

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по учебной практике
ТЕМА: Визуализация алгоритмов на языке Java

Студент гр. 6383	_____	Яковлев Е. А.
Студент гр. 6382	_____	Клименко К. В.
Студентка гр. 6304	_____	Иванкова В. М.
Руководитель	_____	Фирсов М. Ю.

Санкт-Петербург
2018

ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студент Яковлев Е.А. группы 6383

Студентка Клименко К.В. группы 6382

Студентка Иванкова В.М. группы 6304

Тема практики: визуализация алгоритмов на языке Java

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: Поразрядная сортировка

Сроки прохождения практики: 27.06.2018 – 10.07.2018

Дата сдачи отчета: 06.07.2018

Дата защиты отчета: 06.07.2018

Студент	_____	Яковлев Е. А.
Студент	_____	Клименко К. В.
Студентка	_____	Иванкова В. М.
Руководитель	_____	Фирсов М. Ю.

АННОТАЦИЯ

В данной работе изучается алгоритм поразрядной сортировки и реализуется его визуализация. Программа разработана на языке Java, для создания графического интерфейса используется библиотека JavaFX. Проект выполнен в среде IntelliJ IDEA.

Основной целью разработки проекта является получение навыков работы в команде: важно умение распределить обязанности между участниками, работающими над проектом, продумать план выполнения и согласовать процесс так, чтобы закончить работу к необходимому сроку.

Программа имеет понятный и удобный для пользователя интерфейс, который наглядно демонстрирует сортировку введенных или сгенерированных элементов. Работоспособность проекта проверена тестами.

СОДЕРЖАНИЕ

Введение.....	5
1. Постановка задачи.....	6
1.1. Формулировка задания.....	6
1.2. Теоретические сведения.....	6
1.3. Реализуемый алгоритм.....	6
2. Спецификация.....	7
2.1. Описание интерфейса.....	Ошибка! Закладка не определена.
3. План разработки и распределение ролей в бригаде.....	7
3.1. План разработки.....	11
3.2. Распределение ролей в бригаде.....	12
4. Особенности реализации.....	13
4.1. Используемые структуры данных.....	13
5. Тестирование.....	18
5.1. Тестирование работы алгоритма.....	18
5.2. Тестирование интерфейса программы.....	19
Заключение.....	25
Список использованных источников.....	26
Приложение А. Код программы.....	27
1. Main.java.....	27
2. Controller.java.....	27
3. MWController.java.....	35
4. IntRadixSorter.java.....	41
5. StringRadixSorter.java.....	44
6. RadixSorter.java.....	47

ВВЕДЕНИЕ

Цели выполнения учебной практики:

1. Освоение среды программирования Java, особенностей и синтаксиса данного языка.
2. Изучение средств для реализации графического интерфейса, а именно – библиотеки JavaFX.
3. Получение таких навыков как:
 - разработка программ на языке Java;
 - работа с системой контроля версий, репозиториями;
 - командная работа.

Поставленные цели будут достигнуты во время выполнения проекта (визуализации поразрядной сортировки), выбранного бригадой.

Поразрядная сортировка — алгоритм сортировки, который выполняется за линейное время.

1. ПОСТАНОВКА ЗАДАЧИ

1.1. Формулировка задания

Разработать программу, визуализирующей алгоритм поразрядной сортировки на языке программирования Java.

1.2. Теоретические сведения

Исходно предназначен для сортировки целых чисел, записанных цифрами. Но так как в памяти компьютеров любая информация записывается целыми числами, алгоритм пригоден для сортировки любых объектов, запись которых можно поделить на «разряды», содержащие сравнимые значения. Например, так сортировать можно не только числа, записанные в виде набора цифр, но и строки, являющиеся набором символов, и вообще произвольные значения в памяти, представленные в виде набора байт.

1.3. Реализуемый алгоритм

Сравнение производится поразрядно: сначала сравниваются значения одного крайнего разряда, и элементы группируются по результатам этого сравнения, затем сравниваются значения следующего разряда, соседнего, и элементы либо упорядочиваются по результатам сравнения значений этого разряда внутри образованных на предыдущем проходе групп, либо переупорядочиваются в целом, но сохраняя относительный порядок, достигнутый при предыдущей сортировке. Затем аналогично делается для следующего разряда, и так до конца.

2. СПЕЦИФИКАЦИЯ

2.1. Требования к программе

Программа должна выполнять сортировку элементов типа Integer (до 1000 элементов от 0 до `maxInt`) и String (до 1000 элементов по 128 латинских букв и цифр) поразрядным методом (Radix Sort) и осуществлять визуализацию работы данного алгоритма. Она должна обладать графическим интерфейсом пользователя (далее GUI) с возможностью ручного ввода элементов для сортировки и случайной генерации заданного количества элементов для автоматического ввода.

Для визуализации перемещения элементов по колонкам, текущий сортируемый элемент визуально выносится за пределы массива и имеет более крупный размер шрифта для отображения. На следующем шаге он переносится в нужную колонку, которая в свою очередь выделяется цветом. При этом, на его место встает следующий элемент рабочего массива.

Программа предполагает три режима работы:

Автоматический: программа сразу сортирует введенные числа и выводит конечный результат;

Пошаговый: в этом режиме сортировка разделяется на шаги (например, перенос одного числа из рабочего массива в специальный разрядный массив), которые выполняются строго по очереди и при нажатии соответствующей клавиши. Возможно выполнение шагов как вперед, так и назад. При этом, актуальное промежуточное состояние рабочих массивов отображается на экране;

Плавный: шаги работы алгоритма выполняются автоматически через заданные промежутки времени (например, одну секунду).

Способы ввода данных:

Ручной: ввод данных в поле ввода данных осуществляется с клавиатуры.

Ввод из файла: данные поступают с файла в формате *.txt, ввод осуществляется по нажатию на соответствующую кнопку, при этом данные будут дублированы в поле ручного ввода, для возможности их редактирования.

Генерация случайных данных: при нажатии на кнопку генерации производится заполнение поля ввода данных случайными числами по параметрам для генерации: количество чисел, диапазон данных.

Демонстрация текущего шага:

Поле комментария: отдельное поле для демонстрации комментариев по текущим событиям, заполняется автоматически после выполнения «шага».

Подсветка действия: выделение цветом разряда и колонки, с которыми производится работа на текущем действии.

Анимация движений:

Анимация перемещения: плавное перемещение элемента на нужную позицию.

2.2. Анализ задачи

Данная программа будет состоять из двух базовых классов:

класс, отвечающий за GUI и обеспечивающий работу основной логики программы;

класс, реализующий разработку алгоритма поразрядной сортировки.

В самом начале оба класса будут разрабатываться отдельно, затем они будут связаны между собой при помощи предварительно описанных интерфейсов (под интерфейсом в данном случае подразумевается набор методов класса, разрешенных для использования извне).

Отвечающий за GUI класс будет содержать, в частности, методы, служащие для проецирования внутреннего состояния рабочих массивов на графические элементы главного окна программы.

Таким образом, логика работы каждого класса останется только в соответствующем классе и никак не будет зависеть от внешней среды.

Если понадобится, будут реализованы вспомогательные классы, упрощающие разработку некоторых частей программы.

2.3. Интерфейс

Макет представлен на рис. 1

Введите числа:

Чтобы заполнить случайными числами, введите необходимые данные:
Генерировать числа от 0 до: Количество чисел:

32 543 754 349 2332 95 3 764

Текущий разряд сортировки: 1

540 110					15 645 115				
0	1	2	3	4	5	6	7	8	9

Комментарии к выполнению:

Рис. 1

2.4. Уточнения

В последнюю версию программы были внесены уточнения:

1. Необходимость добавить генерацию по интервалу, задаваемого пользователем.
2. Разделение главных итераций (разрядов) в поле комментариев для большей наглядности.

3. Сделать окно пошагового вывода адаптивным от размера ВХОДНЫХ данных.

3. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

3.1. План разработки

Разработка программы будет вестись с использованием языка программирования Java и его мощных возможностей по созданию GUI. Для упрощения написания программы используется интегрированная среда разработки (далее IDE) IntelliJ IDEA.

Для обеспечения доступа к исходному коду всех участников проекта и более гибкой работы с ним используется система контроля версий Git.

