающее на различных операционных системах c JVM, благодаря кроссплатформенности языка Java, которое называется «Shell sort».**[Приложение А (обязательное) Исходный программный код]**

# 

# 2.2 Технические и программные средства

**2.2.1 Программные средства**

Данный проект разрабатывался при помощи при помощи следующих программных средств:

Среда разработки - IDE Intellij Idea community edition версии 15.0.3

Набор Java разработки - JDK-8u91

Проект разрабатывался на основе технологии JavaFX, используя Framework для упрощенной разработки интерфейса - JavaFX Scene Builder версии 2.0**[1]**

Система контроля версий - Git версии 2.7.3

Приложение разработано в операционной системе Windows 7.

Приложения, созданные посредством IDE Intellij Idea данной версии, способны работать на ОС Windows XP/Vista/7/8 или новее. При условии того, что на ОС будет установлена JVM – Java Virtual Machine, которая используется для поддержки запуска приложений, написанных на Java.

Java**[2]** была протестирована, переработана, расширена и улучшена избранным сообществом программистов, архитекторов и энтузиастов.

Отличительные особенности от других языков – скорость выполнения программ, однако скорость выполнения программ ниже, чем на C/C++, что связанно с особенностью программ на Java языке – запуск на JVM, что, однако, дает ей большое преимущество – кроссплатформенность. Так же Java имеет большую популярность у разработчиков за надежность и универсальность.

Для данного проекта был выбран язык Java в соответствии с техническим заданием. Java значительно облегчает работу со структурами данных посредством представления их в виде объектов, классов, интерфейсов и ссылок (аналоги указателей в других языках программирования высокого уровня).

**2.2.2 Технические средства**

Приложение «Shell sort» разрабатывалось и тестировалось при помощи ноутбука Dell Inspiron 3521 на базе x64 процессора Intel(R) Core(TM) i3-3217U, CPU @ 1.80GHz и 4 Гб ОЗУ.

По результатам тестирований данных технических характеристик более чем достаточно для решения задач разрабатываемой программной системы.

# 2.3. Математические методы и модели

# 2.3.1 Виды сортировок

Что такое сортировка? Это последовательное расположение или разбиение множества объектов по какому-либо признаку.

Цель сортировки – облегчить последующий поиск элементов в таком отсортированном множестве.

Существует достаточно много различных алгоритмов сортировок - сортировки выбором, сортировки вставками, сортировки слиянием, сортировки распределением, гибридные сортировки и параллельные сортировки. Есть, кстати, ещё эзотерические сортировки. Это различные шуточные, принципиально нереализуемые и прочие псевдо-алгоритмы.

Хорошей мерой эффективности может быть C – число необходимых сравнений ключей и M – число пересылок (перестановок) элементов. Эти числа – суть функции от числа n-сортируемых элементов.[3]

Хорошие алгоритмы требуют около n \* log n сравнений. Такие алгоритмы называются *улучшенными*. Алгоритмы, где требуется порядка n ^ 2 сравнения ключей, называются *прямыми.*

Прямые сортировки удобны для объяснения характерных черт основных принципов большинства сортировок, в программировании они легки в понимании и достаточно коротки. Поэтому обычно с них и начинают знакомства с методами сортировок.

**2.3.2 Алгоритм сортировки вставками**

Сортировка, используемая в нашем проекте, а именно – сортировка Шелла – относится к числу одной из эффективных сортировок и является улучшенной версией сортировки вставками.

Сама суть алгоритма вставками знакома практически всем: представьте кошелек, в котором лежат купюры разного номинала от 100 до 10000, такие как 100, 500, 1000, 5000, 10000. Сначала мы ищем минимальное число среди всех. Очевидно, что это 100. Далее мы перекладываем ее в начало. Повторяем. Находим еще одну купюру, номиналом в 100 единиц. Кладем ее в начало. Представим, что следующая купюра – 1000. Однако по номиналу следующая минимальная купюра – 500. Алгоритм вставками потому так и называется, потому что на данном шаге нам необходимо вставить эту купюру между 100 и 1000 купюрами. И так алгоритм продолжается до полной сортировки массива.

