

**Министерство образования и науки Российской Федерации**

**ФГАОУ ВО «Уральский федеральный университет**

**имени первого Президента России Б.Н.Ельцина»**

**Центр ускоренного обучения**

Оценка проекта

Члены комиссии

**РАЗРАБОТКА ПРОГРАММЫ ДЛЯ РЕШЕНИЯ ЗАДАЧИ  
«ПРОВЕРКА ВЕБ-СТРАНИЦ»**

**КУРСОВАЯ РАБОТА**

по дисциплине «Информатика и программирование»

Пояснительная записка

09.03.04 58.29.29 005 ПЗ

Руководитель доц., к.т.н.

С.И. Тимошенко

Студент гр. РИВ-350023у

И.Д. Василевский

Екатеринбург 2018

# СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	3
1 Постановка задачи.....	4
2 Анализ поставленной задачи.....	5
2.1 Основные объекты задачи и их взаимодействие.....	5
2.2 Структуры данных.....	5
2.3 Алгоритм.....	5
3 Описание результатов разработки.....	8
3.1 Разработка тестового класса.....	8
3.2 Разработка основного класса программы.....	8
3.3 Окончательный вариант программы.....	10
3.3.1 Общая схема программы.....	10
4 Руководство пользователя.....	11
4.1 Минимальные системные требования.....	11
4.2 Установка и удаление программы.....	11
4.3 Работа с программой.....	11
ЗАКЛЮЧЕНИЕ.....	13
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	14
ПРИЛОЖЕНИЕ А (обязательное) Исходный код программы.....	15
A.1 Класс App.....	15
A.2 Класс Checker.....	16
A.3 Класс DocumentException.....	17
A.4 Класс Document.....	17
A.5 Класс DocumentReaderImpl.....	18
A.6 Интерфейс DocumentReader.....	19
A.7 Класс CheckerTest.....	20
A.8 Класс DocumentReaderImplTest.....	20

# ВВЕДЕНИЕ

Поставленной целью курсовой работы является разработка программы, решающая задачу «Проверке веб-страниц». Для разработки программы необходимо использовать язык программирования Java. При разработке программы, будем руководствоваться описанием задачи в книге Меншикова Ф.В. «Олимпиадные задачи по программированию». Задача о проверке веб-страниц заключается в следующем. Требуется реализовать упрощенную проверку страницы на корректность ссылок. Входной файл содержит название одного или нескольких документов и их содержимое.

Пояснительная записка имеет следующую структуру [5]:

- а) во введении описывается краткая постановка задачи и содержание разделов;
- б) основная часть состоит из нескольких разделов:
  - 1) постановка задачи – описание требований к написанию курсовой работы;
  - 2) анализ поставленной задачи – рассмотрение методов решения, предложения по программной реализации (структур данных, алгоритмов);
  - 3) описание результатов разработки – рассказывается о порядке написания программы, в том числе тестирующих модулей.
- в) в заключении проводится анализ проделанной работы;
- г) список использованных источников – содержит перечень использованной при разработке литературы;
- д) в приложениях содержатся результаты тестирования, динамического программирования, а так же руководство пользователя.

# 1 Постановка задачи

Многим из тех, кому приходилось работать в Интернете, случалось сталкиваться с неправильными ссылками, то есть ссылками на несуществующие документы. Ваша задача — реализовать упрощенную проверку страницы на корректность ссылок. Входной файл содержит название одного или нескольких документов и их содержимое. Содержимое документов имеет следующий вид:

<HTML> Текст <END>

В тесте могут присутствовать ссылки на другие документы на данном сервере. Они имеют следующий вид:

<A HREF=«имя файла»>.

Требуется найти число способов сформировать разные поезда заданной длины из имеющихся видов вагонов. Дополнительным требованием является то, что тип сцеплений на каждом конце состава должен соответствовать типу сцепления локомотива.

Ввод/вывод параметров программы осуществлять двумя вариантами – консоль и файл. Программа должна быть написана на языке программирования Java, в среде разработки IntelliJ IDEA. При разработке обязательно активное использование методов TDD и тестовых средств JUnit. Описание результатов разработки оформить в виде пояснительной записки, в соответствии с ГОСТ 2.105-90, 7.32-2001 и 7.1-2003 [4]. При описании программы использовать схемы работы системы (ГОСТ 19.701-90) и диаграммы классов по правилам UML [2]. Подробное описание задачи должно совпадать с условиями, приведенными в учебном пособии [1].

## **2 Анализ поставленной задачи**

### **2.1 Основные объекты задачи и их взаимодействие**

Рассмотрим поставленную задачу.