Репозиторий с исходным кодом и данной спецификацией расположен на крупнейшем веб-сервисе для хостинга IT-проектов GitHub по адресу: <https://github.com/EgorYackovleff/RadixSort.git>.

План разработки программы разбит на следующие шаги:

1. Создание прототипа: к этому шагу будет создан класс, отвечающий за GUI, и реализован некоторый интерфейс к нему. Результатом работы на данном шаге будет являться макет GUI с неработающими элементами, служащий только для демонстрации;

2. Первая версия: на этом шаге будет реализован класс, отвечающий за работу самого сортировочного алгоритма. Затем он будет соединен с классом GUI, используя интерфейсы обоих классов. В результате получится рабочая программа, которая, тем не менее, может содержать некоторые недоделанные компоненты или баги, также будет реализован ввод из файла, добавление окна с комментариями на текущий шаг.

3. Вторая версия: реализация пошаговой работы программы.

4. Третья (конечная) версия: добавление возможности генерация входных данных, анимация движения, подсветка текущего действия. Во время последней итерации будет вестись работа над исправлением ошибок, неочевидными моментами в работе программы и небольшими улучшениями.

3.2. Распределение ролей в бригаде

Роли в бригаде распределены следующим образом:

- Яковлев Егор: реализация алгоритма сортировки внутри класса и методов взаимодействия с ним; написание отчета.
- Иванкова Валерия: написание спецификации программы; проектирование системы классов и их взаимодействия; связывание готовых классов программы; написание отчета.
- Клименко Кирилл: реализация GUI и методов взаимодействия с ним; написание отчета.

4. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

4.1. Используемые структуры данных

Поля:

private Vector<Canvas> vectorCanvas - Вектор, хранящий холсты для рисования колонок

private Vector<ScrollPane> vectorScrollPane - Вектор, хранящий панели скроллинга для колонок

private Vector<VBox> vectorVBox - Вектор, хранящий вертикальные слои для колонок

private ScrollPane prevMark - Хранит панель скроллинга, в которую записывались данные на предыдущем шаге (нужно для того, чтобы убирать подсветку с предыдущего столбца)

private int canvasLength - Размер холста в колонке

static int CanvasWidth - Размер холста в колонке

int spaceFont - Отступ между строками при выводе данных

int fontSize - Размер шрифта для рисования

private TextField TextFieldTill - Строка ввода данных максимального числа для генерации

private TextField TextFieldTillOt - Строка ввода данных минимального числа для генерации

private TextField TextFieldNum - Строка ввода данных количества чисел для генерации

private Label DigitLabel - Строка вывода текущего разряда

private TextField Text_Field - Строка ввода данных

private HBox MainHBox - Переменная, хранящая слой

private ScrollPane scrollPaneLow - Переменная, хранящая панель скроллинга, в которой располагаются все столбцы

private TextField TextFieldTillStr - Строка ввода данных максимального количества символов в строке для генерации

private TextField TextFieldNumStr - Строка ввода данных количества строк для генерации

private Canvas canvasArea - Холст, для рисования всего массива

class SortVis – класс, содержащий точку входа в программу.

Методы:

public static void main(String[] args)

public void onNextStepButtonClicked(ActionEvent event) – обработка нажатия клавиши "Следующий шаг"

public void onPreviousStepButtonClicked(ActionEvent event) - обработка нажатия клавиши "Предыдущий шаг"

public void onLoadFromFileClicked(ActionEvent event) throws IOException - обработка нажатия клавиши предназначенной для считывания файла

public void onEnterDataClicked(ActionEvent event) - обработка нажатия клавиши "Ввести данные"

public static void infoBox(String infoMessage, String titleBar) – вывод окна комментариев

public void onAutoButtonClicked(ActionEvent event) - обработка нажатия клавиши "Автоматически/Пауза"

public void changeButton(boolean value) - true - установить "Автоматически" false - установить "Приостановить"

public void onClearButtonClicked(ActionEvent event) - обработка нажатия клавиши "Сброс"

public void onFinishSortingButtonClicked(ActionEvent event) - обработка нажатия клавиши "Завершить сортировку"

public void setCanvases(int numCanvases) - Устанавливает количество столбцов с холстами для вывода по разрядам (37 столбцов для строк, 10 столбцов для чисел)

`public void onFillingButtonClicked(ActionEvent event)` - обработка нажатия клавиши "Заполнить"

`public void updateCurrentDigit(int digit)` - Выводит строку с текущим разрядом сортировки

`public void onExitButtonClicked(ActionEvent event)` - обработка нажатия клавиши выхода

`public void updateWorkingArrayInteger(Vector<Integer> array)` - Выводит на экран содержание всего массива чисел, подлежащих сортировке

`public void updateWorkingArrayString(Vector<String> array)` - Выводит на экран содержание всего массива строк, подлежащих сортировке

`public void updateComponentsArrayInteger(Vector<Integer> array, int column, int digit)` - Выводит в столбец `column` числа и подсвечивает разряд `digit`

`public void updateComponentsArrayString(Vector<String> array, int column, int digit)` - Выводит в столбец `column` строки и подсвечивает разряд `digit`

`public void lightColumn(int column)` - Подсветка столбца `column`

`public void setMode(Mode m)` - Устанавливает режим сортировки - числа или строки, `m` - название режима

`public void setControl(Controller c)` - Устанавливает контроллер для данного класса

`public void memorize(IntRadixSorter sorter)` - Запоминает промежуточные состояния сортировки, `sorter` - сортировщик целых чисел

`public void memorize(StringRadixSorter sorter)` - Запоминает промежуточные состояния сортировки, `sorter` - сортировщик строк

`public void clearSaves()` - Очищает все сохраненные состояния сортировщиков

`public IntRadixSorter getIntRadixSorter()` - Отдает последний использовавшийся сортировщик целых чисел

`public StringRadixSorter getStringRadixSorter()` - Отдает последний использовавшийся сортировщик строк

`public boolean rewind()` - Отматывает состояние сортировщика назад на одну операцию

`public int onNextStepButtonClicked()` –

`public void onPreviousStepButtonClicked()` - обработка нажатия клавиши "Предыдущий шаг"

`public int onEnterDataClicked(String str)` { // обработка нажатия клавиши "Ввести данные"

`public void sortNumbers(Vector<Integer> arr)` – вызов функции сортировки чисел, вывод сообщений в окно коминтариев

`public void sortStrings(Vector<String> arr)` - вызов функции сортировки строк, вывод сообщений в окно коминтариев

`public void onFinishSortingButtonClicked()` - Если нажата клавиша "Завершить сортировку"

`public void load(Vector<Integer> array)` - Загружает массив для сортировки внутрь класса, array - массив, который нужно отсортировать

`public Vector<Integer> getWorkingArray()` - Возвращает рабочий массив

`public Vector<Integer> getCategoryArray(int cat)` - Возвращает массив заданной колонки, cat - номер колонки для возврата

`public int doStep()` - Делает один шаг сортировки. По окончании сортировки возвращает false.

`public void clear()` - Очищает рабочее пространство класса

`hasDigits(Integer number)` - Возвращает количество цифр в числе, number - число, в котором нужно посчитать количество цифр

`private int digitAt(Integer number, int pos)` - Возвращает цифру на заданной позиции, number – число, pos - разряд, цифру которого нужно получить

`private int maxDigits()` - Возвращает длину максимального числа в массиве

`public void load(Vector<String> array)` - Загружает массив для сортировки внутрь класса, array - массив, который нужно отсортировать

`public Vector<String> getWorkingArray()` - Возвращает рабочий массив

`public Vector<String> getCategoryArray(int cat)` - Возвращает массив заданной колонки, cat - номер колонки для возврата

`public int doStep()` - Делает один шаг сортировки. По окончании сортировки возвращает false

`public void clear()` - Очищает рабочее пространство класса

`private int maxDigits()` - Возвращает длину максимального числа в массиве

`private static int matchingColumn(char c)` - Возвращает номер колонки для определенного символа, c - символ

5. ТЕСТИРОВАНИЕ

5.1. Тестирование работы алгоритма

Тест №1 (Для чисел)

Исходные данные: 19 856 36 65342 73 82 855 4245 63764 3052 3

Выходные данные:

3 19 36 73 82 855 856 3052 4245 63764 65342

Рис. 11 - Выходные данные для теста №1

Тест №2 (Для чисел)