На языке Java такая сортировка программируется всего в три строчки кода, как видно из рисунка 2.3.2.

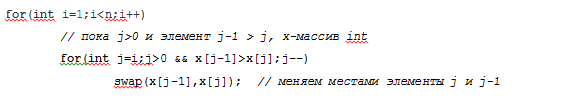


Рисунок 2.3.2 – сортировка вставками

**2.3.3 Алгоритм сортировки Шелла**

Как было сказано выше, алгоритм сортировки Шелла – это усовершенствованная сортировка вставками. Для начала разберем исходный код сортировки на языке Java на рисунке 2.3.3.

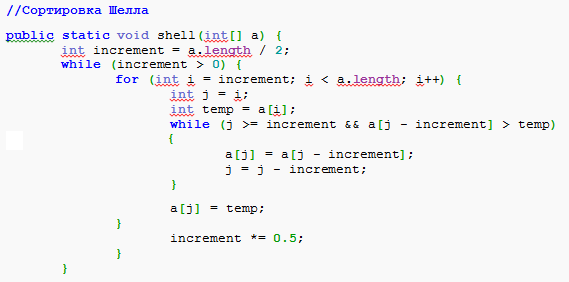


Рисунок 2.3.3 – сортировка Шелла

Для начала нужно определить исходные данные. В данном случае – это массив элементов a. Следующим шагом необходимо определить начальный шаг сортировки (этим совершенствуется сортировка вставками). Стандартно – это половина длины массива.

Рассмотрим саму сортировку. Наш шаг будет уменьшаться в ходе программы в 2 раза, а пока он не уменьшился до 0, программа выполняет свои действия.

Добавим в программу счетчик. Своеобразный «gps», который подсказывает, где мы сейчас находимся. Для начала мы проверяем, можем ли мы двигаться по массиву влево на «счетчик» ходов. Если да, то смотрим, какой там элемент. (Напоминаю, что обычно массив сортируется по возрастанию значений слева-направо. Воспользуемся этим принципом). Если слева элемент больше, чем текущий – то мы их меняем местами и переходим на некоторое время к данному второму элементу.

Далее, от второго элемента проверяем те же самые условия: можем ли мы двигаться в левую сторону и меньше ли там значение. При неверном условии мы возвращаемся к первоначальному значению и передвигаемся от нее на один шаг вправо (пока не доберемся до конца массива).

С данной ячейкой проделываем то же самое, что с первоначальной: проверяем, переходим, меняем, если число больше.

Когда мы дойдем до конца массива, следует поделить пополам ячейку шага (то есть уже четверть массива). Теперь массив проверяется в ширину на четверть от своей длинны.

**Замечание.** При делении шага берем целую часть числа, отбрасывая остаток. Шаг может быть только целым значением.

И вот, когда значение шага наконец станет равным 1, мы будем находится в условиях **сортировки вставками**.

Этим и отличается сортировка Шелла от сортировки вставками. Описание длинное, код тоже занимает больше места, однако мы выигрываем во времени сортировки небольших массивов. На массивах небольших размеров (например, 15 значений) эта сортировка не будет уступать по скорости даже быстрой сортировке.

Разница ощутима только при использовании задач с большими объемами данных.

# 2.4 Алгоритмы работы и организация данных

Работа сортировки Шелла используется в программе на разных этапах. Она сортирует заданный массив учеников по порядковому номеру, либо по количеству пропусков. Для этого создан класс Pupil (Рисунок 2.4.1) (ученик), у которого разработаны поля:

• id (порядковый номер)

• name (ФИО ученика)

• misses (пропуски ученика в виде дробного десятичного числа)

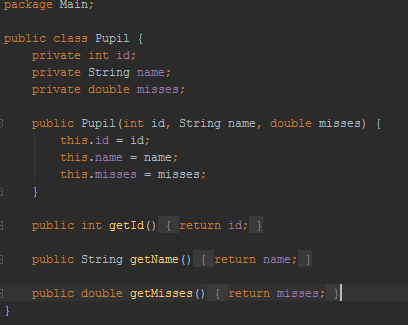


Рисунок 2.4.1 – класс Pupil

Данные об учениках хранятся в файле вида Excel.xlsx версии 2007 года и старше. Пример на рисунке 2.4.2.