Долинский М. С. в книге «Решение сложных и олимпиадных задач по программированию» приводит метод подходящий для решения данной задачи. Идея метода состоит в том, чтобы определить граф, взяв за его вершины имена заданных в исходном файле документов, а также имена файлов, на которые есть ссылки из заданных документов, построить матрицу смежности и преобразовать ее методом Флойда в матрицу достижимости. Тогда общее количество ссылок на несуществующие документы — это сумма всех ссылок в графе на вершины, соответствующие файлам, имена которых не заданы в исходном файле как имена документов. Количество документов, на которые нельзя попасть по ссылкам, начиная с первого, можно вычислить, посчитав количество вершин, недостижимых от первой.

### **2.2 Структуры данных**

Какие структуры данных потребуются при решении задачи? Прежде чем определиться со структурами данных, необходимо проанализировать задачу и понять, какие функции должны будут они выполнять, в ходе реализации нашего алгоритма. Стоит отметить, что основная функция структур данных, которые нам понадобятся, это динамическая обработка и хранение данных. Исходя из этого, можно сделать вывод, что идеальным вариантом является список.

### **2.3 Алгоритм**

Определив структуры данных, перейдем к проектированию алгоритма задачи (Рисунок 1). Одним из вариантов решения задачи может быть следующий:

- а) создадим класс App содержащий точку входа программы. В нем происходит обработка входящих параметров, настройка конфигурации программы;
- б) для непосредственного решения задачи напишем отдельный класс Checker, реализующий логику решения задачи (Рисунок 2);
- в) создадим интерфейс (DocumentReader) и его реализацию (DocumentReaderImpl)

для парсинга входного документа;

- г) напомним класс, представляющий модель документа (Document);
- д) напомним динамический метод `init(InputStream)`, который инициализирует матрицу смежности по содержимому входного файла;
- е) подсчитываем количество мертвых ссылок методом `countDeadLinks()`;
- ж) считаем количество недостижимых документов методом `countNotResolved()` ;

В соответствии с требованием [2] была описана схема работы системы.



*Рисунок 1 – Общая схема работы системы*

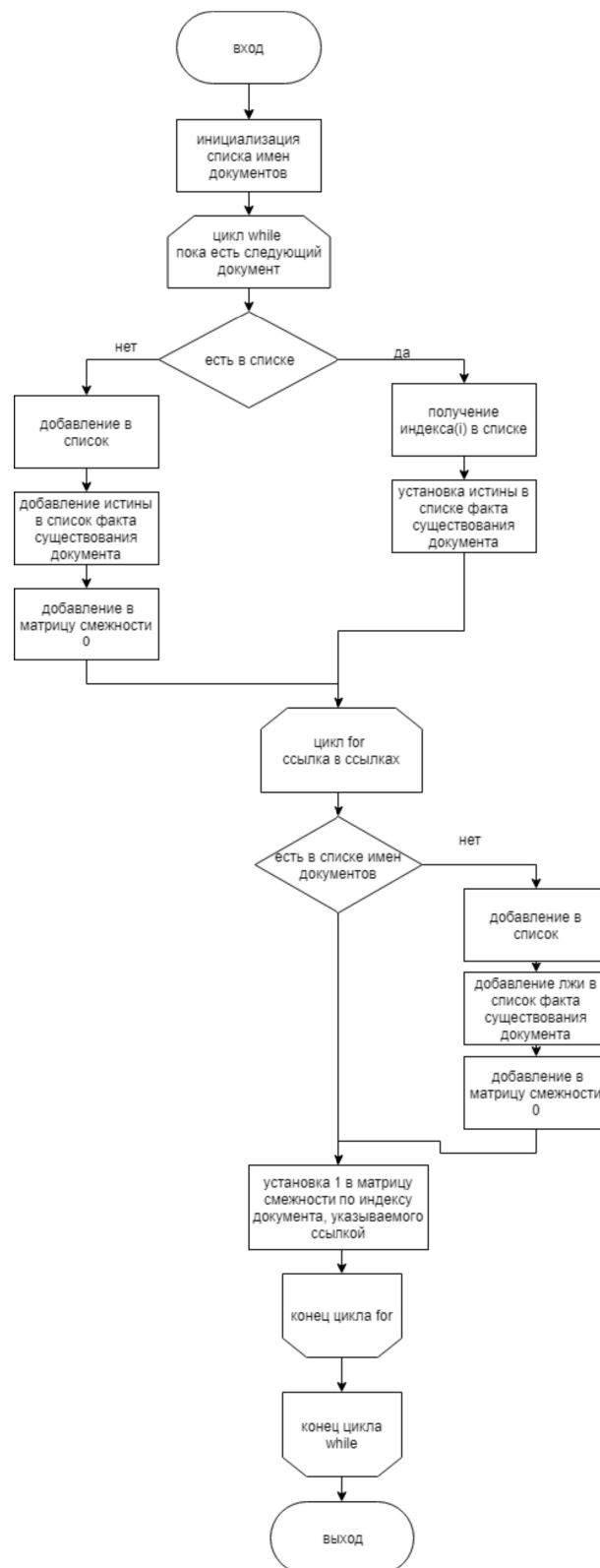


Рисунок 2 – Схема работы класса решающего задачу (Checker)