Исходные данные: 6 224545 607 99 9672 789628 543151 1 4167 910 247
4645 36 3536 859 4156 858 958

Выходные данные:

1 6 36 99 247 607 858 859 910 958 3536 4156
4167 4645 9672 224545 543151 789628

Рис. 12 - Выходные данные для теста №2

Тест №3 (Для чисел)

Исходные данные: 900 8 4 1 56 2 11 111 77 7 45 841

Выходные данные:

1 2 4 7 8 11 45 56 77 111 841 900

Рис. 13 - Выходные данные для теста №3

Тест №4 (Для строк)

Исходные данные: j qxolr o e8fx xcf itn5 rwpp ic fin pl

Выходные данные:

e8fx fin ic itn5 j o pl qxolr rwpp xcf

Рис. 12 - Выходные данные для теста №4

Тест №5 (Для строк)

Исходные данные: a aa b c d ddd e f g hh i z

Выходные данные:



a aaa b bb c d dd e ee eee j z

Рис. 13 - Выходные данные для теста №5

Тест №6 (Для строк)

Исходные данные: pdlalis temensu qesoril sgahl psl0t yc9a x wolkgfx

Выходные данные:



pdlalis psl0t qesoril sgahl temensu wolkgfx x yc9a

Рис. 14 - Выходные данные для теста №6

В результате проведенного тестирования неправильной работы интерфейса или алгоритма выявлено не было.

5.2. Тестирование интерфейса программы

При запуске программы открывается окно программы, имеющее интерфейс, представленный на рисунке 1.

Введите символы: 2531 9337 610 bdnuncr hmg t2i8b ... Ввести данные

Чтобы заполнить рандомно, введите необходимые данные:

Генерировать числа от 0 до 10000 Количество чисел: 3 Макс. кол-во символов: 8 Количество строк: 3

Заполнить

Предыдущий шаг Автоматически Следующий шаг Завершить сортировку Сброс Выход

Рис.1 - Интерфейс программы

Чтобы начать работу с программой, нужно ввести данные. Сделать это можно двумя способами. Непосредственно ввести в строку ввода символов данные или сгенерировать случайную последовательность с помощью встроенного генератора. Чтобы воспользоваться генератором чисел, введем соответствующие данные, затем нажмем кнопку «Заполнить». Результат операции приведен на рисунке 2.

Введите символы: 2531 9337 610 bdnuncr hmg t2i8b ... Ввести данные

Чтобы заполнить рандомно, введите необходимые данные:

Генерировать числа от 0 до 10000 Количество чисел: 3 Макс. кол-во символов: 8 Количество строк: 3

Заполнить

Рис. 2 - Ввод данных

Для загрузки введенных данных необходимо нажать кнопку «Ввести данные». Результат этого действия отражает рисунок 3.

Введите символы: 490 3682 6066 afyutd dte7e2n h87 Ввести данные

Чтобы заполнить случайно, введите необходимые данные:

Генерировать числа от 0 до 10000 Количество чисел: 3 Макс. кол-во символов: 8 Количество строк: 3 Заполнить

Текущий разряд сортировки: 1

490 3682 6066 afyutd dte7e2n h87

\$	0	1	2	3	4	5

Предыдущий шаг Автоматически Следующий шаг Завершить сортировку Сброс Выход

6066\$\$\$\$
afyutd\$
dte7e2n
h87\$\$\$\$\$
sorting strings [490\$\$\$\$\$, 3682\$\$\$\$\$, 6066\$\$\$\$\$, afyutd\$, dte7e2n, h87\$\$\$\$\$]

Рис. 3 – После нажатия кнопки «Ввести данные»

Как видно из рисунка 3, после нажатия кнопки, в соответствующем поле отображаются введенная ранее последовательность символов, под этим полем появляются столбцы, по которым позже, в процессе выполнения программы, будут рассортированы введенные данные, а ещё ниже вывелись данные в окне комментариев

Программа предполагает несколько режимов работы:

Пошаговое выполнение посредством нажатия кнопки «Следующий шаг».

Автоматическое выполнение при нажатии кнопки «Автоматически». В данном режиме программа делает каждый шаг с временной задержкой, равной 1 секунде. Также возможна приостановка программы в текущем режиме, нажатием той же кнопки, которая в этот момент меняет своё название на «Пауза»

Выдача конечного результата сортировки посредством нажатия кнопки «Завершить сортировку»

Программа отменяет предыдущий шаг действие при нажатии кнопки «Предыдущий шаг»

Кнопка «Сброс» позволяет очистить все поля и вернуть программу в начальное состояние

Кнопка «Выход» закрывает приложение.

Текущий разряд сортировки подсвечивается красным цветом.

Введем последовательность чисел, изображенную на рисунке 4 и нажмем кнопку «Ввести данные»



Введите символы: 7 85 943 64 17 23

Рис. 4 - Ввод чисел

Будем последовательно выполнять программу, нажатием кнопки «Следующий шаг». Увидим, как числа занимают свои колонки, при сортировке по 1-ому разряду. Результат изображен на рисунке 5.

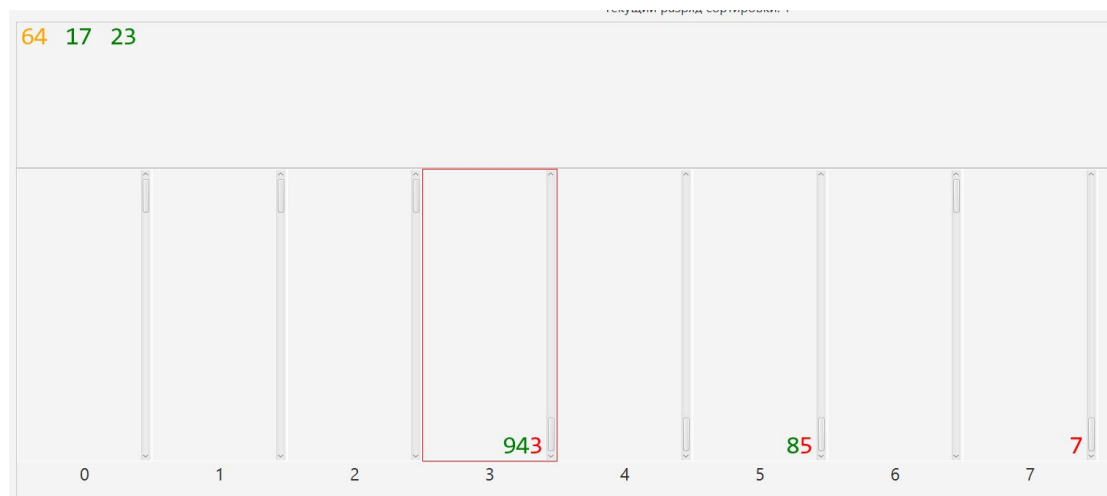


Рис. 5 - После нажатия кнопки «Следующий шаг»

Затем нажмем кнопку «Автоматически» и подождем, пока алгоритм разместит еще 2 числа. Результат приведен на рисунке ниже.