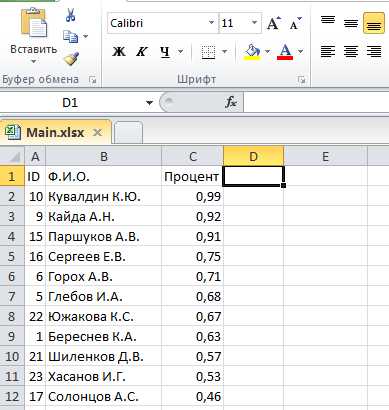


Рисунок 2.4.2 – файл Excel

Внутри программы данные хранятся в специализированном листе, данные из которого после прочтения файла Excel переносятся на окно программы и одновременно заносятся в память индексированного массива для дальнейшей сортировки по номеру или по пропускам учеников.**[4,5]** как на рисунке 2.4.3

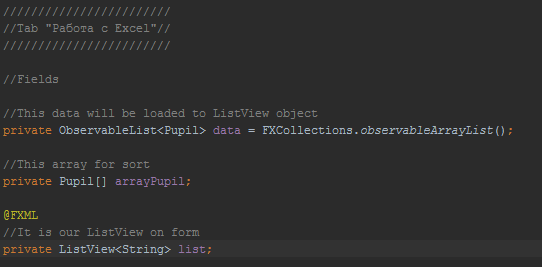


Рисунок 2.4.3 – Листы для хранения данных

Сортировка Шелла в соответствии пункту 2.3.3 используется для сортировки массива учеников по номеру или по количеству пропусков, а так же для визуального представления сортировки случайного массива (Рисунок 2.4.5).

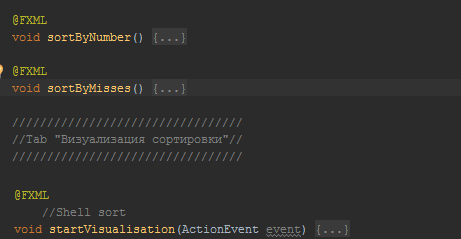


Рисунок 2.4.5 – методы с сортировкой Шелла

В визуальной сортировке был применен принцип мульти-поточного программирования**[6]** (пример на рисунке 2.4.6), поскольку программа не могла последовательно рисовать графические элементы с заданной частотой в одном потоке. Поэтому было принято решение вынести код обработки визуализации в отдельный поток, т.к. в основном потоке должен рисоваться графический интерфейс пользователя, поэтому программу нельзя останавливать в основном потоке, иначе это приведет к эффекту зависания.

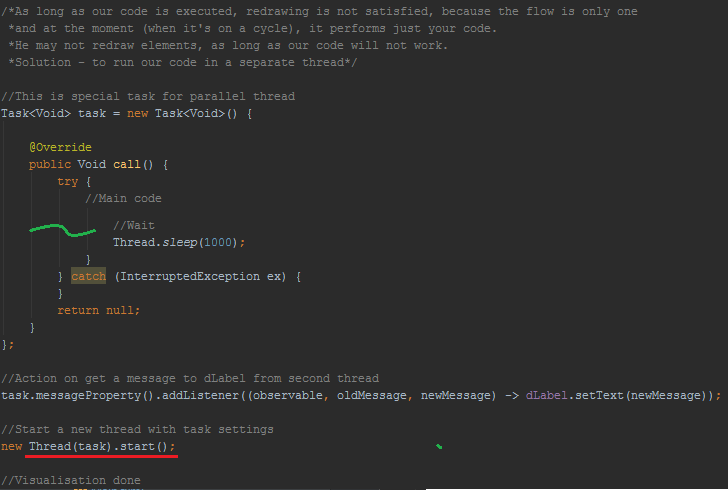


Рисунок 2.4.6 - мультипоточность

Красным подчеркиванием вы можете видеть создание нового потока для работы кода отдельно от рисования графического интерфейса.