## 3 Описание результатов разработки

### 3.1 Разработка тестового класса

В соответствии с принципами TDD, разработку программы начнем с написания тестов [3]. Для тестирования пока еще не существующей программы, создадим JUnit тест, который проверяет метод, проверяющий доступность поля шахматной доски. Тест будет иметь множество уже установленных фигур, множество проверяемых полей и множество ожидаемых результатов проверки. Удачным результатом тестирования будем считать совпадение результатов проверки полей тестируемым методом с ожидаемыми результатами.

Для тестирования кода программы, воспользуемся расширением абстрактного класса JUnit. Назовем тестовый класс – CheckerTest.

Класс CheckerTest тестирующий корректность работы класса Checker содержит три метода. Метод `init()` инициализирующий окружение. Метод `testCountDeadLinks()` проверяющий соответствие количества мертвых ссылок в тестовом входном файле с вычисленным. Метод `testCountNotResolved()` проверяющий соответствие количества недостижимых документов в тестовом входном файле с вычисленным.

Диаграмма UML тестового класса (Рисунок 3):

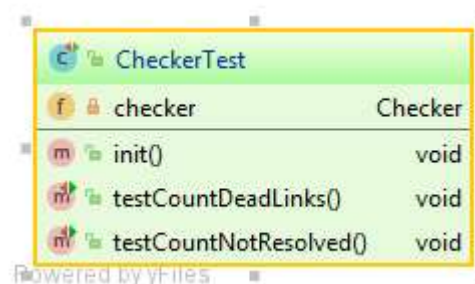
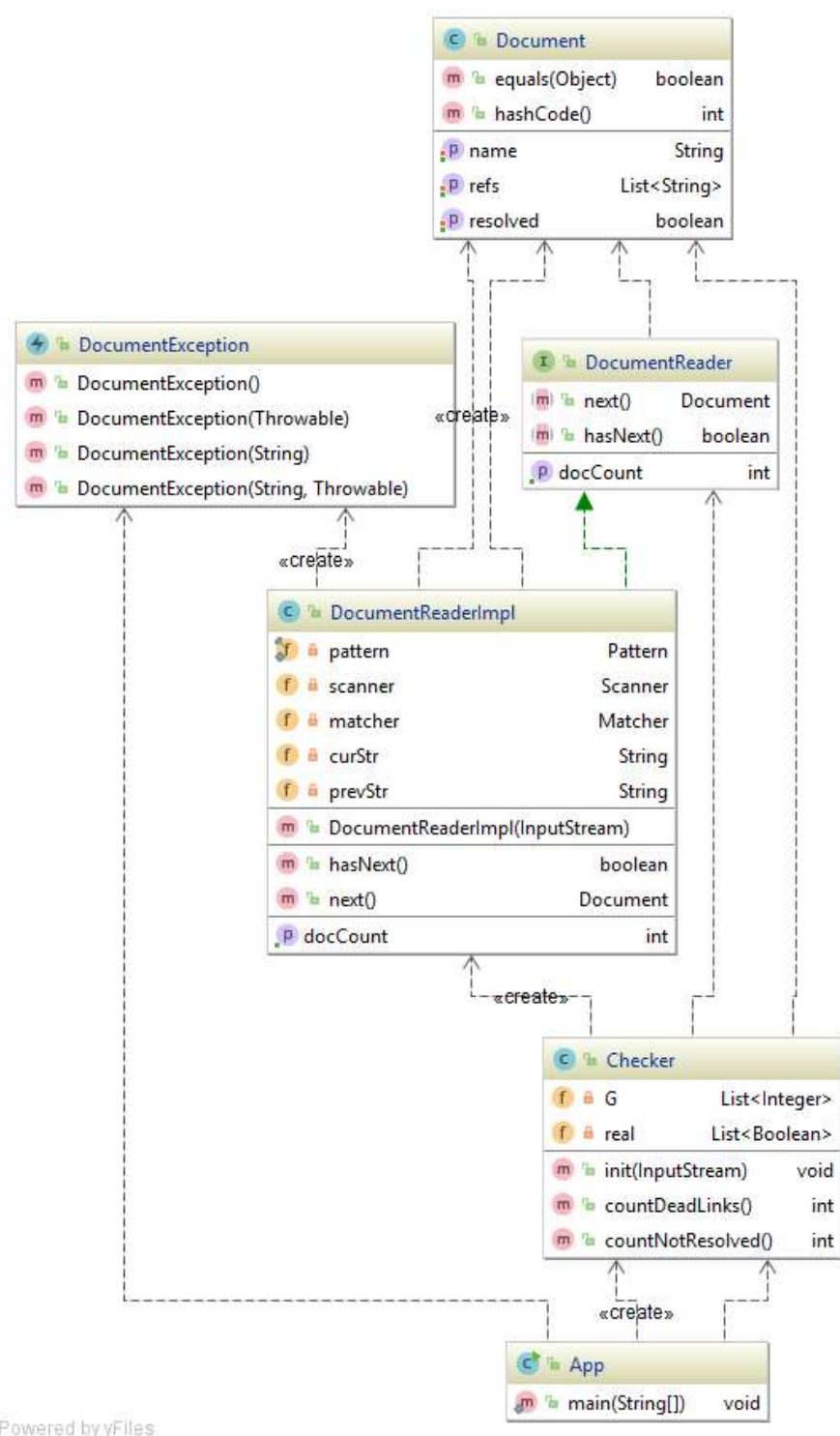