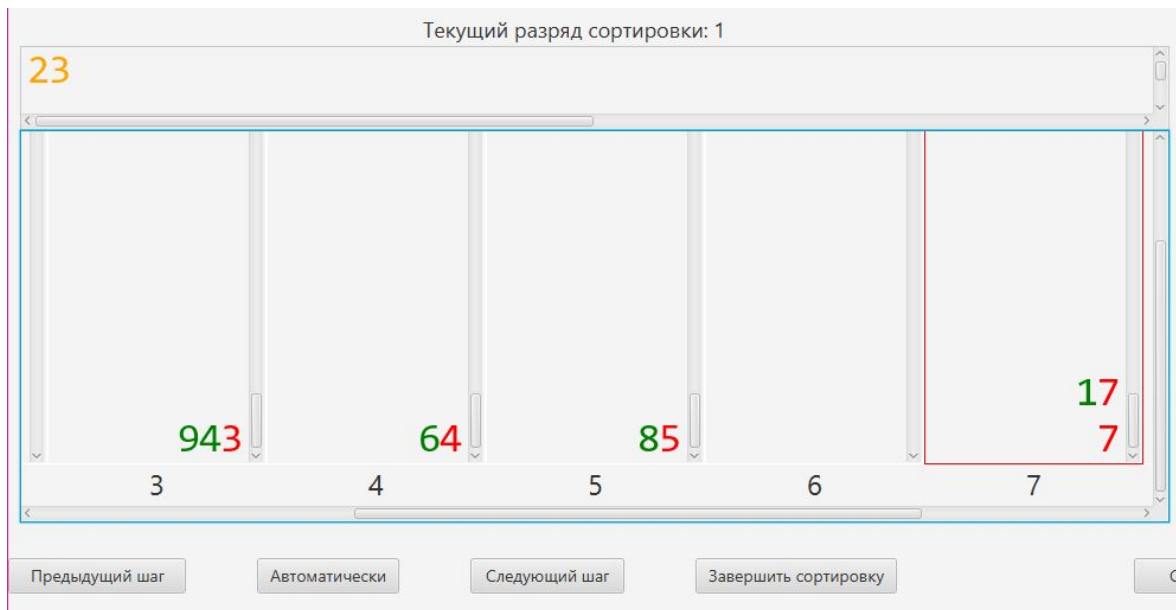


Рис. 6 – после нажатия кнопки «Автоматически»

Теперь два раза нажмем на кнопку «Шаг назад». То есть мы должны вернуться в состояние как на рисунке 5. Результат этого действия приведен на рисунке 7.

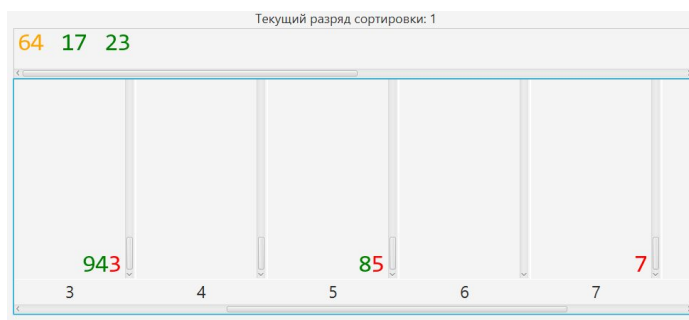


Рис. 7 – После двойного нажатия на «Предыдущий шаг»

Теперь нажмем на кнопку «Завершить сортировку» и посмотрим на финальный результат сортировки – рисунок 8.



Рис.8 – Результат сортировки

После проделанных операций, для очистки содержимого окна нажмем на кнопку «Сброс». Результат этого действия продемонстрирован на рисунке 9.

Введите символы:

Чтобы заполнить рандомно, введите необходимые данные:

Генерировать числа от до Количество чисел: Макс. кол-во символов: Количество строк:

afyutd\$
dte7e2n
h87\$\$\$
sorting strings [490\$\$\$\$, 3682\$\$\$\$, 6066\$\$\$, afyutd\$, dte7e2n, h87\$\$\$\$]
Clearing saves!

Рис. 9 – После нажатия кнопки «Сброс»

Так как максимальная длина строки введенных данных составляет 8 символов, то при попытке превысить диапазон мы получим следующее сообщение, как на рисунке 10.

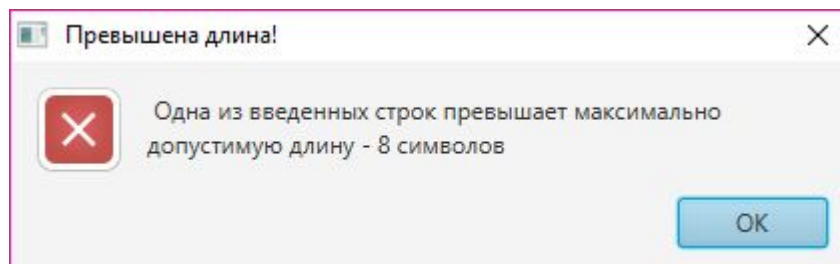


Рис. 10 – Сообщение о превышении длины

ЗАКЛЮЧЕНИЕ

В результате выполнения учебной практики была разработана и протестирована программа, реализующая алгоритм поразрядной сортировки и наглядно демонстрирующая принцип его работы.

Во время реализации проекта были проведены некоторые доработки интерфейса программы для улучшения её удобства.

В ходе работы было проведено тестирование с целью выявления возможных ошибок. По результатам было выяснено, что программа работает корректно и успешно справляется со своей задачей.

Выполнение данного проекта позволило приобрести навыки, необходимые для будущей профессии, тесно связанной с ИТ. Это навыки:

- разработки в среде программирования Java;
- работы в команде;
- использования известной системы контроля версий GitHub;
- использования библиотеки JavaFX для реализации графического интерфейса.

Таким образом, цели практики успешно достигнуты, и по окончании разработки получен корректно работающий визуализатор алгоритма поразрядной сортировки на языке программирования Java.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Эккель Б. Философия Java. 4-е изд. СПб.: Питер, 2014;
2. Дейтел Х.М., Дейтел П.Дж. Как программировать на Java. Кн. 1: Основы программирования. / пер. с англ. А.В. Козлова. 4-е изд. М.: Бином, 2003;
3. Седжвик Р., Уэйн К. Алгоритмы на Java, 4-е изд.: Пер. с англ. – М.: ООО “И.Д.Вильямс”, 2013. – 848 с.
4. Java. Базовый курс. // Stepik. URL: <https://stepik.org/course/Java-Базовый-курс-187/syllabus>;
5. Быстрая сортировка // Wikipedia URL: https://ru.wikipedia.org/wiki/Быстрая_сортировка.

ПРИЛОЖЕНИЕ А. КОД ПРОГРАММЫ

1. Main.java

```
package com.eltech.gui;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        Parent root =
FXMLLoader.load(getClass().getResource("sample.fxml"));
        primaryStage.setTitle("Radix Sort");
        primaryStage.setScene(new Scene(root, 900, 900));
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

2. Controller.java

```
package com.eltech.gui;

import javafx.animation.Animation;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.geometry.Pos;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.*;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.TextAlignment;
import javafx.util.Duration;

import javax.swing.*;
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.List;
import java.util.Random;
import java.util.Vector;
```

```

import java.util.stream.Collectors;

public class Controller {
    private Vector<Canvas> vectorCanvas; //Вектор, хранящий холсты для
рисования колонок
    private Vector<ScrollPane> vectorScrollPane; // Вектор, хранящий
панели скроллинга для колонок
    private Vector<VBox> vectorVBox; // Вектор, хранящий вертикальные
слои для колонок
    private ScrollPane prevMark = null; //Хранит панель скроллинга, в
которую записывались данные на предыдущем шаге (нужно для того, чтобы
убирать подсветку с предыдущего столбца)
    private int canvasLength = 3000; //Размер холста в колонке
    static int CanvasWidth = 10; //168; //Размер холста в колонке
    int spaceFont = 4; //Отступ между строками при выводе данных
    int fontSize = 35; //Размер шрифта для рисования
    int isFinished;

    MWController mainController = new MWController();

    @FXML
    private TextField TextFieldTill; // Строка ввода данных
максимального числа для генерации
    @FXML
    private TextField TextFieldTillOt; // Строка ввода данных
минимального числа для генерации
    @FXML
    private TextField TextFieldNum; // Строка ввода данных количества
чисел числа для генерации
    @FXML
    private Label DigitLabel; // Строка вывода текущего разряда
    @FXML
    private TextField Text_Field; // Строка ввода данных
    @FXML
    private HBox MainHBox; // Переменная, хранящая слой
    @FXML
    private ScrollPane scrollPaneLow; //Переменная, хранящая панель
скроллинга, в которой располагаются все столбцы
    @FXML
    private TextField TextFieldTillStr; //Строка ввода данных
максимального количества символов в строке для генерации
    @FXML
    private TextField TextFieldNumStr; //Строка ввода данных количества
строк для генерации
    @FXML
    private Canvas canvasArea; // Холст, для рисования всего массива
    @FXML
    private Button AutoButton;
    @FXML
    private TextArea textArea;

    @FXML
    void initialize() throws IOException {
        /*JFileChooser fileChooser = new JFileChooser();
        int openDialog = fileChooser.showOpenDialog(null);
        File selectedFile = fileChooser.getSelectedFile();
        List<String> strings =
Files.readAllLines(Paths.get(selectedFile.getPath()));
        String result = strings.stream().collect(Collectors.joining("
"));
        System.out.println(result);*/