Вся структура и логика программы собрана в исполняемый файл Shell sort.jar, который запускает программный комплекс, отвязывая его от среды разработки.

Так же весь проект был загружен на репозиторий сервиса GitHub, где есть возможность познакомиться с полной структурой проекта и посмотреть результаты его работы.

**2.4.1 Работа программы**

# Во время запуска приложения создаются основные формы программы, непосредственно после загрузки всех ресурсов и некоторые поля, для хранения данных.

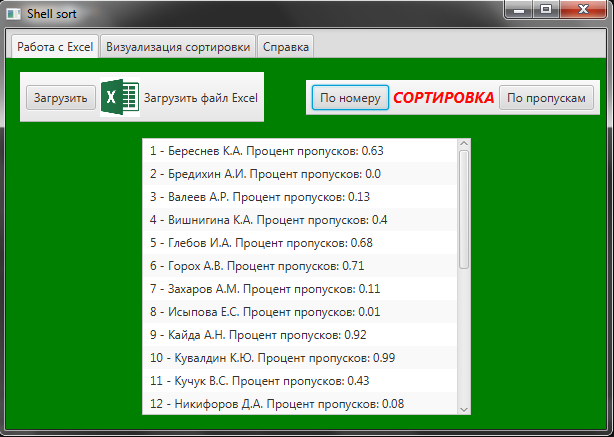


Рисунок 2.4.7 – работа с Excel

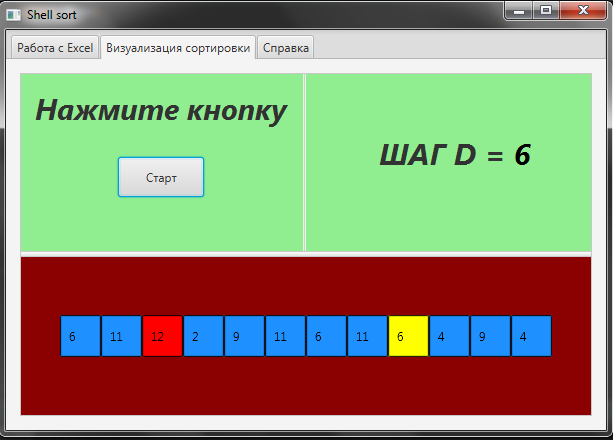


Рисунок 2.4.8 – визуальное отображение сортировки

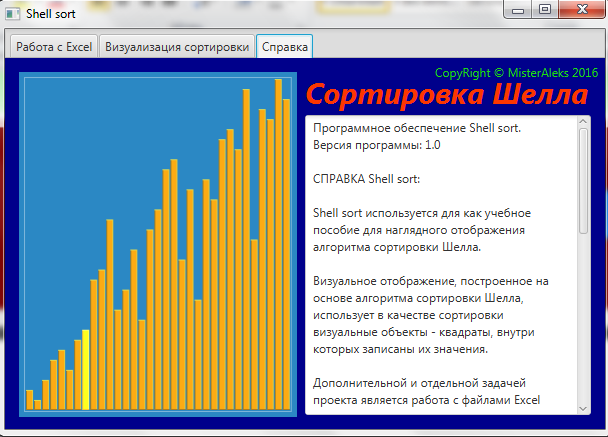


Рисунок 2.4.9 – справка программы

Приложение «Shell sort» состоит из 3 форм: 2 основных и 1 справочной.**[Приложение Б (справочное) ТЕКСТ СПРАВКИ]**

Интерфейс программы достаточно прост и интуитивно понятен. На вкладе "Работа с Excel" вы найдете всего 3 кнопки(рисунок 2.4.7):

"Загрузить" - что значит загрузить данные из файла Main.xlsx в таблицу на форме.

"По номеру" - сортировка загруженных данных по номеру при помощи сортировки Шелла.

"По пропускам" - сортировка загруженных данных по пропускам при помощи сортировки Шелла.