Рисунок 3 – Диаграмма UML тестирующего класса

### 3.2 Разработка основного класса программы

Тест готов, следующим этапом должна стать разработка тестируемой программы в минимальном варианте, возможно не работоспособном. Оформим диаграмму UML разрабатываемых классов (Рисунок 4).





Powered by yFiles

Рисунок 4 – Диаграмма UML классов, решающих задачу

Для того, чтобы произвести тестирование класса реализующего интерфейс парсинга документа был написан класс DocumentReaderImplTest. При запуске тестов, убеждаемся, что тесты завершены успешно.

### 3.3 Окончательный вариант программы

#### 3.3.1 Общая схема программы

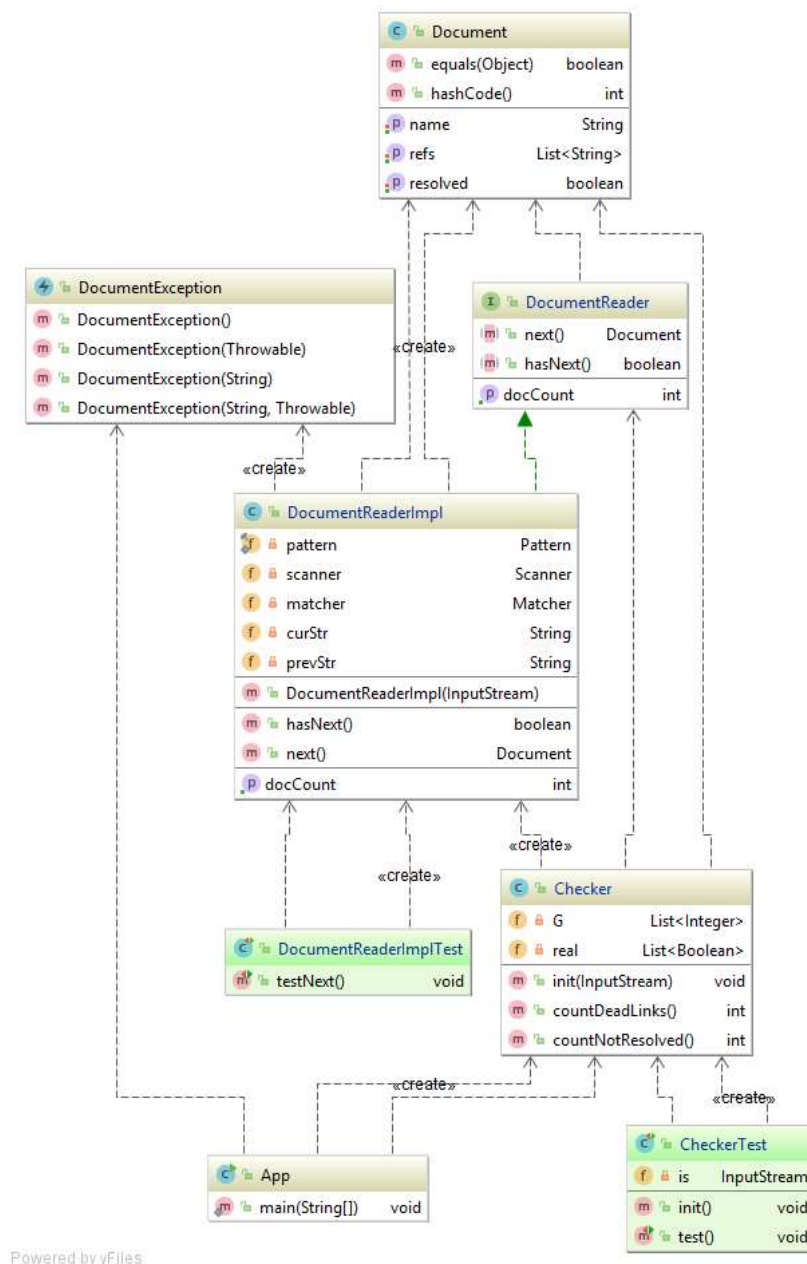


Рисунок 5 – Диаграмма UML классов

## 4 Руководство пользователя

### 4.1 Минимальные системные требования

Для установки и запуска программы «формирование поезда» необходимо выполнение следующих минимальных требований:

- операционная система MS Windows (любая версия);
- установленная виртуальная машина Java (Java 2 Platform, Standard Edition), не ниже 8 версии. Установочные файлы Java можно загрузить с сервера компании Sun, по адресу: <http://java.sun.com>.

### 4.2 Установка и удаление программы

*Для установки программы:*

- Скопируйте файлы из папки DocTest (содержащую файлы программы) в любую директорию Вашей системы (например, в C:\DocTest).
- Далее откройте командную строку. В командной строке выполните следующую команду: `java -jar %path%doctest.jar %params%`:
  - `%path%` - путь к файлу trains.jar, если вы уже находитесь в этой папке, то путь указывать не надо;
  - `%params%` - аргументы программы.

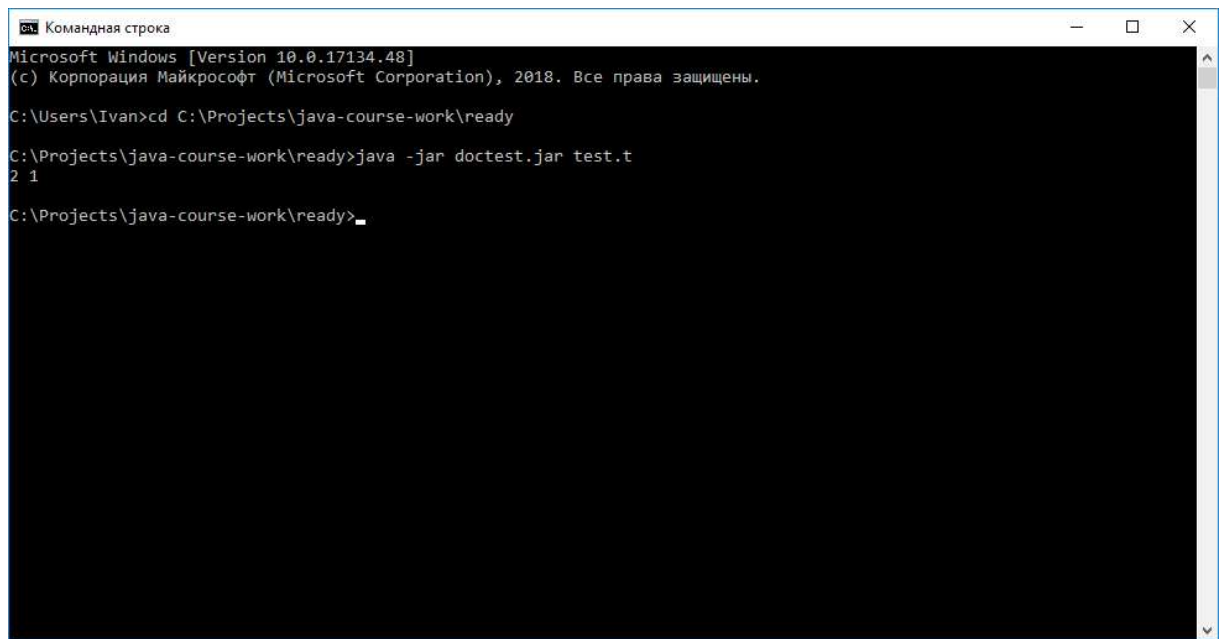
*Для удаления программы, удалите папку DocTest (из директории, в которую она была установлена).*

### 4.3 Работа с программой

Программа предусматривает в себе два режима работы: консольный и файловый.

Консольный режим работы (Рисунок 6). В консольном режиме работа программа запускается по умолчанию. Требуется первым параметром указать входной файл. Программа посчитает количество ссылок на несуществующие документы и количество недостижимых документов, выдаст ответ и завершит работу.

Файловый режим работы (Рисунок 7). Первым параметром указывается входной файл, вторым — выходной.