        // TODO

```

```

    }

    public void printToTextArea(String string) {
        textArea.setText(textArea.getText() + string + "\n");
    }

    Vector<Integer> InitialArrayInt; //Вектор чисел исходных данных
    Vector<String> InitialArrayString; //Вектор строк исходных данных
    Timeline timeline = null;

    public Controller() {
        mainController.setControl(this);
    }

    public void onNextStepButtonClicked(ActionEvent event) { // Если
нажата клавиша "Следующий шаг"
        isFinished = mainController.onNextStepButtonClicked();
    }

    public void onPreviousStepButtonClicked(ActionEvent event) // Если
нажата клавиша "Предыдущий шаг"
    {
        mainController.onPreviousStepButtonClicked();
    }
    @FXML
    public void onLoadFromFileClicked(ActionEvent event) throws
IOException {
        Text_Field.clear();
        JFileChooser fileChooser = new JFileChooser();
        int openDialog = fileChooser.showOpenDialog(null);
        File selectedFile = fileChooser.getSelectedFile();
        List<String> strings =
Files.readAllLines(Paths.get(selectedFile.getPath()));
        String result = strings.stream().collect(Collectors.joining("
"));
        //System.out.println(result);
        Text_Field.appendText(result);
    }

    public void onEnterDataClicked(ActionEvent event) { // Если нажата
клавиша "Ввести данные"
        int mark = 0;
        String str = Text_Field.getText();
        mark = mainController.onEnterDataClicked(str);
        if (mark == 1) {
            Controller.infoBox(" Одна из введенных строк превышает
максимально допустимую длину - 8 символов", " Превышена длина!");
        }
    }

    public static void infoBox(String infoMessage, String titleBar) {
        infoBox(infoMessage, titleBar, null);
    }

    public static void infoBox(String infoMessage, String titleBar,
String headerMessage) {
        Alert alert = new Alert(AlertType.ERROR);
        alert.setTitle(titleBar);
        alert.setHeaderText(headerMessage);
        alert.setContentText(infoMessage);
        alert.showAndWait();
    }

    public void onAutoButtonClicked(ActionEvent event) { //Если нажата

```

кнопка Автоматически/Пауза

```
        if (timeline == null) {
            timeline = new Timeline(new KeyFrame(Duration.seconds(1),
ev -> {
            onNextStepButtonClicked(event);
            if (isFinished == -2) {
                timeline.stop();
                timeline = null;
                changeButton(true);
                return;
            }
        });
        timeline.setCycleCount(Animation.INDEFINITE);
        timeline.play();
        changeButton(false);
    } else {
        timeline.stop();
        timeline = null;
        changeButton(true);
    }
}

public void changeButton(boolean value) //true - установить
"Автоматически" false - установить "Приостановить"
{
    if (value) AutoButton.setText("Автоматически");
    if (!value) AutoButton.setText("Приостановить");
}

public void onClearButtonClicked(ActionEvent event) { // Если
нажата клавиша "Сброс"
    if (!MainHBox.getChildren().isEmpty()) {
        MainHBox.getChildren().remove(0, vectorCanvas.size());
    }
    GraphicsContext gc = canvasArea.getGraphicsContext2D();
    gc.clearRect(0, 0, 1880, 10000);
    Text_Field.clear();
    TextFieldTill.clear();
    TextFieldNum.clear();
    TextFieldNumStr.clear();
    TextFieldTillStr.clear();
    mainController.clearSaves();
    timeline = null;
    updateCurrentDigit(-10);
}

public void onFinishSortingButtonClicked(ActionEvent event) { //
Если нажата клавиша "Завершить сортировку"
    mainController.onFinishSortingButtonClicked();
}

public void setCanvases(int numCanvases) { // Устанавливает
количество столбцов с холстами для вывода по разрядам (37 столбцов для
строк, 10 столбцов для чисел)

    if (vectorCanvas != null) {
        if (!MainHBox.getChildren().isEmpty()) {
            MainHBox.getChildren().remove(0, vectorCanvas.size());
        }
    }
    vectorCanvas = new Vector<Canvas>();
    vectorScrollPane = new Vector<ScrollPane>();
    vectorVBox = new Vector<VBox>();
    Character sym = 'A';
```

```

for (int i = 0; i < numCanvases; ++i) {
    String name = new String();
    VBox vbox = new VBox(); //Создали вертикальный слой
    Label label = new Label();
    if (i < 10 && numCanvases < 11) {
        Integer sign = i;
        name = sign.toString();
    } else {
        if (i < 11 && numCanvases > 10) {
            if (i == 0) {
                name = "$";
            } else {
                Integer sign = i - 1;
                name = sign.toString();
            }
        } else {
            name = sym.toString();
            char ch = (char) sym;
            int n = (int) ch;
            ++n;
            ch = (char) n;
            sym = (Character) ch;
        }
    }
    label.setText(name); //Создали название колонки
    label.setFont(new Font(24));

    //label.setPrefWidth(500);
    //185////////////////////////////////////
    String stroka = TextField.getText();
    //String noPrefixStr =
    stroka.substring(stroka.indexOf(prefix) + prefix.length());

    String[] tokens = stroka.split(" ");

    int maxlen = 0;
    for (String t : tokens)
        if (t.length() > maxlen) {
            maxlen = t.length();
        }
    label.setPrefWidth(50 + maxlen*20);
    CanvasWidth = 30 + maxlen*20;

    label.setAlignment(Pos.CENTER);
    ScrollPane scrollPane = new ScrollPane(); //Создали панель
    скроллинга
    scrollPane.setStyle("-fx-border-color: white");
    scrollPane.setPrefHeight(ScrollPaneLow.getHeight() -
50); ////////////////////////////////////
    scrollPane.setMinHeight(200);
    scrollPane.setPrefWidth(50 + maxlen*20); //
    Canvas canvas = new Canvas(CanvasWidth,
canvasLength); //Создали холст
    scrollPane.setContent(canvas);
    vbox.getChildren().add(scrollPane);
    vbox.getChildren().add(label);
    MainHBox.getChildren().add(vbox);
    vectorCanvas.add(canvas);
    vectorScrollPane.add(scrollPane);
    vectorVBox.add(vbox);
}
}

```

```

    public void onFillingButtonClicked(ActionEvent event) { // Если
нажата клавиша "Заполнить"
        Text_Field.clear();
        boolean was = false;
        final Random random = new Random();
        String strNum = TextFieldNum.getText();
        String strTill = TextFieldTill.getText();
        String strTillOt = TextFieldTillOt.getText();

        if (strNum.length() != 0) {
            int count = Integer.parseInt(strNum);
            int edge = Integer.parseInt(strTill);
            int edgeOt = Integer.parseInt(strTillOt);
            for (int i = 0; i < count; ++i) {
                Integer num = edgeOt + random.nextInt(edge - edgeOt +
1);

                if (i != count - 1) {
                    Text_Field.appendText(num.toString() + " ");
                } else {
                    Text_Field.appendText(num.toString());
                }
            }
            was = true;
        }
        String strNumStr = TextFieldNumStr.getText();
        String strTillStr = TextFieldTillStr.getText();
        if (strNumStr.length() != 0) {
            if (was) Text_Field.appendText(" ");
            int count = Integer.parseInt(strNumStr);
            int edge = Integer.parseInt(strTillStr);

            for (int i = 0; i < count; ++i) {
                String result = new String();
                int numRepeat = random.nextInt(edge) + 1;
                for (int j = 0; j < numRepeat; j++) {
                    String tempresult = new String();
                    int choiceNumOrLettter = random.nextInt(6);
                    if (choiceNumOrLettter == 1) {
                        Integer num = random.nextInt(10);
                        tempresult = num.toString();
                    } else {
                        int num;
                        num = random.nextInt(26);
                        char ch = 'a';
                        int m = (int) ch;
                        m = m + num;
                        ch = (char) m;
                        tempresult = String.valueOf(ch);
                    }
                    result = result + tempresult;
                }

                if (i != count - 1) {
                    Text_Field.appendText(result + " ");
                } else {
                    Text_Field.appendText(result);
                }
            }
        }
    }