На вкладке "Визуализация сортировки" кнопка старт запустит наглядную визуализацию сортировки Шелла с 12 различными числами, которые случайно задаются при каждом новом нажатии кнопки. (рисунок 2.4.8)

Визуальное отображение, построенное на основе алгоритма сортировки Шелла, использует в качестве сортировки визуальные объекты - квадраты, внутри которых записаны их значения.

На вкладке "Справка" вы найдете краткое описание программы.(рисунок 2.4.9)

# 2.5 Результаты работы программы и их оценка

# 2.5.1 Тестирование

Цель тестирования – выявление ошибок в работе тестируемой программы. Во время тестирования приложения «Shell sort» были обнаружены ошибки и недочеты, которые представлены в таблице 2.1.

Таблица 2.1 – выявленные ошибки в работе приложения «Shell sort»

|  |  |  |
| --- | --- | --- |
| Действие | Ожидаемый результат | Реальный результат |
| Загрузка из файла в таблицу | Открытие файла | Отсутствие обработки исключения отсутствия файла. Ошибка выводится в консоль, программа продолжает работать (Недочет) |
| Сортировка пустых значений | Ничего не происходит | Ошибка в консоли (Недочет) |

# 

# 2.5.2. Отладка

Цель отладки – исправление выявленных ошибок и недочетов в работе программы. За время отладки все вышеописанные ошибки были исправлены. Результаты отладки представлены в таблице 2 2.

Таблица 2.2 – исправление ошибок и отладка работы приложения

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Действие | Реальный результат | В чем заключается ошибка | Место локализации | Способ исправления |
| Нажатие на кнопку Загрузить | Отсутствие действий. Ничего не происходит | Отсутствие или повреждение файла | Class Controller  Method, openExcel | Добавлен try {} catch{} обработчик на открытие файла |
| Нажатие на любую из двух кнопок сортировки в первом окне без данных в таблице | Отсутствие действий. Ничего не происходит | Невозможно отсортировать пустую таблицу | Class Controller,  Methods sortByNumber,  sortByMisses | Добавлен обработчик на считывание массива, который нужно отсортировать. Если он пустой – сообщение.  (Примечание – заполняется при открытии файла) |

# 

# 3 Заключение

В ходе выполнения данного курсового проекта был спроектирован и реализован программный продукт «Shell sort», который позволяет наглядно показать базовый алгоритм сортировки Шелла, а также работать с внешними Excel файлами, сортируя их данной сортировкой. В приложении также есть графика (анимация форм).

При разработке приложения была достигнуты главные цели: изучены парадигмы ООП и получен опыт и навыки разработки приложений на языке программирования Java.

# Список использованных источников

1. Официальный сайт документации для Java SE 8, компания Oracle [Электронный ресурс] - <http://docs.oracle.com/javase/8/>

2. Официальный сайт Java [Электронный ресурс] – http://www.java.com/en/about/

3. Н.Вирт – Алгоритмы и структуры данных, издание М.: Мир, 1989, 360 с.

4. Видео ресурс обучения работы с Java и Excel [Электронный ресурс] - <https://www.youtube.com/playlist?list=PLwcDaxeEINafif17no5JAO0iAi9Gw4g6H>

5. Microsoft-API библиотеки для работы с Excel [Электронный ресурс] – <http://www.apache.org/dyn/closer.lua/poi/release/bin/poi-bin-3.14.zip>

6. Статья о многопоточном программировании на электронном ресурсе Habrahabr [Электронный ресурс] - https://habrahabr.ru/post/164487/

Приложение А

(обязательное)

ИСХОДНЫЙ ПРОГРАММНЫЙ КОД

**Класс Main:**

package Main;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
  
  
// For work with JavaFX we need to extend a class Application  
public class Main extends Application {  
  
 public void start(Stage primaryStage) throws Exception {  
  
 //Here is loaded Main.fxml - file of marking which we edit using framework  
 //JavaFX SceneBuilder 2.0  
 Parent root = FXMLLoader.*load*(getClass().getResource("Main.fxml"));  
  
 //Passed in parameter object primaryStage is our main window  
 primaryStage.setTitle("Shell sort");  
 primaryStage.setScene(new Scene(root, 600, 400));  
 primaryStage.show();  
 }  
  
 public static void main(String[] args) {  
  
 //Our launch point. This method raises the FX stack and download our application in it  
 *launch*(args);  
  