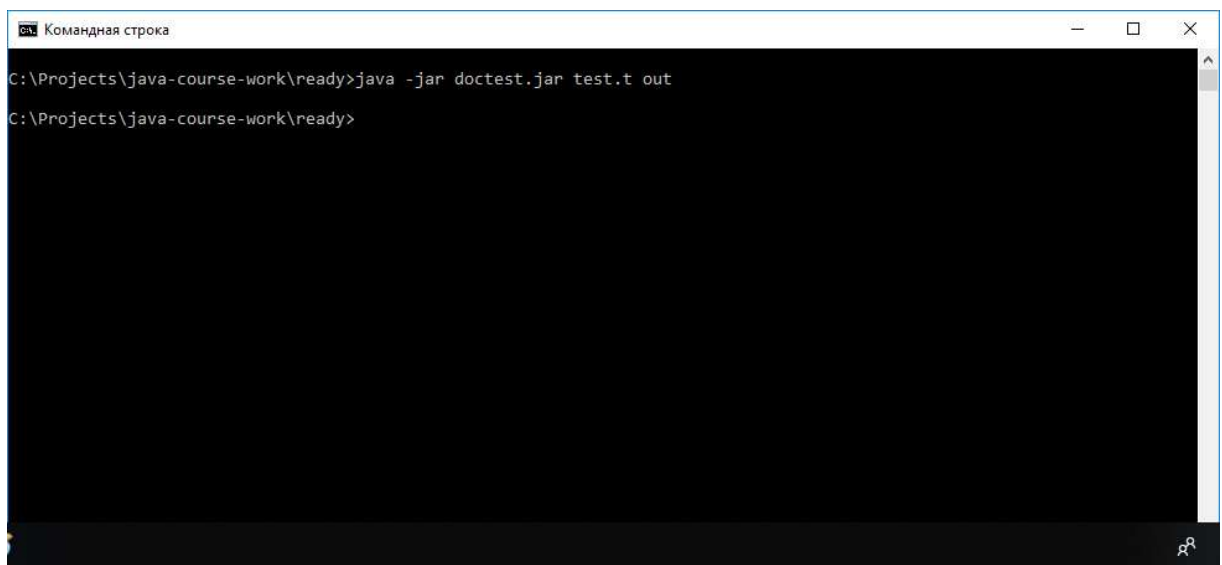
```
Командная строка
Microsoft Windows [Version 10.0.17134.48]
(c) Корпорация Майкрософт (Microsoft Corporation), 2018. Все права защищены.

C:\Users\Ivan>cd C:\Projects\java-course-work\ready

C:\Projects\java-course-work\ready>java -jar doctest.jar test.t
2 1

C:\Projects\java-course-work\ready>
```

*Рисунок 6 – Запуск программы в консольном режиме*



```
Командная строка

C:\Projects\java-course-work\ready>java -jar doctest.jar test.t out

C:\Projects\java-course-work\ready>
```

*Рисунок 7 – Запуск программы в файловом режиме*

## **ЗАКЛЮЧЕНИЕ**

В курсовой работе описан процесс разработки через тестирование программы–задачи «проверка веб-страниц». Исследована методология разработки через тестирование. Результаты всех исследованных алгоритмов приведены в приложении А. Раздел 4 включает в себя руководство пользователя, описывающее: системные требования, процессы установки, настройки и удаления программы. Окончательная версия программы – консольное Windows приложение для Java платформы. В ходе выполнения, программа использует консольный и файловый ввод/вывод.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 М. С. Долинский Решение сложных и олимпиадных задач по программированию [Текст] : учебное пособие / М. С. Долинский. – СПб. : Питер, 2006. – 366 с.
- 2 С. И. Тимошенко Порядок выполнения и требования к оформлению курсовой работы по дисциплине «Программирование на языках высокого уровня» [Текст] : методические указания / С.И.Тимошенко. – Екатеринбург: изд. ИПК УГТУ, 2004. – 16 с.
- 3 Ноутон П. Java™ 2 [Текст] : пер. с англ. / П. Ноутон, Г. Шилдт. – СПб. : БХВ-Петербург, 2005. – 1072 с.
- 4 Кичигин В.Н. Оформление курсовых и дипломных проектов [Текст] : методические указания для студентов технических специальностей / В. Н. Кичигин, И. Е.Мясников, С. И. Тимошенко. – Екатеринбург: ГОУ ВПО «УГТУ-УПИ», 2005. – 80 с.
- 5 Мясников И. Е. Государственная итоговая аттестация [Текст] : учеб.-метод. пособие / И. Е. Мясников, Н. Р. Спиричева, С. И. Тимошенко. – Екатеринбург : Изд-во Урал. ун-та, 2017.– 104 с.