    public void updateCurrentDigit(int digit) { //Выводит строку с

```



```

текущим разрядом сортировки
    if (digit == -10) {
        DigitLabel.setText("");
        return;
    }
    if (digit != 0) {
        Integer dig = new Integer(digit);
        DigitLabel.setText(" Текущий разряд сортировки: " +
dig.toString());
    } else {
        DigitLabel.setText(" Сортировка выполнена");
    }
}

public void onExitButtonClicked(ActionEvent event) {
    System.exit(0);
}

public void updateWorkingArrayInteger(Vector<Integer> array) { //
Выводит на экран содержание всего массива чисел, подлежащих сортировке
    Vector<String> strings = new Vector<String>();
    for (int i = 0; i < array.size(); i++) {
        Integer num = array.get(i);
        strings.add(num.toString());
    }
    updateWorkingArrayString(strings);
}

public void updateWorkingArrayString(Vector<String> array) { //
Выводит на экран содержание всего массива строк, подлежащих сортировке
    GraphicsContext gc = canvasArea.getGraphicsContext2D();
    gc.clearRect(0, 0, 1880, 3000);
    gc.setFont(new Font("Consolas", fontSize)); //Courier New
    gc.setFill(Color.GREEN);
    gc.setTextAlign(TextAlignment.LEFT);
    int x = 5;
    int y = 30;
    for (int i = 0; i < array.size(); i++) {
        String str = array.get(i).replaceAll("$", "");
        if (i == 0) {
            gc.setFill(Color.ORANGE);
        } else {
            gc.setFill(Color.GREEN);
        }
        for (int j = 0; j <= str.length() - 1; j++) {
            char ch = str.charAt(j);
            String s = Character.toString(ch);
            gc.fillText(s, x, y);
            x = x + 18;
        }
        x = x + 25;
        if (i != array.size() - 1) {
            if (x + array.get(i + 1).length() * 18 > 1800) {
                y = y + fontSize + spaceFont;
                x = 5;
            }
        }
    }
}

/**
 * @param array
 * @param column

```

```

        */
        public void updateComponentsArrayInteger(Vector<Integer> array, int
column, int digit) { //Выводит в столбец column числа и подсвечивает
разряд digit
            Vector<String> vectorStr = new Vector<String>();
            for (int i = 0; i < array.size(); i++) {
                Integer num = array.get(i);
                vectorStr.add(num.toString());
            }
            updateComponentsArrayString(vectorStr, column, digit);
        }

        public void updateComponentsArrayString(Vector<String> array, int
column, int digit) { //Выводит в столбец column строки и подсвечивает
разряд digit
            GraphicsContext gc =
vectorCanvas.get(column).getGraphicsContext2D();
            gc.clearRect(0, 0, CanvasWidth, canvasLength);
            gc.setFont(new Font("Consolas", fontSize)); //Courier New
            gc.setFill(Color.GREEN);
            gc.setTextAlign(TextAlignment.RIGHT);
            double scrollMiss = 1.0;
            int y = canvasLength - spaceFont - 5;
            int x = CanvasWidth - 5;
            ;
            for (int i = 0; i < array.size(); i++) {
                String str = array.get(i);
                str = str.replaceAll("[\$]", "_");

                x = CanvasWidth - 5;
                for (int j = str.length() - 1; j >= 0; j--) {
                    char ch = str.charAt(j);
                    String s = Character.toString(ch);
                    if (j == str.length() - digit) {
                        gc.setStroke(Color.RED);
                        gc.setFill(Color.RED);
                        gc.fillText(s, x, y);
                        x = x - 18; //Промежуток между символами
                        gc.setStroke(Color.GREEN);
                        gc.setFill(Color.GREEN);
                    } else {
                        gc.fillText(s, x, y);
                        x = x - 18;
                    }
                }
                y = y - fontSize - spaceFont;
                if (i % 10 == 0 && i > 9) {
                    scrollMiss = scrollMiss - 0.15;
                }
                vectorScrollPane.get(column).setVvalue(scrollMiss);
            }
        }

        public void lightColumn(int column) { //Подсветка столбца column
            if (column < 0) {

                if (prevMark != null) {
                    prevMark.setStyle("-fx-border-color: white");
                }
                prevMark = null;
                return;
            } else {

```

```

        if (mainController.mode == MWController.Mode.STRINGS &&
column > 36)
            return;

        if (mainController.mode == MWController.Mode.NUMBERS &&
column > 9)
            return;

        if (prevMark != null) {
            prevMark.setStyle("-fx-border-color: white");
        }
        vectorScrollPane.get(column).setStyle("-fx-border-color:
red");
        prevMark = vectorScrollPane.get(column);
    }
    scrollPaneLow.setHvalue(0);
    if (column > 8 && column < 19) scrollPaneLow.setHvalue(0.373);
    if (column > 18 && column < 29) scrollPaneLow.setHvalue(0.746);
    if (column > 28) scrollPaneLow.setHvalue(1.0);
}
}

```

3. MWController.java

```

package com.eltech.gui;

import com.eltech.radix.IntRadixSorter;
import com.eltech.radix.StringRadixSorter;

import java.util.Vector;

public class MWController {

    Vector<Integer> InitialArrayInt;
    Vector<String> InitialArrayString;

    Vector<IntRadixSorter> intRadixSortHistory;
    Vector<StringRadixSorter> stringRadixSortHistory;

    Controller control;

    public MWController() {
    }

    /**
     * Список режимов сортировки
     */
    public enum Mode {
        STRINGS, NUMBERS
    }

    ;

    public Mode mode;

    /**
     * Устанавливает режим сортировки - числа или строки
     */
}

```

```

    * @param m - название режима
    */
    public void setMode(Mode m) {
        System.out.println("Setting mode to " + m.toString());
        control.printToTextArea("Setting mode to " + m.toString());
        clearSaves();
        mode = m;
    }

    /**
     * Устанавливает контроллер для данного класса
     */
    public void setControl(Controller c) {
        this.control = c;
    }

    /**
     * Запоминает промежуточные состояния сортировки
     */
    * @param sorter - сортировщик целых чисел
    */
    public void memorize(IntRadixSorter sorter) {
        IntRadixSorter irs = new IntRadixSorter(sorter);
        intRadixSortHistory.add(irs);
    }

    /**
     * Запоминает промежуточные состояния сортировки
     */
    * @param sorter - сортировщик строк
    */
    public void memorize(StringRadixSorter sorter) {
        StringRadixSorter srs = new StringRadixSorter(sorter);
        stringRadixSortHistory.add(srs);
    }

    /**
     * Очищает все сохраненные состояния сортировщиков
     */
    public void clearSaves() {
        System.out.println("Clearing saves!");
        control.printToTextArea("Clearing saves!");

        intRadixSortHistory = new Vector<>();
        stringRadixSortHistory = new Vector<>();
    }

    /**
     * Отдает последний использовавшийся сортировщик целых чисел
     */
    * @return
    */
    public IntRadixSorter getIntRadixSorter() {
        if (intRadixSortHistory.isEmpty()) {
            IntRadixSorter sorter = new IntRadixSorter();
            sorter.setController(control);
            intRadixSortHistory.add(sorter);
        }

        return intRadixSortHistory.lastElement();
    }
}

```

```

/**
 * Отдает последний использовавшийся сортировщик строк
 *
 * @return
 */
public StringRadixSorter getStringRadixSorter() {
    if (stringRadixSortHistory.isEmpty()) {
        StringRadixSorter sorter = new StringRadixSorter();
        sorter.setController(control);
        stringRadixSortHistory.add(sorter);
    }

    return stringRadixSortHistory.lastElement();
}

/**
 * Отматывает состояние сортировщика назад на одну операцию
 *
 * @return
 */
public boolean rewind() {
    if (mode == Mode.NUMBERS) {

        if (intRadixSortHistory.size() == 1) {
            return false;
        } else {

intRadixSortHistory.removeElementAt(intRadixSortHistory.size() - 1);
            return true;
        }

    } else if (mode == Mode.STRINGS) {

        if (stringRadixSortHistory.size() == 1) {
            return false;
        } else {

stringRadixSortHistory.removeElementAt(stringRadixSortHistory.size() -
1);
            return true;
        }

    }

    return false;
}

public int onNextStepButtonClicked() {
    int col = -1;
    if (mode == Mode.NUMBERS) {

        IntRadixSorter irs = getIntRadixSorter();