 }  
}

**Класс Pupil:**

package Main;  
  
public class Pupil {  
 private int id;  
 private String name;  
 private double misses;  
  
 public Pupil(int id, String name, double misses) {  
 this.id = id;  
 this.name = name;  
 this.misses = misses;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public double getMisses() {  
 return misses;  
 }  
}

**Класс Controller:**

package Main;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.control.\*;  
import javafx.scene.paint.Color;  
import javafx.scene.shape.Rectangle;  
import javafx.scene.text.Text;  
import javafx.concurrent.Task;  
import org.apache.poi.ss.usermodel.Workbook;  
import org.apache.poi.xssf.usermodel.XSSFWorkbook;  
  
import java.io.FileInputStream;  
import java.io.IOException;  
  
public class Controller {  
  
 ////////////////////////  
 //Tab "Работа с Excel"//  
 ////////////////////////  
  
 //Fields  
  
 //This data will be loaded to ListView object  
 private ObservableList<Pupil> data = FXCollections.*observableArrayList*();  
  
 //This array for sort  
 private Pupil[] arrayPupil;  
  
 @FXML  
 //It is our ListView on form  
 private ListView<String> list;  
  
 //Methods  
  
 @FXML  
 void openExcel(ActionEvent event) throws IOException {  
  
 //Checking for correctness  
 try {  
 new FileInputStream("Main.xlsx");  
 } catch (Exception err) {  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*);  
  
 alert.setTitle("Упс!");  
 alert.setHeaderText(null);  
 alert.setContentText("Не получается открыть файл. Возможны следующие проблемы:\nФайл отсутствует" +  
 "\nФайл поврежден\nФайл имеет неверный формат\n\nУбедитесь, что файл лежит в одной директории " +  
 "с программой и имеет название и формат Main.xlsx");  
  
 alert.showAndWait();  
 }  
  
 //Open file and assign with WorkBook  
 FileInputStream fin = new FileInputStream("Main.xlsx");  
 Workbook wb = new XSSFWorkbook(fin);  
  
 //Counter  
 int n = 1;  
  
 while (true) {  
 try {  
 //Our tabulated values  
 int id = (int) wb.getSheetAt(0).getRow(n).getCell(0).getNumericCellValue();  
 String name = wb.getSheetAt(0).getRow(n).getCell(1).getStringCellValue();  
 double misses = wb.getSheetAt(0).getRow(n).getCell(2).getNumericCellValue();  
  
 Pupil pupil = new Pupil(id, name, misses);  
  
 data.add(pupil);  
  
 n++;  
 } catch (Exception er) {  
 n--;  
 break;  
 }  
 }  
  
 arrayPupil = new Pupil[n];  
  
 for (int i = 0; i < n; i++) {  
 String s = data.get(i).getId() + " - " + data.get(i).getName() + " Процент пропусков: " +  
 data.get(i).getMisses();  
  
 list.getItems().add(s);  
  
 arrayPupil[i] = data.get(i);  
 }  
  
 fin.close();  
 }  
  
 @FXML  
 void sortByNumber() {  
  
 list.getItems().clear();  
  
 if (data.isEmpty()){  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*);  
  
 alert.setTitle("Упс!");  
 alert.setHeaderText(null);  
 alert.setContentText("Кажется вы еще не загрузили файл! Попробуйте еще раз после загрузки файла");  
  
 alert.showAndWait();  
 } else {  
 //Shell sort  
 int increment = arrayPupil.length / 2;  
  
 while (increment > 0) {  
  
 for (int i = increment; i < arrayPupil.length; i++) {  
  
 int j = i;  
 Pupil temp = arrayPupil[i];  
  
 while (j >= increment && arrayPupil[j - increment].getId() > temp.getId()) {  
 arrayPupil[j] = arrayPupil[j - increment];  
 j = j - increment;  
 }  
  