# ПРИЛОЖЕНИЕ А

## (обязательное)

### Исходный код программы

#### А.1 Класс App

```
package ru.vasilevsky;
import ru.vasilevsky.document.DocumentException;
import java.io.*;
public class App {
    /**
     * Точка входа программы, в методе используется {@link Checker},
     * в который в последствии передается управление.
     * @param args параметры запуска програмы
     */
    public static void main(String[] args) {
        if (args.length == 0) {
            System.err.println("Пожалуйста, укажите файл!");
        } else {
            try {
                Checker checker = getChecker(getInputData(args[0]));
                String result = getResult(checker);
                if (args.length > 1) {
                    FileWriter fw = new FileWriter(args[1]);
                    fw.write(result);
                    fw.close();
                } else {
                    System.out.println(result);
                }
            } catch (FileNotFoundException e) {
                System.err.println("Входной файл не существует!");
            } catch (DocumentException e) {
                System.err.println("Некорректный входной файл!");
            } catch (IOException e) {
                System.err.println("Ошибка записи в файл!");
            }
        }
    }
    /**
     * Получение источника данных.
     * @param fileName - имя файла.
     * @return поток ввода, содержащий документы.
     * @throws FileNotFoundException - если файл не найден.
     */
    private static InputStream getInputData(String fileName) throws
FileNotFoundException {
        return new FileInputStream(fileName);
    }
    /**
     * Получение экземпляра класса решающего задачу.
     * @param data - поток ввода, содержащий документы.
     * @return экземпляр класса решающего задачу.
     * @throws DocumentException - некорректный формат файла.
     */
    private static Checker getChecker(InputStream data) throws
DocumentException {
        Checker checker = new Checker();
    }
}
```

```

        checker.init(data);
        return checker;
    }
    /**
     * Получение решения задачи.
     * @param checker - экземпляра класса решающего задачу.
     * @return строка содержащая решение задачи.
     */
    private static String getResult(Checker checker) {
        return String.format("%d %d", checker.countDeadLinks(),
        checker.countNotResolved());
    }
}

```

## A.2 Класс Checker

```

package ru.vasilevsky;
import ru.vasilevsky.document.*;
import java.io.InputStream;
import java.util.*;
/**
 * Класс, реализующий алгоритм задачи о проверке веб-страниц
 */
public class Checker {
    private List<Integer> G = new ArrayList<>();
    private List<Boolean> real = new ArrayList<>();
    /**
     * Метод, инициализирующий матрицу смежности по содержимому файла.
     * Использует {@link DocumentReaderImpl}.
     * @param is поток ввода входных данных (веб-страницы)
     * @throws DocumentException
     */
    public void init(InputStream is) throws DocumentException {
        DocumentReader reader = new DocumentReaderImpl(is);
        //строим матрицу смежности графа
        List<String> names = new ArrayList<>(reader.getDocCount());
        while (reader.hasNext()) {
            Document doc = reader.next();
            int i;
            if (!names.contains(doc.getName())) {
                names.add(doc.getName());
                real.add(true);
                G.add(0);
                i = G.size() - 1;
            } else {
                i = names.indexOf(doc.getName());
                real.set(i, true);
            }
            int x = 0;
            List<String> refs = doc.getRefs();
            for (String ref : refs) {
                if (!names.contains(ref)) {
                    names.add(ref);
                    real.add(false);
                    G.add(0);
                }
                x |= 1 << names.indexOf(ref);
            }
            G.set(i, x);
        }
    }
}

```



```

    }
    /**
     * Метод подсчета ссылок на несуществующие документы.
     * @return количество ссылок на несуществующие документы
     */
    public int countDeadLinks() {
        int c = 0;
        for (boolean r : real) {
            if (!r) c++;
        }
        return c;
    }
    /**
     * Метод подсчета недостижимых документов.
     * @return количество недостижимых документов
     */
    public int countNotResolved() {
        //строим матрицу достижимости
        for (int k = 0; k < G.size(); k++) {
            for (int i = 0; i < G.size(); i++) {
                int x = G.get(i);
                for (int j = 0; j < G.size(); j++) {
                    if ((G.get(i) & (1 << k)) > 0 && (G.get(k) & (1 << j)) >
0) x |= 1 << j;
                }
                G.set(i, x);
            }
        }
        int c = 0;
        for (int i = 1; i < G.size(); i++) {
            if ((G.get(0) & 1 << i) == 0 && real.get(i)) c++;
        }
        return c;
    }
}

```

### A.3 Класс DocumentException

```

package ru.vasilevsky.document;
public class DocumentException extends Exception {
    public DocumentException() {
    }
    public DocumentException(Throwable t) {
        super(t);
    }
    public DocumentException(String msg) {
        super(msg);
    }
    public DocumentException(String msg, Throwable t) {
        super(msg, t);
    }
}

```

### A.4 Класс Document

```

package ru.vasilevsky.document;
import java.util.List;
/**
 * Модель документа.
 */
public class Document {