        // Заносим состояние сортировщика в историю
        memorize(irs);

        irs = getIntRadixSorter();
        col = irs.doStep();

        // Теперь обновляем
        control.updateWorkingArrayInteger(irs.getWorkingArray());
        for (int i = 0; i < 10; ++i) {

control.updateComponentsArrayInteger(irs.getCategoryArray(i), i,

```

```

    irs.getCurrentDigit());
    }
    control.updateCurrentDigit(irs.getCurrentDigit());
    control.lightColumn(col);

    } else if (mode == Mode.STRINGS) {

        StringRadixSorter srs = getStringRadixSorter();

        // Заносим состояние сортировщика в историю
        memorize(srs);
        srs = getStringRadixSorter();

        col = srs.doStep();
        // Теперь обновляем
        control.updateWorkingArrayString(srs.getWorkingArray());
        for (int i = 0; i < 37; ++i) {

            control.updateComponentsArrayString(srs.getCategoryArray(i), i,
            srs.getCurrentDigit());
            }
            control.updateCurrentDigit(srs.getCurrentDigit());
            control.lightColumn(col);
        }
        return col;
    }

    public void onPreviousStepButtonClicked() // Если нажата клавиша
    предыдущий шаг
    {
        if (mode == Mode.NUMBERS) {

            rewind();
            IntRadixSorter irs = getIntRadixSorter();

            // Теперь обновляем
            control.updateWorkingArrayInteger(irs.getWorkingArray());
            for (int i = 0; i < 10; ++i) {

                control.updateComponentsArrayInteger(irs.getCategoryArray(i), i,
                irs.getCurrentDigit());
                }
                control.updateCurrentDigit(irs.getCurrentDigit());

            } else if (mode == Mode.STRINGS) {

                rewind();
                StringRadixSorter srs = getStringRadixSorter();

                // Теперь обновляем
                control.updateWorkingArrayString(srs.getWorkingArray());
                for (int i = 0; i < 37; ++i) {

                    control.updateComponentsArrayString(srs.getCategoryArray(i), i,
                    srs.getCurrentDigit());
                    }
                    control.updateCurrentDigit(srs.getCurrentDigit());
                }

                control.lightColumn(-1);

            }

        }

    public int onEnterDataClicked(String str) { // Если нажата клавиша

```

```

"Ввести данные"
boolean choice = true; // true - это строка чисел, false -
строка

System.out.println("OnEnterData:");
control.printToTextArea("OnEnterData:");

String strArray[] = str.split(" ");
for (int i = 0; i < strArray.length; i++) {
    if (strArray[i].length() > 8) return 1;
    if (!isDigit(strArray[i])) //Вернет true, если строка может
быть преобразована в число
    {
        choice = false;
    }
}

// Сортируем числа
if (choice) {
    System.out.println("    sorting numbers");
    control.printToTextArea("    sorting numbers");

    setMode(Mode.NUMBERS);

    InitialArrayInt = new Vector<Integer>();
    for (int i = 0; i < strArray.length; ++i) {
        InitialArrayInt.add(Integer.parseInt(strArray[i]));
    }
    control.setCanvases(10);
    sortNumbers(InitialArrayInt);
}

// Сортируем строки
else {
    System.out.println("    sorting strings");
    control.printToTextArea("    sorting strings");

    setMode(Mode.STRINGS);
    int maxlength = 0;
    for (int i = 0; i < strArray.length; i++) {
        if (strArray[i].length() > maxlength) maxlength =
strArray[i].length();
    }
    InitialArrayString = new Vector<String>();
    for (int i = 0; i < strArray.length; ++i) {
        int countLength = strArray[i].length();
        while (maxlength != countLength) {
            strArray[i] = strArray[i] + '$';
            ++countLength;
        }
        InitialArrayString.add(strArray[i].toLowerCase());
    }
    for (int i = 0; i < InitialArrayString.size(); i++) {
        System.out.println(InitialArrayString.get(i));
        control.printToTextArea(InitialArrayString.get(i));
    }

    control.setCanvases(37);
    sortStrings(InitialArrayString);
}
return 0;
}

```

```

        private static boolean isDigit(String s) throws
        NumberFormatException {
            try {
                Integer.parseInt(s);
                return true;
            } catch (NumberFormatException e) {
                return false;
            }
        }

        public void sortNumbers(Vector<Integer> arr) {
            IntRadixSorter irs = getIntRadixSorter();
            irs.load(arr);

            memorize(irs);

            control.updateCurrentDigit(1);
            control.updateWorkingArrayInteger(arr);

            System.out.println("sorting numbers " + arr.toString());
            control.printToTextArea("sorting numbers " + arr.toString());
        }

        public void sortStrings(Vector<String> arr) {
            StringRadixSorter srs = getStringRadixSorter();
            srs.load(arr);

            memorize(srs);

            control.updateCurrentDigit(1);
            control.updateWorkingArrayString(arr);

            System.out.println("sorting strings " + arr.toString());
            control.printToTextArea("sorting strings " + arr.toString());
        }

        public void onFinishSortingButtonClicked() { // Если нажата клавиша
        "Завершить сортировку"
            if (mode == Mode.NUMBERS) {

                IntRadixSorter irs = getIntRadixSorter();
                while (irs.doStep() != -2) {

                    // Заносим состояние сортировщика в историю
                    memorize(irs);
                }
                control.lightColumn(-1);

                // Теперь обновляем
                control.updateWorkingArrayInteger(irs.getWorkingArray());
                for (int i = 0; i < 10; ++i) {

                    control.updateComponentsArrayInteger(irs.getCategoryArray(i), i,
                    irs.getCurrentDigit());
                }
                control.updateCurrentDigit(0);
            } else if (mode == Mode.STRINGS) {
                StringRadixSorter srs = getStringRadixSorter();
                while (srs.doStep() != -2) {

                    // Заносим состояние сортировщика в историю
                    memorize(srs);
                }
            }
        }
    }

```



```

    }

    // Теперь обновляем
    control.updateWorkingArrayString(srs.getWorkingArray());
    for (int i = 0; i < 37; ++i) {

        control.updateComponentsArrayString(srs.getCategoryArray(i), i,
            srs.getCurrentDigit());
    }
    control.updateCurrentDigit(0);
}
}
}

```

4. IntRadixSorter.java

```

package com.eltech.radix;

import java.util.Vector;

public class IntRadixSorter extends RadixSorter {

    private Vector<Integer> m_workingArray;
    private Vector<Vector<Integer>> m_categoriesArray;

    public IntRadixSorter() {
        super();
    }

    /**
     * Конструктор копии
     *
     * @param other
     */
    public IntRadixSorter(IntRadixSorter other) {
        m_maxDigits = other.m_maxDigits;
        m_currentDigit = other.m_currentDigit;
        controller = other.controller;

        m_workingArray = new Vector<>(other.m_workingArray);
        m_categoriesArray.clear();
        for (int i = 0; i < 10; ++i) {
            m_categoriesArray.add(
                new Vector<>(
                    other.m_categoriesArray.elementAt(i)));
        }
    }

    /**
     * Загружает массив для сортировки внутрь класса
     *
     * @param array - массив, который нужно отсортировать
     */
    public void load(Vector<Integer> array) {
        clear();
        m_workingArray = new Vector<>(array);
        m_maxDigits = maxDigits();
    }

    /**
     * Возвращает рабочий массив,

```

```

        *
        * @return
        */
    public Vector<Integer> getWorkingArray() {
        return m_workingArray;
    }

    /**
     * Возвращает массив заданной колонки
     *
     * @param cat - номер колонки для возврата
     * @return
     */
    public Vector<Integer> getCategoryArray(int cat) {
        return m_categoriesArray.elementAt(cat);
    }

    /**
     * Делает один шаг сортировки
     *
     * @return По окончании сортировки возвращает false
     */
    public int doStep() {
        if (m_currentDigit == 0) {
            System.out.println("Starting the sort!");
            controller.printToTextArea("Starting the sort!");
            m_currentDigit = 1;
        } else if (m_currentDigit > m_maxDigits) {