 arrayPupil[j] = temp;  
 }  
 increment \*= 0.5;  
 }  
  
 //Output to the list  
 for (int i = 0; i < arrayPupil.length; i++) {  
 String s = arrayPupil[i].getId() + " - " + arrayPupil[i].getName() + " Процент пропусков: " +  
 arrayPupil[i].getMisses();  
  
 list.getItems().add(s);  
 }  
 }  
 }  
  
 @FXML  
 void sortByMisses() {  
  
 list.getItems().clear();  
  
 if (data.isEmpty()) {  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*);  
  
 alert.setTitle("Упс!");  
 alert.setHeaderText(null);  
 alert.setContentText("Кажется вы еще не загрузили файл! Попробуйте еще раз после загрузки файла");  
  
 alert.showAndWait();  
 } else {  
 //Shell sort  
 int increment = arrayPupil.length / 2;  
  
 while (increment > 0) {  
  
 for (int i = increment; i < arrayPupil.length; i++) {  
  
 int j = i;  
 Pupil temp = arrayPupil[i];  
  
 while (j >= increment && arrayPupil[j - increment].getMisses() > temp.getMisses()) {  
 arrayPupil[j] = arrayPupil[j - increment];  
 j = j - increment;  
 }  
  
 arrayPupil[j] = temp;  
 }  
 increment \*= 0.5;  
 }  
  
 //Output to the list  
 for (int i = 0; i < arrayPupil.length; i++) {  
 String s = arrayPupil[i].getId() + " - " + arrayPupil[i].getName() + " Процент пропусков: " +  
 arrayPupil[i].getMisses();  
  
 list.getItems().add(s);  
 }  
 }  
  
 }  
  
 /////////////////////////////////  
 //Tab "Визуализация сортировки"//  
 /////////////////////////////////  
  
 //Fields  
  
 @FXML  
 //Values of array  
 private Text t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12;  
  
 @FXML  
 //Rectangles  
 private Rectangle rt1, rt2, rt3, rt4, rt5, rt6, rt7, rt8, rt9, rt10, rt11, rt12;  
  
 @FXML  
 //Step of sorting as a Label for clarity  
 private Label dLabel;  
  
 //Methods  
  
 @FXML  
 //Shell sort  
 void startVisualisation(ActionEvent event) {  
  
 //We have 12 rectangles with 12 Text objects  
 Text[] arrayText = {t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12};  
 Rectangle[] arrayRectangle = {rt1, rt2, rt3, rt4, rt5, rt6, rt7, rt8, rt9, rt10, rt11, rt12};  
  
 //Clean our data  
 for (int i = 0; i < 12; i++) {  
 arrayText[i].setText("");  
 arrayRectangle[i].setFill(Color.*DODGERBLUE*);  
 }  
  
 //And clean D step label  
 dLabel.setText("");  
  
 /\*As long as our code is executed, redrawing is not satisfied, because the flow is only one  
 \*and at the moment (when it's on a cycle), it performs just your code.  
 \*He may not redraw elements, as long as our code will not work.  
 \*Solution - to run our code in a separate thread\*/  
  
 //This is special task for parallel thread  
 Task<Void> task = new Task<Void>() {  
  
 @Override  
 public Void call() {  
 try {  
 //Main code  
  
 //Initialize array  
 for (int i = 0; i < arrayText.length; i++, Thread.*sleep*(250)) {  
  
 //Math.random() - [0,1); right border is not strict  
 //On typecast there will be a maximum value equal 12 (Fraction of a number will be removed)  
 int a = (int) (Math.*random*() \* 13);  
 String a\_str = Integer.*toString*(a);  
  
 arrayText[i].setText(a\_str);  
 }  
  
 //The beggining of sort  
  
 //Looking for a initial value  
 int increment = arrayText.length / 2;  
 String dLabelMessage = Integer.*toString*(increment);  
  
 //Transfer it to main thread  
 updateMessage(dLabelMessage);  
  
 while (increment > 0) {  
  
 //Reset color of label  
 dLabel.setTextFill(Color.*BLACK*);  
  
 for (int i = increment; i < arrayText.length; i++) {  
 int j = i;  
  