```

```

private boolean resolved;
private List<String> refs;
private String name;
public boolean isResolved() {
    return resolved;
}
public void setResolved(boolean resolved) {
    this.resolved = resolved;
}
public List<String> getRefs() {
    return refs;
}
public void setRefs(List<String> refs) {
    this.refs = refs;
}
public void setName(String name) {
    this.name = name;
}
public String getName() {
    return name;
}
@Override
public boolean equals(Object obj) {
    if (obj == null || !(obj instanceof Document)) return false;
    Document doc = (Document) obj;
    return name.equals(doc.name);
}
@Override
public int hashCode() {
    return name.hashCode();
}
}

```

## A.5 Класс DocumentReaderImpl

```

package ru.vasilevsky.document;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
/**
 * Класс реализующий интерфейс {@link DocumentReader}
 */
public class DocumentReaderImpl implements DocumentReader {
    private static final Pattern pattern = Pattern.compile("<((\\w+)\\s*(\\w+|='.*')?)*>");
    private int docCount = 0;
    private Scanner scanner;
    private Matcher matcher;
    private String curStr;
    private String prevStr;
    /**
     *
     * @param is - поток ввода, содержащий данные для парсинга
     * @throws DocumentException
     */
    public DocumentReaderImpl(InputStream is) throws DocumentException {

```

```

        scanner = new Scanner(is);
        try {
            matcher = pattern.matcher("");
            docCount = scanner.nextInt();
        } catch (Exception e) {
            throw new DocumentException(e);
        }
    }

    public boolean hasNext() {
        do {
            prevStr = curStr;
            if (!scanner.hasNextLine()) return false;
            curStr = scanner.nextLine().toUpperCase();
            matcher = Pattern.compile("<HTML>").matcher(curStr);
        } while (!matcher.find());
        return true;
    }

    public Document next() throws DocumentException {
        Document document = new Document();
        document.setName(prevStr);
        List<String> refs = new ArrayList<>();
        m:
        do {
            matcher = Pattern.compile("<A\\s+HREF='(.*)'>|(<END>)").matcher(curStr);
            while (matcher.find()) {
                if (matcher.group(2) != null) {
                    break m;
                } else {
                    refs.add(matcher.group(1));
                }
            }
            prevStr = curStr;
            if (!scanner.hasNextLine()) break;
            curStr = scanner.nextLine().toUpperCase();
        } while (true);
        document.setRefs(refs);
        return document;
    }

    public int getDocCount() {
        return docCount;
    }
}

```

## A.6 Интерфейс DocumentReader

```

package ru.vasilevsky.document;
import java.util.Map;
/**
 * Интерфейс чтения входных данных, как документов {@link Document}
 */
public interface DocumentReader {
    /**
     * @return следующий документ {@link Document}
     * @throws DocumentException
     */
    Document next() throws DocumentException;
    /**
     * Проверяет наличие документа {@link Document}.
     * @return правда/ложь
     */
}

```

```

    boolean hasNext ();
    /**
     * Подсчитывает количество документов.
     * @return количество документов
     */
    int getDocCount ();
}

```

## A.7 Класс CheckerTest

```

package ru.vasilevsky;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import ru.vasilevsky.Checker;
import ru.vasilevsky.document.DocumentException;
import java.io.FileNotFoundException;
import java.io.InputStream;
/**
 * Класс, тестирующий корректность работы класса {@link Checker}.
 */
public class CheckerTest extends Assert {
    private Checker checker;
    /**
     * Метод инициализирующий окружение.
     * @throws DocumentException
     */
    @Before
    public void init() throws DocumentException {
        InputStream is =
getClass().getResourceAsStream("/ru/vasilevsky/test.t");
        checker = new Checker();
        checker.init(is);
    }
    /**
     * Метод проверяющий соответствие количества мертвых ссылок
     * в тестовом входном файле с вычисленным.
     */
    @Test
    public void testCountDeadLinks() {
        assertEquals(2, checker.countDeadLinks());
    }
    /**
     * Метод проверяющий соответствие количества недостижимых документов
     * в тестовом входном файле с вычисленным.
     */
    @Test
    public void testCountNotResolved() {
        assertEquals(1, checker.countNotResolved());
    }
}

```

## A.8 Класс DocumentReaderImplTest

```

package ru.vasilevsky;
import org.junit.Assert;
import org.junit.Test;
import ru.vasilevsky.document.DocumentReaderImpl;
import ru.vasilevsky.document.Document;
/**

```

```

    * Тестирование класса реализующего интерфейс парсинга документа {@link
    DocumentReaderImplTest}
    */
    public class DocumentReaderImplTest extends Assert {
        /**
         * Тестирование обхода всех документов тестового файла.
         * @throws Exception
         */
        @Test
        public void testNext() throws Exception {
            DocumentReaderImpl docReader = new
            DocumentReaderImpl(getClass().getResourceAsStream("/ru/vasilevsky/test.t"));
            int i = 0;
            while (docReader.hasNext()) {
                docReader.next();
                i++;
            }
            assertEquals(4, i);
        }
    }
}

```