            System.out.println("Sorting finished!");
            controller.printToTextArea("Sorting finished!");

            m_currentDigit = 0;

            return -2;
        }

        // Если рабочий массив не пустой - переносим первый его элемент
        // в нужную колонку
        if (!m_workingArray.isEmpty()) {
            Integer nextNumber = new Integer(m_workingArray.remove(0));
            System.out.println("Step: taking " + nextNumber);
            controller.printToTextArea("Step: taking " + nextNumber);

            int moveTo;
            if (m_currentDigit > hasDigits(nextNumber)) {
                moveTo = 0;
            } else {
                moveTo = digitAt(nextNumber, m_currentDigit);
            }

            System.out.println("    moving to " + moveTo + " group");
            controller.printToTextArea("    moving to " + moveTo + "
group");

            m_categoriesArray.elementAt(moveTo).add(nextNumber);
            return moveTo;

            // Если пустой, склеиваем все колонки в новый рабочий
            массив
        } else {
            System.out.println("Working array is empty: gluing all the
shit together!");
            controller.printToTextArea("Working array is empty: gluing

```

```

all the shit together!");

        for (int i = 0; i < 10; ++i) {

            for (int j = 0; j <
m_categoriesArray.elementAt(i).size(); ++j) {

m_workingArray.addElement(m_categoriesArray.elementAt(i).elementAt(j));
            }

            m_categoriesArray.elementAt(i).clear();

        }

        m_currentDigit++;
        return -1;
    }

}

/**
 * Очищает рабочее пространство класса
 */
public void clear() {
    super.clear();

    m_workingArray = new Vector<>();
    m_categoriesArray = new Vector<>();

    for (int i = 0; i < 10; ++i) {
        m_categoriesArray.add(new Vector<>());
    }
}

/**
 * Возвращает количество цифр в числе
 *
 * @param number - число, в котором нужно посчитать количество цифр
 * @return
 */
private int hasDigits(Integer number) {
    int digits = 1;
    Integer numberCpy = new Integer(number);

    while (numberCpy >= 10) {
        numberCpy /= 10;
        digits++;
    }

    return digits;
}

/**
 * Возвращает цифру на заданной позиции
 *
 * @param number - число
 * @param pos - разряд, цифру которого нужно получить
 * @return
 */
private int digitAt(Integer number, int pos) {
    Integer numberCpy = new Integer(number);

    if (pos > hasDigits(number))

```

```

        return -1;

    if (pos != 1) {
        numberCpy /= (int) Math.pow(10, pos - 1);
    }
    return numberCpy % 10;
}

/**
 * Возвращает длину максимального числа в массиве
 *
 * @return
 */
private int maxDigits() {
    int max = 0;

    for (int i = 0; i < m_workingArray.size(); ++i) {
        if (hasDigits(m_workingArray.elementAt(i)) > max)
            max = hasDigits(m_workingArray.elementAt(i));
    }

    return max;
}
}

```

5. StringRadixSorter.java

```

package com.eltech.radix;

import java.util.Vector;

public class StringRadixSorter extends RadixSorter {
    private Vector<String> m_workingArray;
    private Vector<Vector<String>> m_categoriesArray;

    public StringRadixSorter() {
        super();
    }

    /**
     * Конструктор копии
     *
     * @param other
     */
    public StringRadixSorter(StringRadixSorter other) {
        m_maxDigits = other.m_maxDigits;
        m_currentDigit = other.m_currentDigit;
        controller = other.controller;

        m_workingArray = new Vector<>(other.m_workingArray);
        m_categoriesArray.clear();
        for (int i = 0; i < 37; ++i) {
            m_categoriesArray.add(
                new Vector<>(
                    other.m_categoriesArray.elementAt(i));
            )
        }
    }

    /**
     * Загружает массив для сортировки внутрь класса
     *

```

```

        * @param array - массив, который нужно отсортировать
        */
    public void load(Vector<String> array) {
        clear();
        m_workingArray = new Vector<>(array);
        m_maxDigits = maxDigits();
    }

    /**
     * Возвращает рабочий массив,
     *
     * @return
     */
    public Vector<String> getWorkingArray() {
        return m_workingArray;
    }

    /**
     * Возвращает массив заданной колонки
     *
     * @param cat - номер колонки для возврата
     * @return
     */
    public Vector<String> getCategoryArray(int cat) {
        return m_categoriesArray.elementAt(cat);
    }

    /**
     * Делает один шаг сортировки
     *
     * @return По окончании сортировки возвращает false
     */
    public int doStep() {
        if (m_currentDigit == 0) {
            System.out.println("Starting the sort!");
            controller.printToTextArea("Starting the sort!");
            m_currentDigit = 1;
        } else if (m_currentDigit > m_maxDigits) {

            System.out.println("Sorting finished!");
            controller.printToTextArea("Sorting finished!");
            m_currentDigit = 0;

            return -2;
        }

        // Если рабочий массив не пустой - переносим первый его элемент
        в нужную колонку
        if (!m_workingArray.isEmpty()) {
            String nextString = new String(m_workingArray.remove(0));
            System.out.println("Step: taking " + nextString);
            controller.printToTextArea("Step: taking " + nextString);

            int moveTo;
            if (m_currentDigit > nextString.length()) {
                moveTo = 0;
                System.out.println("    bug ");
                controller.printToTextArea("    bug ");
            } else {
                moveTo =
                matchingColumn(nextString.charAt(nextString.length() -
                (m_currentDigit)));
            }

```

```

        System.out.println("      moving to " + moveTo + " group");
        controller.printToTextArea("      moving to " + moveTo + "
group");

        m_categoriesArray.elementAt(moveTo).add(nextString);
        return moveTo;

        // Если пустой, склеиваем все колонки в новый рабочий
массив
    } else {
        System.out.println("Working array is empty: gluing all the
shit together!");
        controller.printToTextArea("Working array is empty: gluing
all the shit together!");

        for (int i = 0; i < m_categoriesArray.size(); ++i) {

            for (int j = 0; j <
m_categoriesArray.elementAt(i).size(); ++j) {
m_workingArray.addElement(m_categoriesArray.elementAt(i).elementAt(j));
            }

            m_categoriesArray.elementAt(i).clear();

        }

        m_currentDigit++;
        return -1;
    }

}

/**
 * Очищает рабочее пространство класса
 */
public void clear() {
    super.clear();

    m_workingArray = new Vector<>();
    m_categoriesArray = new Vector<>();

    for (int i = 0; i < 37; ++i) {
        m_categoriesArray.add(new Vector<>());
    }

}

/**
 * Возвращает длину максимального числа в массиве
 *
 * @return
 */
private int maxDigits() {
    int max = 0;

    for (int i = 0; i < m_workingArray.size(); ++i) {
        if (m_workingArray.elementAt(i).length() > max)
            max = m_workingArray.elementAt(i).length();
    }

    return max;
}

```

```

/**
 * Возвращает номер колонки для определенного символа
 *
 * @param c - СИМВОЛ
 * @return
 */
private static int matchingColumn(char c) {
    switch (c) {
        case '$':
            return 0;
        case '0':
            return 1;
        case '1':
            return 2;
        case '2':
            return 3;
        case '3':
            return 4;
        case '4':
            return 5;
        case '5':
            return 6;
        case '6':
            return 7;
        case '7':
            return 8;
        case '8':
            return 9;
        case '9':
            return 10;

    }

    return c - 'a' + 11;
}

}

```

6. RadixSorter.java

```

package com.eltech.radix;

import com.eltech.gui.Controller;

public abstract class RadixSorter {
    public Controller getController() {
        return controller;
    }

    public void setController(Controller controller) {
        this.controller = controller;
    }

    Controller controller;
    protected int m_currentDigit;
    protected int m_maxDigits;

    /**
     * Конструктор класса
     */
    public RadixSorter() {
        clear();
    }
}

```

```

    }

    /**
     * Возвращает текущий разряд сортировки
     *
     * @return
     */
    public int getCurrentDigit() {
        return m_currentDigit;
    }

    /**
     * Делает один шаг сортировки
     *
     * @return Возвращает номер колонки, в которую был добавлен элемент
или -1,
     * если колонки склеиваются в рабочий массив, либо -2, если
сортировка закончился
     */
    public abstract int doStep();

    /**
     * Выполняет сортировку до конца
     */
    public void sort() {
        while (doStep() != -2) {
        }
    }

    /**
     * Очищает рабочее пространство класса
     */
    public void clear() {
        m_currentDigit = 0;
        m_maxDigits = 0;
    }
}

```