 //Checking sorted  
 boolean less = true;  
  
 //Objects on right half of massive  
 int buffer = Integer.*parseInt*(arrayText[i].getText());  
  
 //Increment less && wrong sorting  
 while (j >= increment && Integer.*parseInt*(arrayText[j - increment].getText()) > buffer) {  
  
 //Selected object will be a YELLOW  
 //Pair object will be a RED because sorting is wrong  
 arrayRectangle[j].setFill(Color.*YELLOW*);  
 arrayRectangle[j - increment].setFill(Color.*RED*);  
  
 //Always will be wait for a second  
 Thread.*sleep*(1000);  
  
 //Wrong sorted object moved to right position  
 //Right position - RED  
 //Previous position - standart DODGERBLUE  
 arrayRectangle[j].setFill(Color.*RED*);  
 arrayRectangle[j - increment].setFill(Color.*DODGERBLUE*);  
  
 //Swap of values  
 arrayText[j].setText(arrayText[j - increment].getText());  
 arrayText[j - increment].setText(Integer.*toString*(buffer));  
  
 //Wait  
 Thread.*sleep*(1000);  
  
 //Return to position[j] standart color  
 arrayRectangle[j].setFill(Color.*DODGERBLUE*);  
  
 //Next step. Always with 'increment' variable  
 j = j - increment;  
  
 //Flag wrong sorting  
 less = false;  
 }  
  
 //If sort right  
 if (less) {  
  
 //Only show colors YELLOW (selected) and GREEN (previous)  
 //It means that all correct  
 arrayRectangle[j].setFill(Color.*YELLOW*);  
 arrayRectangle[j - increment].setFill(Color.*GREEN*);  
 Thread.*sleep*(1000);  
  
 //Return standart colors  
 arrayRectangle[j].setFill(Color.*DODGERBLUE*);  
 arrayRectangle[j - increment].setFill(Color.*DODGERBLUE*);  
 }  
 }  
  
 //Decrease variable in half  
 increment \*= 0.5;  
  
 //Clean colors  
 for (int i = 0; i < arrayRectangle.length; i++) {  
 arrayRectangle[i].setFill(Color.*DODGERBLUE*);  
 }  
  
 //Highlighted a dLabel with RED color  
 dLabel.setTextFill(Color.*RED*);  
  
 //Transfer a value  
 dLabelMessage = Integer.*toString*(increment);  
 updateMessage(dLabelMessage);  
  
 //Wait  
 Thread.*sleep*(1000);  
 }  
 } catch (InterruptedException ex) {  
 }  
 return null;  
 }  
 };  
  
 //Action on get a message to dLabel from second thread  
 task.messageProperty().addListener((observable, oldMessage, newMessage) -> dLabel.setText(newMessage));  
  
 //Start a new thread with task settings  
 Thread th = new Thread(task);  
 th.start();  
  
 //Visualisation done  
 }  
}

Приложение Б

(справочное)

ТЕКСТ СПРАВКИ

Программное обеспечение Shell sort.

Версия программы: 1.0

СПРАВКА Shell sort:

Shell sort используется для как учебное пособие для наглядного отображения

алгоритма сортировки Шелла. Визуальное отображение, построенное на

основе алгоритма сортировки Шелла, использует в качестве сортировки

визуальные объекты - квадраты, внутри которых записаны их значения.

Дополнительной и отдельной задачей проекта является работа с файлами Excel.

Эксплуатация программы Shell sort:

Интерфейс программы достаточно прост и интуитивно понятен.

На вкладе "Работа с Excel" вы найдете всего 3 кнопки:

"Загрузить" - что значит загрузить данные из файла Main.xlsx в таблицу на форме.

"По номеру" - сортировка загруженных данных по номеру при помощи сортировки Шелла.

"По пропускам" - сортировка загруженных данных по пропускам при помощи сортировки Шелла.

На вкладке "Визуализация сортировки" кнопка старт запустит наглядную

визуализацию сортировки Шелла с 12 различными числами, которые случайно генерируются при каждом новом нажатии кнопки.

На вкладке "Справка" вы найдете краткое описание программы.